BROWN UNIVERSITY
Department of Computer Science
Master's Project

CS-94-M5

"Three Dimensional Morphing Using an Adaptive Oriented Particle System"

by

Matt Corkum

Three Dimensional Morphing Using an Adaptive Oriented Particle System

Matt Corkum

Harry Mamaysky

Department Of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science at Brown University

February 1994

[illegible]

Professor John F. Hughes
Advisor

Three Dimensional Morphing Using an Adaptive Oriented Particle System

Matt Corkum mcc@cs.brown.edu
Harry Mamaysky hm@cs.brown.edu

Dept. of Computer Science
Brown University
Box 1910
Providence, RI 02912

Abstract

This paper presents an automated 3D surface morphing algorithm. Our system requires minimal user input to generate an automatic correspondence for morphing between 3D surfaces. Our surfaces are not constrained to genus zero and we can smoothly morph from one genus to another. Our morphing algorithm is also not limited by the surface topology required during the eventual rendering. Thus our system addresses quite easily the apparent changing surface topologies that arise during a metamorphosis between different genus. We are able to do this because our internal representation is an adaptive heterogeneous oriented particle system (HOPS); HOPS is a topology-free physically-based way to model 3D surfaces. This representation frees us from most of the traditional morphing problems. Our system can generate a polygonal model using a modified Delaunay triangulation algorithm.

1.0 Terminology

We will use the following terms, taken mostly from [Kent92].

Table 1: terms

term	refers to
object	an entity having 3D surface geometry
shape	the set of points in object space that comprises the object's surface
model	any complete description of the shape of an object
topology	the vertex/edge/face network of a polyhedral model
geometry	an instance of a topology for which the vertex coordinates have been specified
genus	number of holes in a object
surfel[a] (oriented particle)	a small 2D planar disc having a position and normal
destination surface	the surface which we are attempting to transform the initial surface into
final surface	this is the end frame of a morphing sequence which will be a close approximation to the destination surface
sweep line	the line defined by the center of a chosen particle and a point in the plane of this particle

a. The term surfel was coined by [Szel91].

Note 1: A single *object* may have many different *models* that describe its shape.

2.0 Introduction

This paper presents an automated real time 3D surface morphing algorithm. In general, 3D morphing involves computing an object-space correspondence between two models and then computing an object-space interpolation based on this correspondence. A good correspondence is essential in generating a 3D morph. It is also desirable to have a flexible surface representation which avoids directly addressing the unknown and arbitrary topologies that could be generated during interpolation. Our experience with oriented particle systems [Szel91] suggested that they might provide the necessary foundation for representing these arbitrary topologies. Our surface model is internally represented by a heterogeneous oriented particle system (HOPS); HOPS is a topology-free way to model 3D surfaces. It is based on a physical model that shares characteristics of deformable surface models and particle systems. HOPS surfaces are represented by oriented particles, hereafter called "particles," whose behavior is affected by long range attraction forces, short range repulsion forces, and other surface potential functions. Thus as we attempt to morph an initial HOPS surface into a destination HOPS surface, we rely on these forces to keep the intermediate HOPS surfaces and the final HOPS surface well behaved. Later we discuss how to transform our internal HOPS surface model into a traditional polygonal model. [See "Triangulation of the Oriented Particle System" on page 59.] So in essence we do not restrict our thinking to interpolating polygons directly; however our collection of algorithms can visually interpolate between polygonal models of vastly different topologies.

The overall structure of a morph is as follows: first a user generates an initial and a final HOPS surface [See "The Details of Our Particle System" on page 32.]; next the user specifies a seed correspondence by selecting at least one particle and sweep line in both systems[1] [See "The Correspondence Algorithm" on page 19.]. From this seed set of user indicated correspondence pairs we generate an initial correspondence between the two surfaces. This initial correspondence represents a mapping of the initial surface onto the destination one. Our morphing algorithm will either grow or shrink the initial surface until it has roughly the same number of particles as the destination surface. This is

1. Refer to the Terminology Section for what we mean for a "sweep line."

accomplished by introducing external forces into the system which represent a linear interpolation between particles in the correspondence pairs that define the morph. Thus the initial surface undergoes a series of smooth transitions through intermediate stages until it basically settles into the final surface.

As our goal is to morph in real time, we also address issues of complexity. We considered several algorithms for reducing spatial complexity; we eventually chose a spatial subdivision algorithm [Turk90]. This algorithm is able to insert or update a particle in its spatial hash table in constant time. Likewise searching for the neighbors of a particle can be accomplished in constant time.

We have two approaches for rendering. The default real time rendering technique involves rendering the intermediate HOPS surface as being composed of many surfels (oriented particles). We can also generate and render polygons from our internal HOPS representation. Therefore, in essence we can morph between polygonal representations of different genera; but our approach is not restricted to thinking of morphing polygons directly.

3.0 Background

This section presents an overview of the fields of morphing and of particle systems.

3.1 Morphing

3.1.1 History

Recently, image based morphing has achieved widespread use in the entertainment industry. Variations in image based techniques have created astonishing special effects for commercials, movies, and music videos. Basically, image morphing involves specifying functions that map a point (pixel) from a initial image to a point (pixel) in a final image. By simultaneously interpolating the position and color of corresponding pixels in the initial and final images, intermediate frames can be generated. Then when viewed in a forward or backward sequence the images appear to transform from one image to the other in a smooth fashion.

One of the first morphing applications occurred in a 1941 Victor Fleming version of "Dr. Jekyll and Mr. Hyde" when an image fade of actor Spencer Tracey occurred between "Dr. Henry Jeckyll" and "Mr. Hyde." This primitive metamorphosis was basically a fade from one image to the other from a fixed view point looking at a fixed location in space. The advantages of a fade are that blending a linear amount of one image with another is simple and is computationally inexpensive (one add and one shift per pixel in an image). As Spencer Tracy with two sets of makeup is basically the same person when viewed from a fixed point of view, this fade was quite successful. However if the source and destination images in a morph are not extremely similar a fade will look unconvincing. Because features in images are not always in exactly in the same location, nor are they the same size, nor may they be at the same orientation, the resulting intermediate images may have ghosting of the various differences. To overcome this drawback in 2D morphing, image warping is applied in addition to image fading, to transform important features of one image to that of the other. At a high level, an image warp appears to stretch and squeeze parts of image to allow key features to align in the computed intermediate frames.

3.1.2 Image Warping

Image warping can be thought of placing an image on a "rubber sheet." We can then stick pins in the rubber sheet, and we can move these pins, thus stretching the rubber sheet in local areas. This has the net effect of dragging features and the surrounding parts of the image. In effect we can warp the image on the rubber sheet into another image. Instead of only using pins or points, we can use "rods" which represent a collection of points. This provides an even more expressive power. If we also allow the lengths of the rods at the start and the finish of the morph to be different, we can interpolate linearly, which further extends the net expressive power. The overall effect of a smooth image blend and image warp is a smooth metamorphosis from one image to another. This is basically the approach employed by Thaddeus Beier and Shawn Neely [BEIR92] in their feature based metamorphosis. The drawbacks of this 2D approach are that it requires lots of disk space, it is not real time, and establishing this feature correspondence is tedious.

3.1.3 Image Based Morphing

We have implemented Beier's morphing algorithm and it works very well when a good user-specified correspondence between images exists. With this approach, the user is required to identify all important features to warp from one image to another. This user specified correspondence, although very expressive, is tedious to establish. The user of this image-based algorithm has the flexibility to decide the actual number of features to equate through the above described; the user must encapsulate each feature in each image by the same number of lines. The actual lengths and orientations of the lines act to warp the image features as described in the previous section. This 2D image based solution results in very detailed images from a fixed view point. As the image based morph is compute intensive, the user is required to wait a for computations of the intermediate frames prior to displaying an animation (consisting of flipping through these intermediate frames) to see if the resulting morph is satisfactory.

3.1.4 Problems with Image Based Morphing

2D image morphing [Beir92] is now well established and much used in both television commercials and movies; however it has limitations. The first is dependence on images rather than on underlying models. The second is the large amount of disk space required to store the many intermediate images required for smooth animation.

3.1.5 Model Based Morphing

Researchers investigate 3D based morphing because it provides several advantages over 2D image based techniques. First is that it is object-based rather than image-based, the second is the ability to animate the objects independently of the metamorphosis transformation, and the third is the minimal amount of memory required for 3D compared with the 2D image based approaches. Image-based techniques require lots of disk space. An example of this is that a short 15 second full screen 1K by 1K resolution image with 60 frames/second playback would require 900 Meg of disk to store the morph 898 resulting intermediate frames and the initial and final frames. Computing this morph might take as little as a few hours to a day or so.

[Kent92] present an algorithm for computing a transformation that merges topological structures of a pair of 3D polyhedral models into a common vertex/edge network. Once this transformation has occurred, then a correspondence between vertices allows for an easy computation to interpolate from one model to another. The basic algorithm requires that the 3D polyhedra models be preprocessed to generate two new 3D models that have the same shape as the original ones, but allow an easier correspondence to be generated. The previous efforts of the authors of this paper, [Pare91] described an earlier version of their algorithm. The earlier approach could only work for star-shaped[1]polyhedral solids. This algorithm allowed for a trivial mapping to a unit sphere as a direction vectors to the locations on the star shaped polyhedra could simply be normalized.

They describe that a 3D morph is basically two steps, the first step being the generation of the correspondence and the second being the interpolation of the corresponding components. They further mention that the two steps are interrelated. They also mention that a good correspondence is the key to a good 3D morph and we agree. They restrict their approach by working with Euler-valid, genus 0, polyhedral objects and they felt that their correspondence algorithm for genus 0 was the main contribution in the paper. They make the claim that techniques that make use of the topology and geometry of the models tend to yield better results. This may be true for a subset of genus 0; however as of the writing of their paper no known techniques existed for 3D morphing above genus 0, or between different genus, or between vastly different topologies such as the morphing within genus 0 of one sphere to two spheres. Thus, their statement may need to be revised. They mention that they tried physically-based methods, but their efforts were unsuccessful. They ran into troubles with cycles and non-convex shapes being produced. They also mention that they still had troubles with arbitrary shapes and with models that had more than 1000 vertices due to numerical inaccuracies. Also, self-intersecting polygonal models and non-planar polygons caused problems in their system. Their system also had very little user control over the morph. They did present some very impressive intermediate frames of morph

1. A star-shaped polyhedral object is one for which some interior point, p, exists such that a semi-infinite ray originating at p intersects the surface of the object at exactly one point. Note that the set of convex polyhedra is a proper subset of the set of star-shaped objects [Kent91]

attempts of busts of heads and columns. [Kent 92] failed at physically-based morphing attempts, but the authors mention that [Hauman88] attempted to treat each vertex of the topology as a spring and the edge as a spring to generate a physical based model to treat the surface as a flexible object. In conclusion, this is an excellent reference [Kent92] for those interested in morphing within a subset genus 0 with models having similar topology.

Many authors have suggested that morphing may provide insight into metamorphosis and shape blending. [Kent91] wrote that the metamorphosis of one object to another may suggest a possible development process, such as a tadpole growing into a frog. [Chen89] presents a transformation algorithm for piecewise linear 2D contours and then the paper very briefly addresses an extension for 3D lofted objects. His interest in morphing arose from wanting to obtain a more aerodynamic-looking car by interpolating between a car and a teardrop. Chen also investigated shape interpolation by investigating and averaging similar designs. [Beth89] describes an algorithm that adds degenerative vertices and faces until a common topology between two models is created prior to computing the correspondence between models.

Others have attempted 3D morphing.[Peachy85] projects each surface point onto the *xy*-plane by ignoring its *z* component. Obviously this is bad for pairs of points with the same x and y components, since they map to the same 2D coordinates, and thus some portion of the image may not be used. [Bier89] produced a more evenly distribution mapping based on first encapsulating the object inside an object such as a sphere. Surface points are then projected onto the encapsulating object, for star shaped or convex polyhedra, the surface is projected bijectively onto the encapsulating object. If the polyhedra do not fall into one of these restrictive classes, then this technique has troubles with many-to-one and one-to-many mappings. [Knep92] presents an algorithm for mapping between a 3D surface and a connected subset of the *uv* plane. One of the applications mentioned in the paper is the ability to wrap a surface patch around an object and to transform one surface into another.

3.1.6 Summary of Morphing

Several attempts at 3D morphing have been presented, but they usually only work for a limited set of polyhedra with restrictive topology and geometry within genus 0. On the other hand, successful techniques have been developed for 3D morphing between object models [Kent91, Kent92, Chen89, Knep92, Terz89]. However many of these techniques have one or more of the following flaws: (1) The models consist of a highly restrictive class of objects which can be morphed (i.e star shaped polyhedra), (2) The intermediate models have a high distortion (i.e., the mapping of a small region in one model to a large region in the other model), (3) The intermediate models have faces which fly apart or self intersect during the intermediate metamorphosis due to a lack of an underlying physical model or a lack of topological information. (4) The object models are restricted to the class of objects of genus zero. (5) The resulting intermediate models ignore any sort of physical properties of the object.

3.2 Deformable Surface Models

[Barr84] does an excellent job with solid object metamorphosis via hierarchical deformations to bend, twist, taper, or to generally transform solid geometric objects. His approach involves transforming the initial 3D model shape and surface properties into the shape and surface properties of the final destination shape and surface properties. This reference is essential if one intends to model solids for CAD/CAM in hierarchical structures.

[Terz87] models elastically deforming models using elasticity theory. The theories of elasticity are described using time based differential equations to model deforming materials such as rubber, cloth, paper, and other flexible materials. These elastically deforming models respond to forces, constraints, and impenetrable obstacles. These dynamic models are solved using a numerical integrator to create a realistic and dynamic model. This was one of the first references that unified the description of shape and the actual description of motion response of the shape.

3.3 Particle Systems

Particle systems have been used to model natural phenomena such as waterfalls and fire [Reev83] [Sims90]. In these early particle systems the particles are acted on by force fields and constraints; however there is no interaction between the particles, so particles are allowed to intersect with each other without experiencing any forces from other particles in the local area.

Our work borrows heavily from [Szel92]. The authors, Richard Szeliski and Dave Tonnesen were the first to present an oriented particle system. An oriented particle system tends to arrange itself as a surface which favors locally planar or spherical arrangements. They developed potential functions to address particle interaction. To avoid redundancy the specific potential functions are discussed in great detail below in "A Summary of Our Particle System Implementation" on page 12

In addition to 3D Morphing, our OPS system is good at smoothing, filling in, and interpolating sparse data. In order to reduce the number of particles in a model without a loss of fine detail, we can also opt to generate a HOPS model from our OPS model. We wanted to capture the surface with the minimal number of particles. A heterogeneous particle system provides a minimal representation of the surface by using oriented particles of different sizes. Thus areas of the surface that vary greatly in curvature will be represented with many small particles and the areas that are planar[1] will be represented with fewer and larger particles. The heterogeneous particle system model offers a good approximation to a given surface, while reducing the total number of particles necessary for the representation.

3.4 A Summary of Our Particle System Implementation

A particle system generally consists of point masses in space which are acted upon by forces. In our system, each particle additionally has associated with it a normal vector, which gives it an orientation with respect to its neighbors. Four potential functions define the energy experienced by any two particles in the system. The overall system energy is taken to be the sum of the individual particle energies. The system is animated in a way so as to minimize overall system energy. The *Lennard-Jones* potential governs the distance that particle centers want to be away from one another. In other words, all particles will tend to settle into a configuration where the distance between any two adjacent particles is the same as the distance between any other such pair. The *co-planarity* potential leads to sur-

1. Where planar is a user decided allowable threshold for the standard deviation allowed from a given particles point of view.

faces whose rest configuration is a flat plane. This potential asks that particles lie in one another's tangent planes. The *co-normality* potential serves to control surface twisting by requiring that adjacent particle normals all point in the same direction. Finally, the *co-circularity* potential allows for surface areas of constant curvature. This potential achieves its zero state when adjacent particle normals are anti-symmetrical with respect to the vector joining the two particles.

The derivatives of these potentials give us the forces which are experienced by particles in the system. Knowing, from Newtonian mechanics, that $f = ma$ we are able to compute the acceleration experienced by each particle. We animate the system by integrating this acceleration twice in order to obtain particle positions. Additionally, we are able to perform shape transformations by introducing external morphing forces into the system. Each particle has applied to it a *morphing* force which tends to move that particle in the direction required by the morphing algorithm. The external morphing forces affect both particle centers and normals. This will be discussed in greater detail in Section 4.0 on page 14.

Surfaces represented by an oriented particle system at its "rest state" tend to be smooth, and locally planar or constantly curved. Different surface properties may be generated by varying the degree of the contribution of each of the four potentials to overall system energy. Animation occurs because the surface will tend to settle into its rest energy state. Thus when a particle or a group of particles is somehow taken out of a low energy state, the system's overall energy increases. The offending particles will then be pulled or pushed into a lower energy configuration by the forces which they will experience from their neighbors[1]. Generally particle motion tends to be graceful and intuitive, in that the user is quickly able to discern exactly what a particular low energy configuration appears to be.

As mentioned earlier, we have extended the oriented particle system in [Szel91] to allow for different sized, or heterogeneous particles. The advantages of this approach lie in that surface areas of high curvature may be modelled with numerous smaller particles, while flatter surface regions may be modelled with larger, and consequently fewer, particles. This often results in heterogeneous represen-

1. Those particles which are spatially adjacent.

tations of particular objects which have far less particles than their homogenous counterparts. See "Heterogeneous Particles" on page 38. While we believe that heterogeneous particles may be useful for purposes of morphing, we do not currently incorporate them into our morphing algorithm.

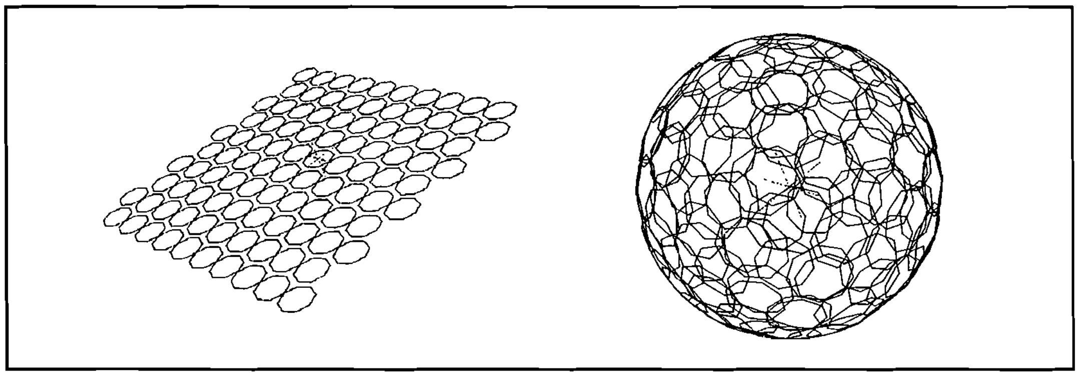

FIGURE 1. An example of surfaces represented by oriented particle systems.

The section entitled, "The Details of Our Particle System" on page 32, presents a more thorough explanation of our system. (Figure 1) shows examples of surfaces represented by oriented particles.

4.0 Morphing

We divide the morphing process into two separate steps which we call the *macro* and the *micro* morphs. The macro morph consists of either a standard or generalized rigid body transformation involving eigenvectors which translates and rotates the initial model in such a way so that it achieves a best fit with the destination model. Once this best fit has been achieved, we perform a micro morph which actually transforms the initial model into an exact copy of the destination one. The micro morph assumes the two models are to be super-imposed in the best fit way, as indicated by the macro morph. The final visual transformation is obtained by combining the macro morph with the micro morph. That is, in the i^{th} frame of the visual morph, the user sees the i^{th} step model in the micro morph in the i^{th} position of the macro transformation. We believe this split up to be the most intuitive approach to a more general 3D morphing problem; furthermore, it affords the user, with only minimal input, a great deal of influence over the appearance of the final morph.

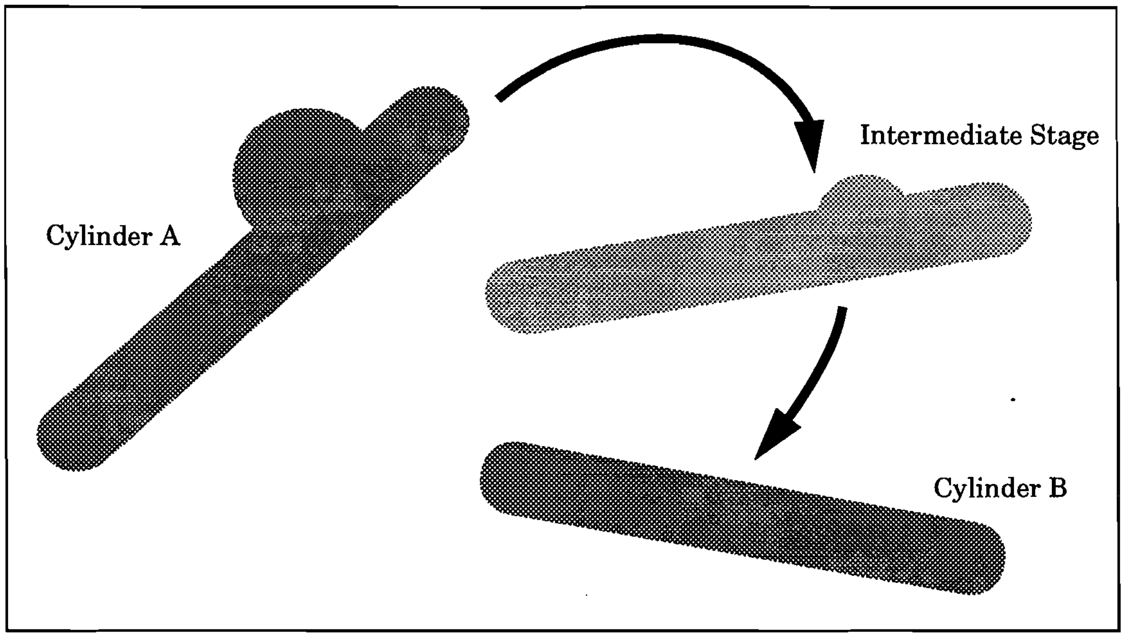

FIGURE 2. A cylinder with a bump and one without a bump. We rotate the first cylinder onto the second one, while the bump slowly goes away.

For example, in transforming cylinder a in (Figure 2) into cylinder b, our approach allows the user to rotate one cylinder onto the other, while throughout this transformation the bump in cylinder a slowly disappears. The intermediate stage shows a cylinder half way between the source and destination ones. The morph of the bump and the rotation can both be clearly seen. Without this two step split-up of the morphing process, the first model would flow into the second one, without the macro-morph-induced rotation, which in this case seems visually intuitive.

4.1 Macro

The user can decide whether it is best to specify the macro level morph by providing some UI selection of particles or by having the system automatically align two models using a technique employing a special symmetric matrix and the resulting eigenvectors.

4.1.1 User Input and Rigid Body Alignment

We implemented UI functionality which allows the user to pick one particle from each particle system which should correspond when the macro morph is complete. We first compute the centroids of the objects and then we compute a macro alignment vector for each system which is basically a normalized vector which emanates from the computed centroid towards the chosen particle in that system. We then align these two centroids via incremental translation and incremental rotation to align these two macro alignment vectors over the course of the user specified number of frames. For each frame we apply a rigid body transform to translate by the total translation vector divided by the number of frames. We then update the centroid to this new location, and then translate to the origin and rotate by the total rotation divided by the total number of frames and then translate back to the new centroid position. There is one major drawback to this approach: the user does not have full control over the alignment, because s/he is only aligning one axis of each object.

4.1.2 Automatic Macro Alignment using Eigenvectors

The second approach does not require user interaction. Instead, this more general solution involves computing the eigenvectors of a symmetric matrix which has the following entries.

$$\begin{bmatrix} \sum_p x \cdot x & \sum_p x \cdot y & \sum_p x \cdot z \\ \sum_p x \cdot y & \sum_p y \cdot y & \sum_p y \cdot z \\ \sum_p x \cdot z & \sum_p x \cdot z & \sum_p z \cdot z \end{bmatrix} \text{ where } \sum_p component \cdot component \text{ is a summing over all particles.} \qquad \text{(EQ 1)}$$

The computed eigenvectors of any symmetric matrix will be distinct and are orthogonal [Foley90]. We can utilize these 3 distinct eigenvectors to align two oriented particle systems. This is accomplished by aligning the largest eigenvector in each system, and then the next largest eigenvector. After we auto align these two largest eigenvectors, we can then align the smallest eigenvectors that are currently not aligned. After making eigenvectors correspond, we need to normalize them and to compute the transformation to align the eigenvectors. We do this by computing a dot product to compute the angle to rotate the largest two eigenvectors. We then compute the same ordered cross product in each system to compute the third axis for both systems. As the initial handedness of the two systems may be different we may need to perform an inversion of a direction of an eigenvector and possibly rotate 180 degrees to ensure that the two systems will align properly. One use of this more general approach is noticed when we have two surfaces which are initially identical. If we first arbitrarily translate and rotate one surface about some axis while leaving the other surface alone, then the computed eigen-based solution that aligns the two systems in an affine manner is exactly, within the constraints of numerical accuracy, the same rotation matrix that initially rotated the surface in space.

4.2 Micro

The micro morph itself consists of a two step process. The first step is to establish a correspondence between the initial and destination particle systems. The second step involves interpolating between the two models, or intelligently moving the particles from the source system to the location of their corresponding particles in the destination system.

4.2.1 The User-Specified Initial Correspondence

The user specifies the initial particle correspondence between the initial and destination particle systems. The initial correspondence consists of an arbitrary number of correspondence pairs, as selected by the user. Clearly by selecting more initial correspondence pairs, the user will have more control over the appearance of the micro morph. While the upper limit on the number of correspondence pairs is the number of particles in the smaller of the two systems, we need at least one user specified correspondence pair to automatically compute a correspondence.

In order to select a single correspondence pair, the user first selects a particle in the initial model. Then the view of the model is rotated in such a way so that the camera faces the selected particle along that particle's normal in order to allow the selection of another point in that particle's plane. This point serves to define a sweep line, which is simply any vector emanating from the particle center and passing through the user selected point which lies in the plane of that particle. The correspondence algorithm uses this sweep line as an indication of the order in which the neighbors of the selected particle will be entered into the correspondence. Once this particle and sweep line have been selected in the initial system, the user repeats the process for the destination system. This pair of choices of particle and sweep line constitutes a single correspondence pair. Initial correspondence pairs are generated until the user is satisfied that the correspondence algorithm has been given enough clues as to what it should do.

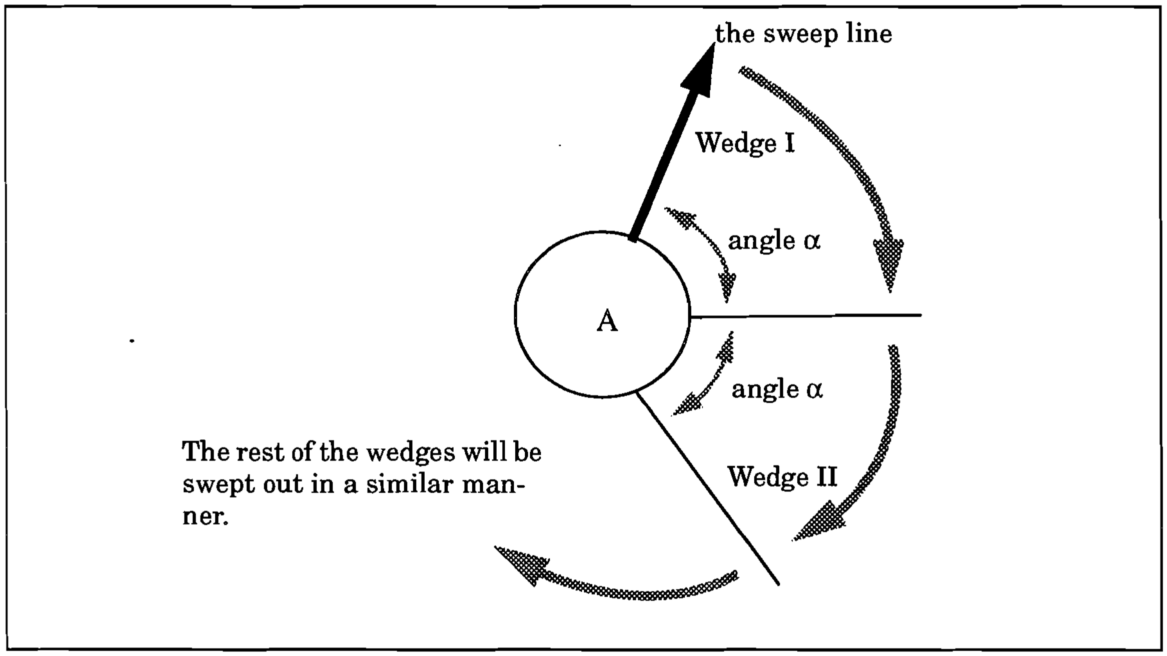
the sweep line
Wedge I
angle α
A
angle α
Wedge II
The rest of the wedges will be swept out in a similar manner.

FIGURE 3. A particle with its sweep line highlighted. The sweep line is used to divide the space around the particle into wedges as indicated. Any neighbor of A within Wedge I will be enterred into wedge 1 of the partition. Any neighbor is Wedge II will be enterred into wedge 2 of the partition. And so on. Notice that each wedge is swept out at an angle of α.

The function of the sweep line is to indicate the order in which the neighbors of the first particle in a correspondence pair will be matched up with the neighbors of the second particle in the correspondence pair. Keep in mind that the first particle in each correspondence pair comes from the initial system, and the second particle comes from the destination system. On a high level, the correspondence algorithm partitions the neighbors of each particle in a correspondence pair into some arbitrary number of wedges (e.g. 6 wedges), with the first wedge lying directly to the right of the sweep line, the second lying directly to the right of the first, and so on (i.e., the wedges are swept out clock-wise initial at the user specified sweep line). See (Figure 3). The particles which lie in the first wedge of the first particle (i.e., those from the initial model) are matched up to the particles which are in the first wedge of the second particle (i.e., those from the destination model). These two sets of particles serve to create a new correspondence pair. This process continues until as many particles as possible have been entered into a correspondence pair. The next section describes the correspondence algorithm in greater detail.

We should note at this point that our algorithm computes a correspondence between two particle systems based on user input. In fact, we believe it erroneous to assume that a single correct correspondence between any two particle systems actually exists. Each correspondence leads to a different morph which may please one user and displease another. Likewise, no single correct morph exists between any two systems, as the visual appeal of any given morph is a purely subjective judgement. For this reason, our approach has been not to try to guess what a given user would want to see in a morph, but to allow a user to specify what the morph should look like in as simple a manner as possible.

4.2.2 The Correspondence Algorithm

The correspondence algorithm maintains two correspondence pair lists, the *seed list* and the *newly created correspondence pair* list. The seed list initially contains the user specified correspondence pairs. The new correspondence pairs generated by the seed list are added to the new pair list. The seed list correspondence pairs are then saved as part of the final correspondence. The new pair list becomes the seed list, and the entire process is repeated until no more new correspondence pairs are generated. Any given particle may be a member of only one correspondence pair. Therefore, when a particle is entered into a correspondence pair, we mark that particle as ineligible of being a member of any other correspondence pair.

Recall that a correspondence pair consists of a set of particles from the initial particle system and a corresponding sweep line, and a set of particles from the destination system and their corresponding sweep line. Therefore, a correspondence pair may indicate a correspondence between more than a single particle from a given system. In practice, we found that the number of particles in a particle set of a correspondence pair rarely exceeds three for a particle system consisting of uniform size one surfels.

Given a set a of particles and a sweep line, we establish a *partition* of the neighbors of a into wedges in the following way. We first find the union of all the neighbors of particles in a. We then project the center of each particle in this union onto the plane, P, defined by the average normal of all the particles in a and by the average center of all the particles in a. We then project the sweep line onto this same plane P. We compute the angle between the projected sweep line and the vector from the average center of a to the particle projection point on P. Based on this angle, we place the given neighbor into the corresponding wedge of the partition. A partition may have an arbitrary number of wedges (we used 6), so long as this number is constant for any given run of the correspondence algorithm. Wedge 1 in the partition would contain all reported angles between 0 and 60 degrees. Wedge 2 would contain angles 61 through 120, and so on.

For a given correspondence pair, we compute a partition for its set of particles from the initial system, and a partition for its set of particles from the destination system. We then look at wedge 1 of both partitions. If this wedge is non-empty in both partitions, we create a new child correspondence pair from these two sets of particles. The sweep line for the source set of particles, s, in the newly created correspondence pair is computed by translating the sweep line from the source set of particles from the seed correspondence pair onto the center of s. We then project this vector onto the plane defined by the average normal and average center of the particles in s. This becomes the sweep line for s. The sweep line for the destination set of particles in the child correspondence pair is computed in a similar manner. This procedure is repeated for all wedges in the given partitions. Clearly, the initial system partition and the destination system partition must have the same number of wedges.

4.2.3 The Orphan Particle Problem

A problem with the method as presented above occurs when a wedge in the source partition is non-empty, and the corresponding wedge in the destination partition is. Such a situation results in *orphan* particles which have no correspondence in the destination system. Each orphan particle is marked as being orphaned and the particle set from the source system which gave rise to that orphan is recorded. It is possible that this orphan will get placed into a correspondence pair as a child of a different source particle set at some later time in the correspondence computation. This is preferable to any other option since it will create a valid new correspondence pair. Therefore we wait until correspondence computations stop in the hope that all orphans will get placed as children of some other particle sets from the source system.

Once no more new correspondence pairs can be generated by the correspondence algorithm, we are forced to deal with any particles which are still orphaned. We do so by simply adding the orphaned particles to the particle set from the source system which gave rise to them in the first place.

This is admittedly not an exact solution, as these orphaned particles really have no place to go in the destination system, and are forced to squeeze in, as it were, where ever there is space. Essentially this association gives the orphan particles an idea as to where they should go in the destination system. This is very useful because often times large regions of the source system are orphaned (i.e., consist largely of orphaned particles); this approach allows these regions to be morphed in a fairly intuitive way into the approximately correct areas of the destination system. Furthermore, any orphaned particle may get squeezed out of the morph if it becomes too tightly surrounded by particles which are real members (as opposed to orphaned members) of correspondence pairs. This is discussed further in the interpolation section below.

We should note that once the orphan resolution process has been completed, the correspondence algorithm may once again be used to find new correspondence pairs because particles which are neighbors of the previous orphans now are able to be partitioned and either placed into correspondence pairs right away, or become orphaned themselves. Even so, via orphan resolution, we are able to associate these previously unseen particles in the source system with existing correspondence pairs to at least suggest minimally where these particles ought to go in the morph. Again, these particles may disappear at some later time in the morph if they become surrounded too tightly by real correspondence pair members.

4.2.4 Interpolation

Once we have completed the correspondence computations we proceed to the interpolation part of the micro morph. For every correspondence pair we perform the following 4 step computation:

> (1) Compute the vector between the center of the source set and the center of the destination particle set.
>
> (2) Compute the difference vector between the average normal of the source particle set and the average normal of the destination particle set.
>
> (3) Compute the difference vector between the source sweep line and the destination sweep line.
>
> (4) Scale each of the above vectors by the same scalar ε so as to yield vectors which have similar magnitudes to the forces generated by the derivatives of the potential functions in the particle system. Currently we have $\varepsilon = 0.03$.

We update the source system sweep line of the correspondence pair in question as follows

$$s_{new} = \|s_{old} + \Delta s\| \qquad \text{(EQ 2)}$$

For every particle in the source set of the correspondence pair in question we update the center c and normal n vectors as follows:

$$c_{new} = c_{old} + \Delta c \qquad \text{(EQ 3)}$$

$$n_{new} = \|n_{old} + \Delta n\| \qquad \text{(EQ 4)}$$

Essentially, the vectors from step (4) act simply as new forces which are introduced into the particle system as a result of the morph. Thus, on a high level, a morph consists simply of introducing forces into the system which push the source system towards the destination system. Once the morph forces have been added to a given system at some time step t, we proceed to compute the forces produced by inter-particle interactions and perform Euler integration as described in previous sections. This allows various surface properties effected by the system's potential functions to show up during the morph. Furthermore, attractive forces from the Lennard-Jones potential exerted by particles in correspondences tend to pull along particles outside of correspondence pairs, thus allowing for morphing to take place even though a full correspondence does not exist.

It is interesting to note the effect of ε is step (4) of the morph forces computations. Higher values of ε place more emphasis on the morphing forces in the system than do lower values. In the former case, morphs tend to proceed in a way less dependant on surface properties. This is expected as the morph forces tend to outweigh the system dynamics, thus bringing about morphs which tend to look more like linear interpolations from the initial model to the destination one. On the other hand, by lowering the value of ε we are able to de-emphasize the value of the morph forces. Thus surfaces deformations tend to adhere more strictly to surface properties as defined by the interactions of the surface potentials. For example, in morphing a single sphere into two smaller spheres, a high choice for the value of ε will tend to produce a morph where the larger sphere rips somewhat abruptly before being transformed into the two smaller spheres. Whereas a smaller choice for the value of ε will allow the larger sphere to be pulled apart much like a piece of rubbery taffy, with the middle slowly thinning out, until the rip finally occurs. (Figure 4) illustrates this point.

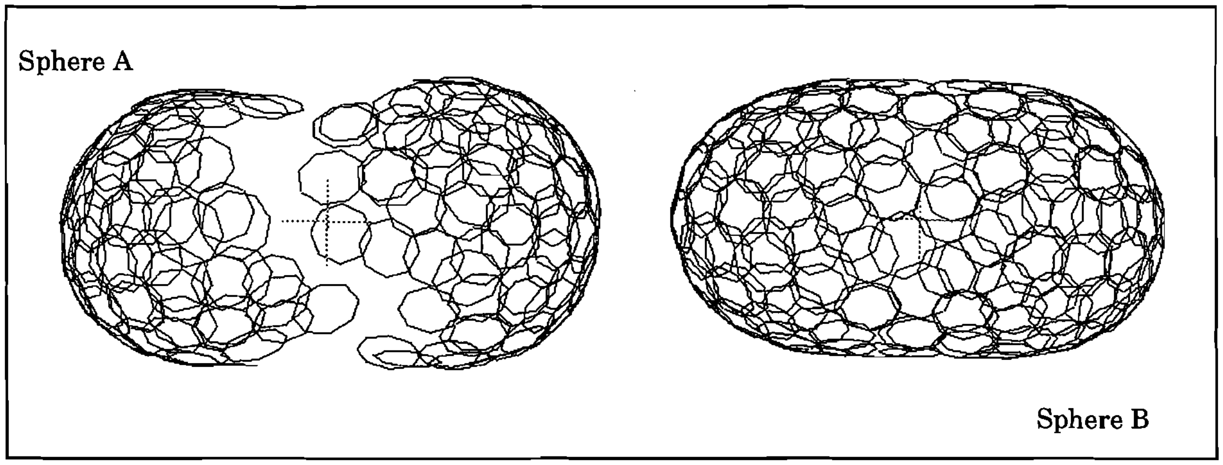

FIGURE 4. Two deformations which began with the same sphere as a initial system. Sphere A pulls apart abruptly. While sphere B, with a smaller value for ε, shows the stretching effect.

4.2.5 Particle Generation

Often the source and the destination particle systems which are to be morphed do not contain the same, or even a similar, number of particles. Despite this, we would still like to be able to morph between the two systems. This section addresses the problem of the source system having fewer particles than the destination system. On a high level, as the morph proceeds, we would like to create new particles in the source system as soon as the space to contain them becomes available. Furthermore, we would like to make sure that these particles become a part of a real correspondence with particles from the destination system.

Our temporal approach to particle growing during a morph sequence is as follows. First we establish a correspondence between the initial and the destination morph systems. Since the initial system has fewer particles than the destination one, this will essentially map the former system onto parts of the latter one. We begin the interpolation part of the morph sequence, and move the initial system particles towards their corresponding particles in the destination system. For every correspondence pair in the correspondence pair list, we establish a partition of the source and destination particle sets. If it is the case that some wedge i in the source partition is empty and the wedge i in the destination partition is not, and the destination wedge i contains at least one particle which is not already a member of a correspondence pair, we attempt to grow a particle in the space represented by wedge i in the source system. If space exists for this potential new particle in the source system, we create a new correspondence pair which contains in its source particle set the newly created particle, and contains in its destination particle set the particle in the destination partition wedge i which had not been a member of a correspondence pair.

Intuitively, we treat each source particle in a correspondence as a marker into the destination system. Thus we attempt to make the neighbors of a given source particle as similar as possible to the neighbors of the corresponding destination particle. The process of adding new particles stops when every particle in the destination system has been entered into a correspondence pair. At this point, the source and destination systems have a very similar, if not equivalent, number of particles. In addition to this, the newly created source particles are all real members of correspondence pairs, and have neighbors which correspond in a properly oriented way to the neighbors of the destination particles in their correspondence pairs. This procedure leads to very intuitive looking morphs.

We should note that placing a newly created source particle into an empty wedge i of a source partition is not a trivial task. The particle must be placed in such a way so as not to disrupt the source system (i.e., placed in a way that will not raise overall system energy by a significant amount) by either being above or below the existing surface at the insertion point, or by being inserted into an already filled space. The latter problem is easily remedied by simply making sure that the surrounding space of the source wedge i is sufficiently empty. The former problem, that of aligning the newly created particle with the local surface, is somewhat more tricky. Our solution is as follows.

We take the sweep line of the source particle set (which is what defines the wedges of the source partition as already described) and rotate it around the average normal of the set of source particle by an angle which would place the rotated vector into wedge i of the partition. Since sweep lines are normalized vectors, we then scale this rotated unit length vector by the natural interparticle spacing between the newly created particle and the parent source particle set which spawned the creation of the new particle (i.e., that source set from the correspondence pair whose wedge i had initially been empty). We add this vector to the center of the source set in question to get a point P, which we will use to decide where to place the newly created particle.

We look at the set S of all particles which exists within some distance away from point P (currently we use $r = 2.2 \times rad$ where rad is the radius of the newly created particle). The normal of the new particle is equal to the average normal of the particles in S. We find the center of the particles in S and make this the position of the new particle. This will tend to position the new particle directly in the opening which exists in wedge i of the source partition. We compute the standard deviation of normals of particles within distance r of P as follows

$$\sigma = \sqrt{\frac{1}{n} \sum_{i \in Parts} \left(\overline{norm} - norm_i\right) \cdot \left(\overline{norm} - norm_i\right)} \qquad \text{(EQ 5)}$$

For a nicely behaved surface this serves as an effective measure of local curvature. We translate the newly created particle along its normal by a distance of $\sigma \times rad$. This will tend to align the new particle with the local surface. Of course the procedure will not work in the case that the surface is locally concave (it would be necessary in this case to move a distance of $-\sigma \times rad$ along the particle normal). However, the potential functions we use tend to produce only locally convex surfaces. (Figure 5) illustrates this procedure in two dimensions.

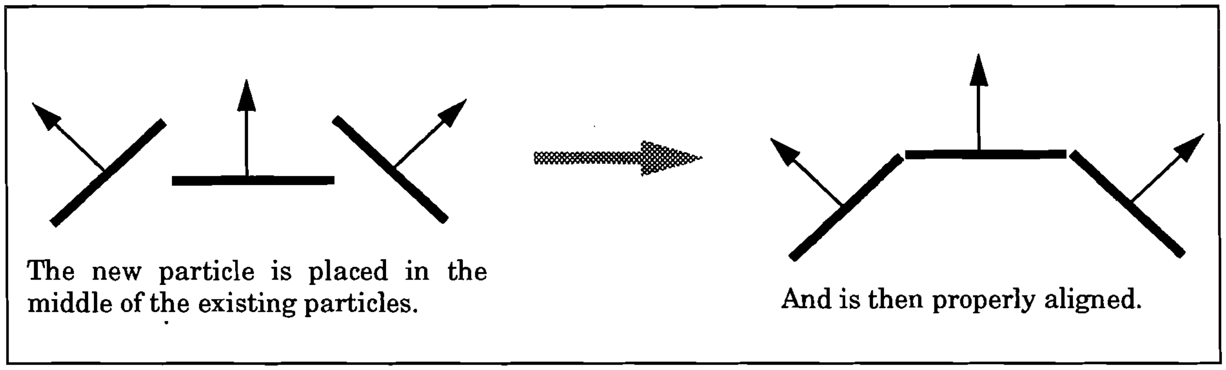
The new particle is placed in the middle of the existing particles.
And is then properly aligned.

FIGURE 5. A particle is placed in the center of the surrounding particle set, and is then moved up to be aligned with the local surface based on the value of the standard deviation of normals, σ. This also illustrates why, in the case of a concave surface, we would have to move in the opposite direction of the new particle normal.

It is also necessary to insure that new particles are created in such a way so as not to disrupt the existing surface. For example, in morphing a torus to a sphere, we do not want to grow particles in the torus' hole. To guard against such situations, we check to make sure that the standard deviation (as presented above) of particle normals in the area into which we are placing a new particle is less than some given threshold[1].

4.2.6 Particle Deletion

When the source system has a greater number of particles than the destination system, we are confronted with the opposite problem of the one we addressed in the previous section. Specifically, we now need to delete particles, instead of adding new ones. In this case we expect that the destination system will have each of its particles in a correspondence pair with a particle set from the source system. Thus any particle in the source system which is a real correspondence pair member (as opposed to having been an orphan) needs to be left alone, as it will be morphed onto its corresponding particle set in the destination system. The orphaned particles, on the other hand, have no place to go in the destination system. We continuously check all orphaned and unmatched (or un-corresponded) particles to see whether they are tightly surrounded by other particles. We define *tightly surrounded* as

1. Currently the threshold we use is 0.7. Note that all the normal vectors are unit length. This renders the number 0.7 significant. For this approach to work, all normals must clearly be of the same length.

having three or more particles within a distance of $\beta \times rad$ from the center of the particle in question; *rad* is the radius of this particle. Currently we have $\beta = 0.4$. If it is the case that one of the orphaned or unmatched particles is in an already densely populated area of the particle system, we simply get rid of this particle and continue the morph. This simple approach seems to work quite well as unwanted particles are squeezed out of the system as the morph proceeds. Furthermore, since all destination particles are mapped onto by particles from the source system, we end up with an exact correspondence at the end of the morphing sequence.

4.3 The Nature of our Morphing Approach

We should note that it is possible that not all orphaned and unmatched particles will be deleted in the course of a morph, some of them may work themselves into the resulting surface and never have an excessive number of neighbors as defined above. It is the nature of our approach that the resultant system at the end of the morph may not be identical to the destination system. This is due to the interactions between morphing forces and system forces. Sometimes these two sets of forces reach an uneasy equilibrium where the morph is trying to take the system into a higher energy surface configuration. Clearly the system resists all such efforts, and we are left with a sort of morphing dead lock where morph forces are doing one thing, and the system forces are doing the exact opposite. In fact we observed this very phenomenon when we initialled tried to morph a sphere into a torus. For a sufficiently low value of ε (the morphing force coefficient) the morph can proceed only until the hole in the torus is ready to form. At this point, system forces refuse to allow the hole formation. Without increasing the value of ε at this point (or proceeding from the very beginning with a higher value) we are stuck in this undesired equilibrium. (Figure 6) illustrates this situation.

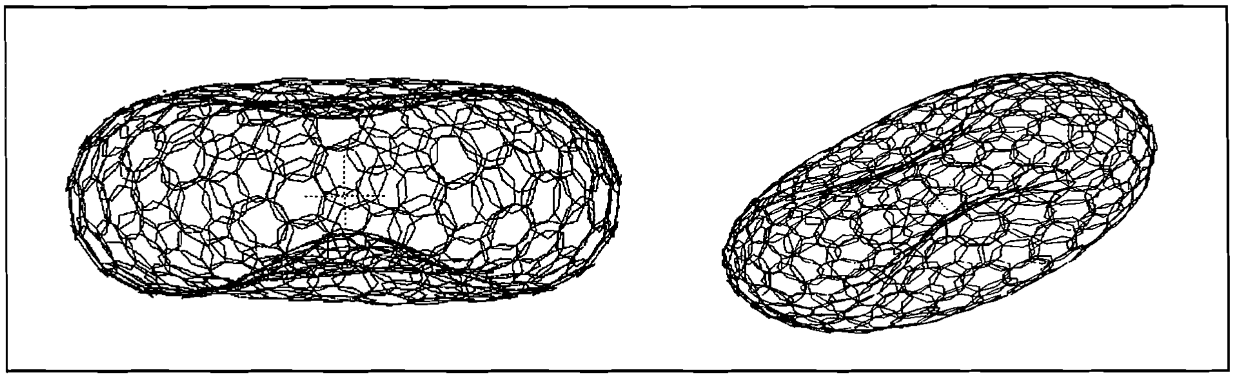

FIGURE 6. At this point, when $\varepsilon = 0.02$, the morph will proceed no further as system forces become exactly opposite in direction and magnitude to the morphing forces. The two views are of the same object.

Thus, our approach is inherently approximate. In may be the case that the resultant system will have a slightly different number of particles than the destination system; or the source system particles which are real correspondence pair members may not be in exactly the same position as their destination system counter-parts. Of course, by increasing the value of ε we are able to reduce the differences between the resultant and destination particle systems.

Another point is that morphs do not tend to be inverses of each other. In other words the morph which takes a sphere into two spheres looks different than the reverse sequence of the morph of two spheres into a single one. Again this is to be expected. When a sphere pulls apart we expect to see a stretching effect which we do not expect to occur when two spheres are pushed together to form one larger one.

Finally, we should note that our approach works only for those objects which we are able to effectively model by using a particle system. Sharp edges are hard to model using the existing potential functions. Therefore we would be unable to morph a real cube into anything as we would be unable to model a real cube with our system. However, an approximation to a geometrically accurate cube can be modelled, and therefore is able to be morphed. The next section shows and discusses some of the morphs which we were able to create.

4.4 Some Morphing Results

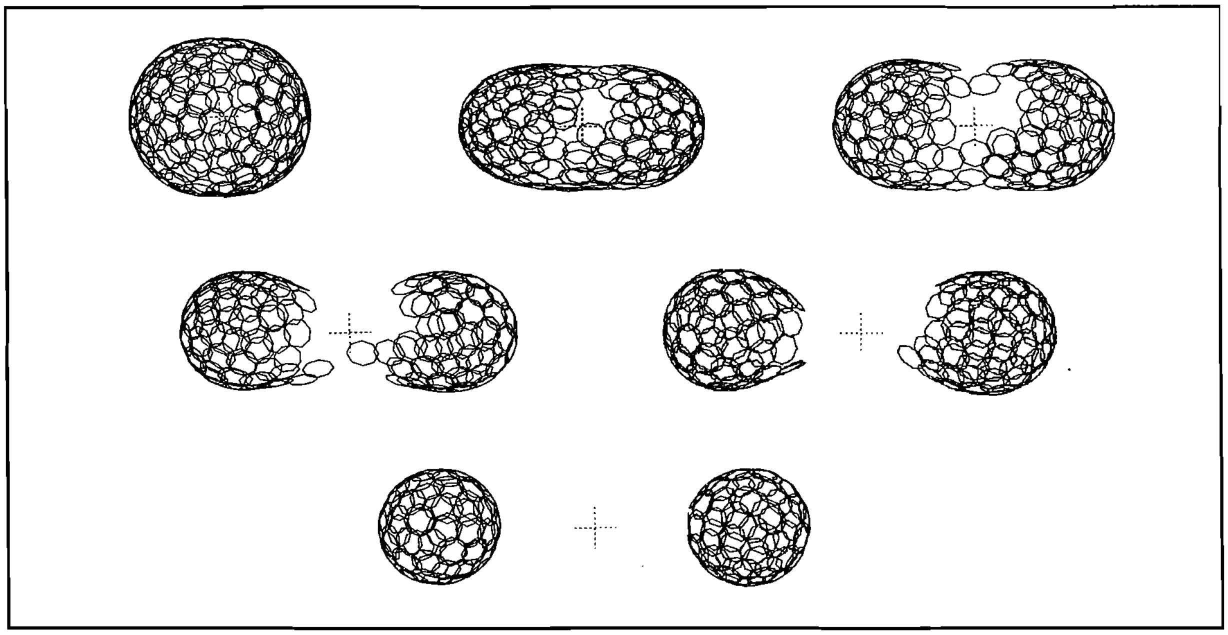

FIGURE 7. A morph from a single sphere to two smaller spheres.

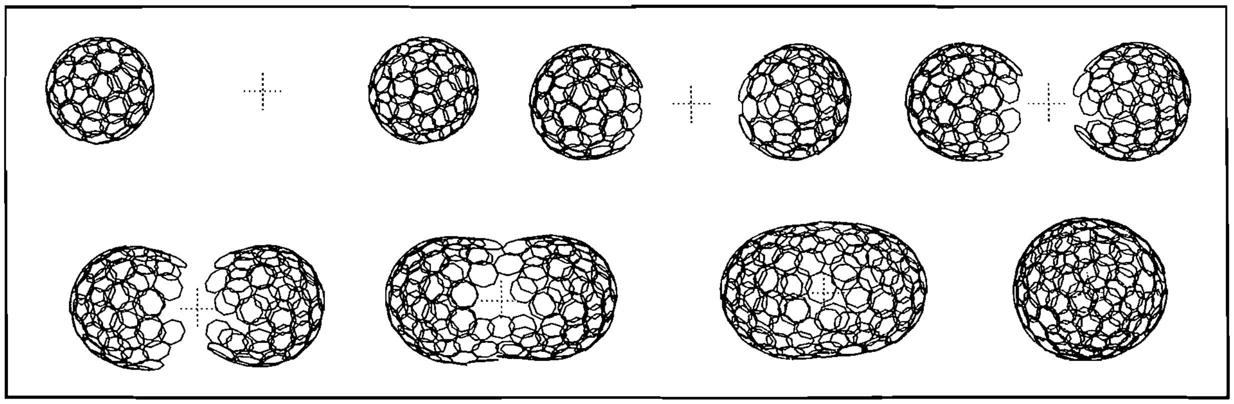

FIGURE 8. A morph from two spheres into a single sphere. Notice that this is not the inverse of the previous morph.

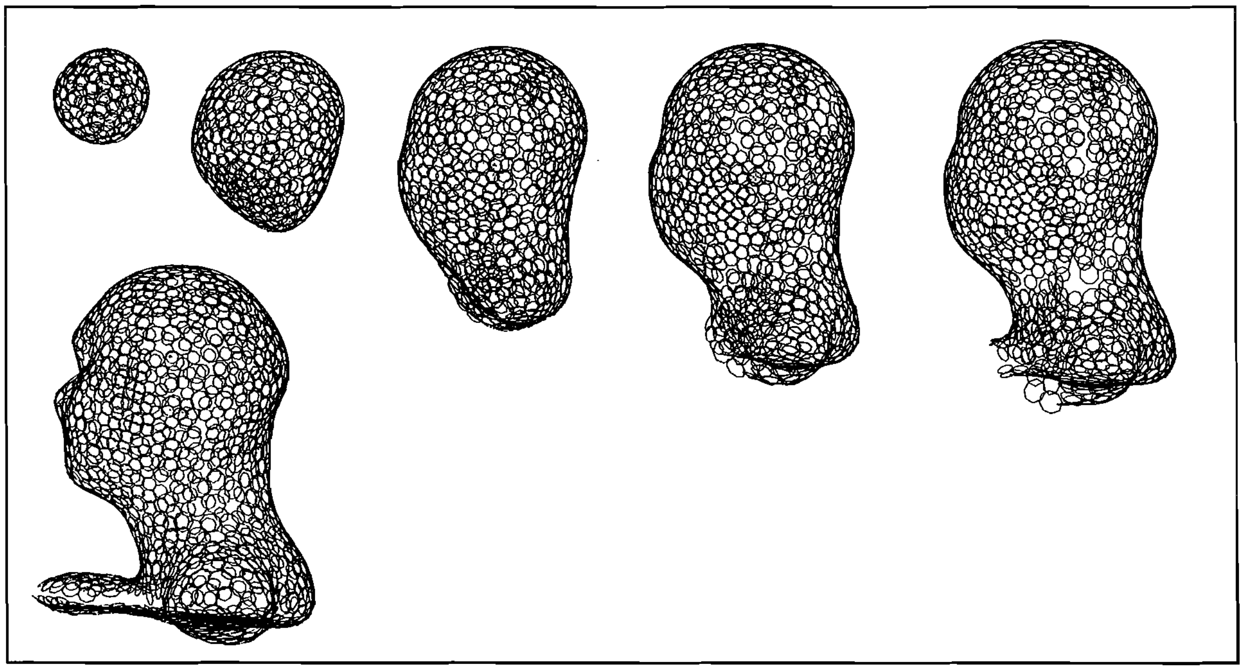

FIGURE 9. A morph from a sphere to a head which was obtained from a volumetric data sample. The morph is essentially a test of the growing heuristic since the head has considerably more particles than does the sphere.

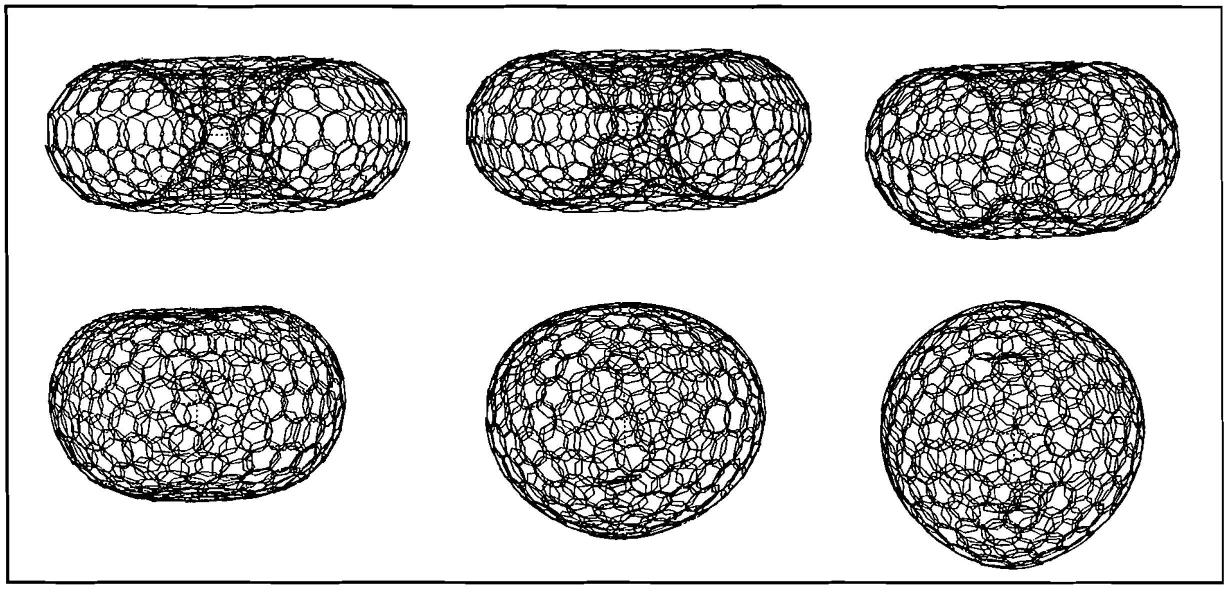

FIGURE 10. A morph from a torus to a sphere. Note that the genus of the initial object is 1 and the final object is of genus 0.

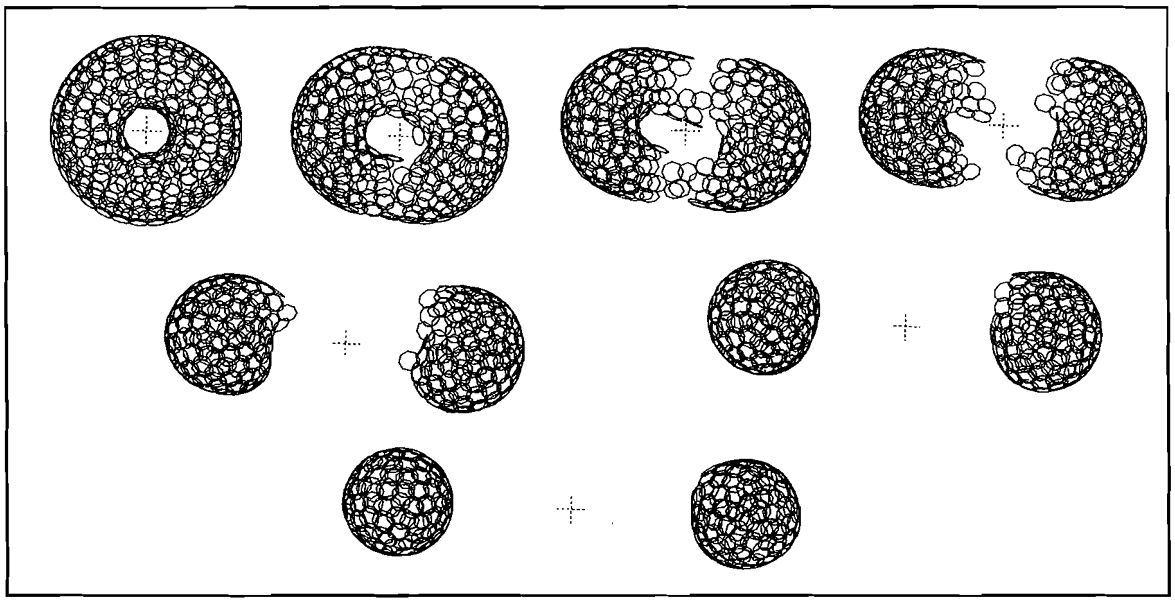

FIGURE 11. A morph from a torus to two spheres shows a genus 1 object being ripped apart to become two genus 0 objects.

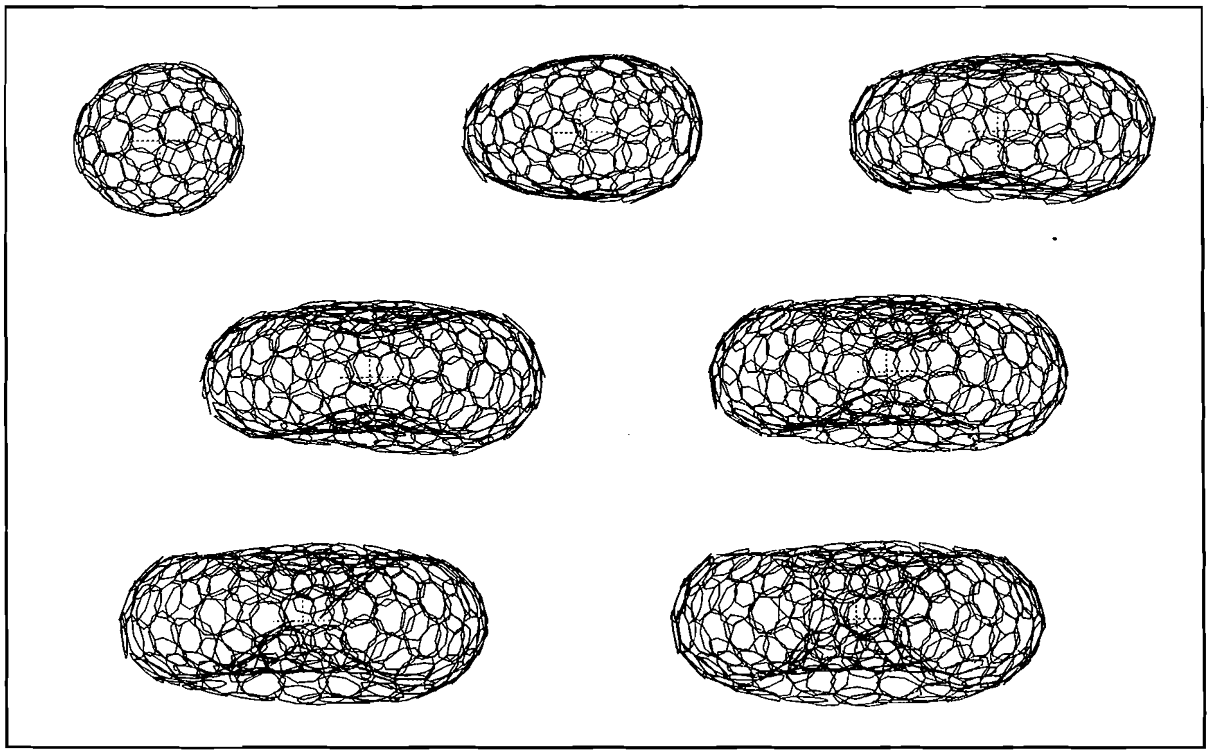

FIGURE 12. The morph from a small sphere to a bigger torus. Note that this morph requires particle generation, as each subsequent frame has more particles.

5.0 The Details of Our Particle System

This section describes the internal concepts and algorithms which are used in our oriented particle system. Our work borrows quite heavily from the oriented particle system presented in [Szel 92]. We have made an effort to point out those ideas which come directly from [Szel 92]. Additionally, we present several extensions which we have made to the system, these consisting mainly of the use and the generation of heterogeneous particles and of employing spacial subdivision for particle neighbor computations.

5.1 The Potential Functions

The Lennard-Jones potential governs the attraction and repulsion which particles experience towards and away from one another. When particles are relatively far away, they will be drawn closer. And when they are relatively close, they will be repelled. This interaction continues until particles settle into a low or zero energy configuration. The Lennard-Jones potential (figured below) is defined as follows

$$\phi_{LJ}(r) = \frac{B}{r^n} - \frac{A}{r^m} \qquad \text{(EQ 6)}$$

where r is the distance between two particles.

The derivative of energy with respect to r gives us the magnitude of the force mutually experienced by two particles at a distance r away from each other. The forces introduced into the system by the Lennard-Jones potential are given by

$$\phi'_{LJ}(r) = \frac{mA}{r^{m+1}} - \frac{nB}{r^{n+1}} \qquad \text{(EQ 7)}$$

As the figure below shows, when r approaches zero, the energy of the system goes to infinity. This is problematic in that when particles get too close, they exert a tremendous repulsion force on each other and are consequently catapulted away into space. This behavior, however, is readily correctable by limiting the magnitude of the force which any given particle may exert on another particle in a given system iteration.

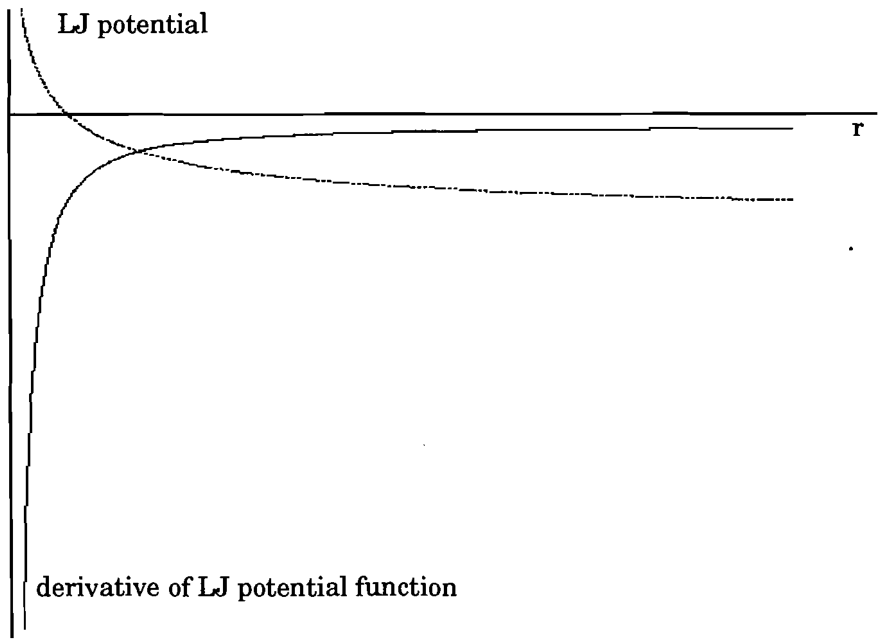

FIGURE 13. LJ potential function and derivative

As is pointed out in [Szel 92], particles which are subject only to the Lennard-Jones potential will tend to arrange themselves into solids, rather than surfaces. In order to induce surface type particle arrangements, [Szel 92] introduce the concept of the oriented particle, which in addition to its center, also stores information about its normal. In essence, then, each particle acts like a small piece of a 2D surface situated somewhere in 3 space. These oriented particles will tend to arrange themselves in surface type configurations.

[Szel 92] develop three potential functions which take into account not only a particle's position, but also its normal. The first of these, co-planarity, asks that adjacent particles lie in each other's tangent planes. That is the dot product of the vector between any two particles and their respective normals ought to be 0. Formally, we have

$$\phi_P(n_i, r_{ij}) = (n_i \cdot r_{ij})^2 \psi(\|r_{ij}\|) \quad \text{(EQ 8)}$$

The function $\psi(\|r_{ij}\|)$ is a monotone function which decreases with r and acts to limit the range of inter-particle interactions.

The second potential function, co-normality, serves to control surface twisting by requiring that the difference between adjacent particle normals goes to 0

$$\phi_N(n_i, n_j, r_{ij}) = \|n_i - n_j\|^2 \psi(\|r_{ij}\|) \quad \text{(EQ 9)}$$

Finally, to allow surfaces with a constant curvature, [Szel 92] introduce the co-circularity potential which asks that adjacent normals deviate from the vector orthogonal to *r*, the vector joining the particle centers, by exactly the same amount.

$$\phi_C(n_i, n_j, r_{ij}) = ((n_i + n_j) \cdot r_{ij})^2 \psi(\|r_{ij}\|) \quad \text{(EQ 10)}$$

The above potentials are not arbitrary; in fact, [Szel 91] presents a derivation for them from a finite element analysis of local deformation energies. This derivation is unnecessary for the purposes of our paper. The interested reader should consult [Szel 91] for more information.

The distance drop-off function $\psi(r)$ is defined as follows:

$$\psi(r) = e^{\frac{-r^2}{2\sigma_r^2}} \text{ where } \sigma_r = 1.0 \quad \text{(EQ 11)}$$

This limits the distance at which effective interaction between particles may take place. This is necessary to insure that only localized particle interactions occur. In other words, non-adjacent portions of the surface should not interact with one another. Rather interaction should take place only between physically proximate particles. This allows different surface positions to have vastly different surface configurations without worrying how these configurations might interact with others not physically close to them.

In order to allow for various surface properties (such as amount of curvature, or resistance to twisting, and so on), the energy contributed to the system by particle *a*'s interaction with particle *b* is defined as a weighted sum of all the potentials

$$E_{ab} = c_{LJ}\phi_{LJ} + c_P\phi_P + c_N\phi_N + c_C\phi_C \quad \text{(EQ 12)}$$

The total internal energy of the system is the summation of the energies contributed by every possible two particle pair

$$E_{int} = \sum_a \sum_b E_{ab} \tag{EQ 13}$$

5.2 Animating the System

By differentiating the various potentials with respect to particle centers and normals, we are able to obtain the forces contributed to the system by each potential function. The force experienced by a particle a consists of all the inter-particle forces acting on a from its neighbors summed with all external forces experienced by a and a velocity dependent dampening term

$$f_a = \sum_{b \in N_a} f_{ab} + f_{ext}(a) - \beta v_a \tag{EQ 14}$$

where β is some real constant.

The neighbors of a given particle a are defined to be those particles within a distance of some constant (e.g. 3.0) times r_{nat} away from the center of a, where r_{nat} is the natural interparticle spacing between any two particles computed by taking the derivative of the Lennard-Jones potential, setting it equal to 0, and solving for r.

The dampening term is necessary because without it the system does not seem to settle into a low energy state, but rather tends to oscillate uncontrollably. We hypothesize that this is a result of numerical errors experienced during integration. By varying the dampening coefficient we are able to effect behaviors which range from very fluid to very stiff.

The derivatives, with respect to particle positions and normals, of the three potentials are as follows

$$F_{LJ} = (-\hat{r}_{ij})\,\phi'_{LJ}(\|r_{ij}\|) \text{ where } \hat{r}_{ij} \text{ is the unit vector from } i \text{ to } j \tag{EQ 15}$$

We note that the Lennard-Jones potential does not consider particle normals and therefore only has a derivative with respect to position.

$$posF_P = 2n_i(n_i \cdot r_{ij})\,\psi(\|r_{ij}\|) + \hat{r}_{ij}(n_i \cdot r_{ij})^2\,\psi'(\|r_{ij}\|) \tag{EQ 16}$$

$$normF_P = 2r_{ij}(n_i \cdot r_{ij})\,\psi(\|r_{ij}\|) \tag{EQ 17}$$

Next we have the forces affecting particle positions and normals which arise from the co-normality potential

$$posF_N = \hat{r}_{ij}\|n_i - n_j\|^2\psi'(\|r_{ij}\|) \tag{EQ 18}$$

$$normF_N = 2(n_i - n_j)\,\psi(\|r_{ij}\|) \tag{EQ 19}$$

Finally the co-circularity forces are as follows

$$posF_C = 2(n_i + n_j)\,((n_i + n_j) \cdot r_{ij})\,\psi(\|r_{ij}\|) + \hat{r}_{ij}((n_i + n_j) \cdot r_{ij})^2\psi'(\|r_{ij}\|) \tag{EQ 20}$$

$$normF_C = 2r_{ij}((n_i + n_j) \cdot r_{ij})\,\psi(\|r_{ij}\|) \tag{EQ 21}$$

where

$$\psi'(\|r_{ij}\|) = -\frac{\|r_{ij}\|}{\sigma_r^2}\psi(\|r_{ij}\|) \tag{EQ 22}$$

From Newtonian mechanics, $f = ma$, rearranging for acceleration yields $a = \frac{f}{m}$. Assuming that all particles have a unit mass, we have that the acceleration experienced by any given particle is equal to the forces acting on that particle. Furthermore we have that

$$v' = a \tag{EQ 23}$$

$$x' = v \tag{EQ 24}$$

Therefore by integrating twice we are able to obtain a new position and a new normal for each particle once we have the forces which act on that particle in a given system iteration.

We should note that while we integrate directly to obtain a new normal, [Szel 92] integrate torques to obtain a rotation matrix in order to update a particle normal. We use the derivative of each potential with respect to the normal to give us acceleration. [Szel 92] use this derivative in a cross product with other vectors (see [Szel 92] for details) to obtain an axis of rotation and the magnitude of rotation for each normal. We did not see the necessity of this approach, and therefore chose our simpler method with satisfactory results. Furthermore, since it avoids a cross product computation and a matrix multiplication, our approach is computationally faster.

We use a simple Euler integration scheme for velocity v and position p as follows:

$$v_{new} = v_{old} + a\delta \text{ and } p_{new} = p_{old} + v\delta$$

where δ is the integration step size. By making δ smaller, we are able to achieve more stable integration results at the cost of slower performance.

Although this is not a high order integration scheme it has proved adequate for our purposes. Higher order schemes can be used to increase accuracy. See [Pres92] and [Burd93] for a review of higher order methods such as Runge-Kutta or semi-implicit integration.

Similar to [Szel91], when the system is being used to construct and shape surfaces, we found it necessary to animate the system for only a limited number of iterations, and then to disallow further particle interaction until the user explicitly manipulates some part of the surface. At that point, only those particles in the affected surface area are animated, while the rest of the surface remains static. This is necessary because if allowed to continuously animate the surface tends to lose all fine detail, and only retains more broad features. (Figure 14) shows an example of this phenomenon.

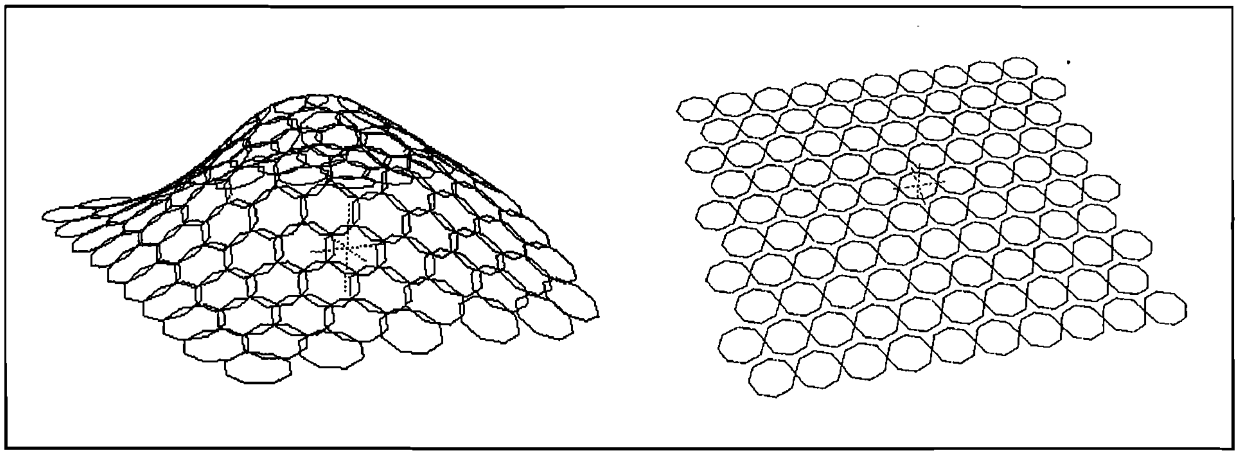

FIGURE 14. The plane with a bump would settle into the flat plane on the right if allowed to animate sufficiently.

This is an expected behavior because while any given local surface area may be in its lowest energy state, it is not in the lowest energy state in relation to its neighboring particles. Thus fine features tend to disappear as system animation progresses. Our approach allows enough settling to occur so that no surface areas look ragged, and yet no fine level of detail is lost.

5.3 Heterogeneous Particles

5.3.1 Problems with Homogenous Particles

A major difficulty in using an oriented particle system to model complex objects is that such objects require a large number of particles for accurate representation, and therefore require a considerable amount of computation to animate. In fact, with our original implementation we were unable to maintain any sort of real time interaction with the system once the number of particles exceeded a thousand or so. One of the reasons for having a large number of particles is that many objects have a very fine level of detail in certain places, and therefore require a small particle size to model these areas accurately. This, however, necessitates having a very large number of particles to cover many areas of these objects even if these have very little or no detail. We noticed that many objects which we dealt with had a fine level of detail only in certain portions of the surface, and that the rest of the object surface was not finely detailed at all. Such objects required the use of a small particle size in order to model the areas of fine detail, even though much of the object could have been modelled with relatively larger particles with no significant reduction to the accuracy of the model.

A model of a head, for example, has a considerable amount of detail around the nose and eye areas, and very little detail in many other places, see (Figure 15). Therefore, an accurate model of a head requires the use of small particles, even in areas which could be effectively modelled with larger particles. A way of reducing computation time for complex objects, therefore, is to have smaller particles in areas where there is a fine level of detail, and to have larger sized particles in areas with a lesser amount of detail.

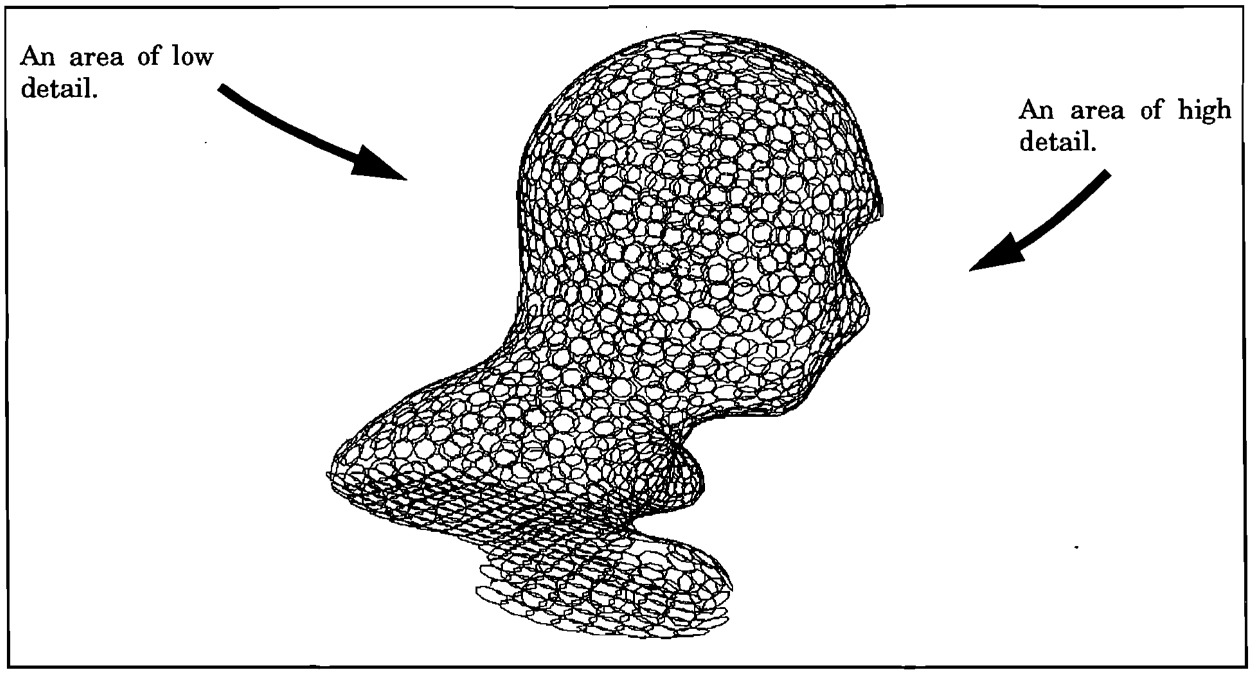

FIGURE 15. The back of the head has low levels of detail and may be modelled with larger particles without loss in the accuracy of the representation. The front of the head needs the smaller particles to represent the nose and eye areas.

5.3.2 A Heterogeneous Oriented Particle System

To overcome the above difficulty, we introduce the concept of heterogeneous particles. In addition to its position and normal, every particle also has a *size* associated with it. This concept is analogous to particle mass in the case of 3D particles; however, since our particles are really planar discs oriented in 3 space the idea of mass does not apply. We translate the concept of particle size into a particle radius in the following way. We start with the Lennard-Jones potential function equation

$$\phi_{LJ}(r) = \frac{B}{r^n} - \frac{A}{r^m} \qquad \text{(EQ 25)}$$

where r is the distance between the two particles in question. Differentiating this with respect to r gives

$$\phi_{LJ}'(r) = -\frac{nB}{r^{n+1}} + \frac{mA}{r^{m+1}} \qquad \text{(EQ 26)}$$

Setting this equal to 0 and solving for r yields

$$r = \left(\frac{nB}{mA}\right)^{\frac{1}{n-m}} \qquad \text{(EQ 27)}$$

We say that this distance, or $r(1, 1)$, is the distance which minimizes the potential energy between any two size 1 particles, or the natural interparticle spacing for two size 1 particles. For two particles a and b, we define $r(s_a, s_b)$ as follows

$$r(s_a, s_b) = r(1, 1) \times k \text{ where } k = \frac{s_a + s_b}{2} \qquad \text{(EQ 28)}$$

Taking the above definition of $r(s_a, s_b)$ we unwind (Eq 28) to get it into a form similar to the original Lennard-Jones derivative

$$r^{n-m} = \left(\frac{nB}{mA}\right)k^{n-m}$$

$$\frac{r^{n+1}}{r^{m+1}} = \frac{k^n nB}{k^m mA}$$

$$\frac{k^m mA}{r^{m+1}} = \frac{k^n nB}{r^{n+1}} \qquad \text{(EQ 29)}$$

The difference of the two terms in (Eq 29) gives us the derivative of the new Lennard-Jones type potential function.

$$\phi'_{LJ}(r) = \frac{k^m mA}{r^{m+1}} - \frac{k^n nB}{r^{n+1}} \qquad \text{(EQ 30)}$$

Integrating we have

$$\phi_{LJ}(r) = \int \phi'_{LJ}(r)\, dr = -\frac{k^m mA}{r^m m} + \frac{k^n nB}{r^n n} = \frac{k^n nB}{r^n n} - \frac{k^m mA}{r^m m} = \left(\frac{k}{r}\right)^n B - \left(\frac{k}{r}\right)^m A \qquad \text{(EQ 31)}$$

which is the modified Lennard-Jones potential function designed to take particle size into account.

We define the radius, *rad(a)*, of a particle of size s_a as

$$rad(a) = \frac{r(s_a, s_a)}{2} = \frac{r(1,1) \times s_a}{2} \qquad \text{(EQ 32)}$$

This definition for particle radius has the nice property that the sum of the radii of any two particles a and b is equal to the natural interparticle spacing between them. This is true since

$$r(s_a, s_b) = r(1,1)\left(\frac{s_a + s_b}{2}\right) = \frac{r(1,1) \times s_a}{2} + \frac{r(1,1) \times s_b}{2} = rad(a) + rad(b)$$

That this should be the case is very visually intuitive since any two particles should settle into a rest configuration where their borders are just touching. And this is exactly the behavior which our modified Lennard-Jones type potential function induces.

Recall the original definition of the distance drop-off function $\psi(r) = e^{\frac{-r^2}{2\sigma_r^2}}$ where $\sigma_r = 1.0$. Because we now have particles with radii significantly greater than the original *r* (which we define to be *r(1,1)*), this function will return values very close to zero (and thus nullify interaction due to orientation forces) for any two particles whose natural interparticle spacing sufficiently exceeds *r(1,1)*. We normalize this function by making it particle dependant, and thus having it return the same value for any two particles which have settled into their natural interparticle spacing. The new psi function, ψ_{new}, becomes

$$\psi_{new} = \psi(r_{norm}) \text{ where } r_{norm} = \frac{r}{k} \text{ and } k = \frac{s_a + s_b}{2}. \qquad \text{(EQ 33)}$$

This has the desired effect since $r(1,1) = \frac{r(1,1)k}{k} = \frac{r(s_a, s_b)}{k}$. Thus ψ_{new} will return the same value for any two particles located at their natural spacing away from each other.

Finally, we want larger particles in the system to act "heavier" than smaller sized particles. Intuitively, we can think of the larger particles as being conglomerations of smaller sized ones. Therefore, when subject to the same forces we want a larger particle to experience less acceleration than a smaller one. This is necessary to maintain system stability since we do not want large portions of surface to experience the same acceleration as smaller surface portions when subject to the same forces. This extension is very easy to implement since it follows directly from the classical Newtonian equation $f = ma$; thus we know that the acceleration experienced by any given particle is equal to the forces acting on that particle divided by its mass. The only problem here is that it is unclear what we should use as the mass of a given particle. We experimented with using the size and the radius of a given particle as its mass. However, we were able to achieve the best results in terms of system stability when we took mass to be equal to the particle disc's area.

Again, this result follows from the intuition that a large particle is merely a collection of smaller particles. The number of smaller particles which can cover a larger particle is equal to the area of the large particle divided by the area of the small particle. This suggests that area is the correct analog to mass for our purposes.

It should be noted that heterogeneous particles require no change in the co-planarity, co-normality, and co-circularity potentials, with the exception of the modified ψ function in (Eq 33).

5.3.3 Results and Benefits

We found that by using heterogeneous particles we were able to significantly reduce the number of particles necessary to model complex objects. The reader should consult Section 7.0 on page 55 for a discussion of the results we obtained.

5.4 KD Trees

A non-adaptive optimized KD tree was initially implemented to avoid the *O(n*n)* complexity of using a naive linked list. A non-adaptive optimized KD tree is basically a special binary tree that can be built if the data exists a priori. It is built from a linked list in the following manner. First one chooses a random item in the list, and then employs random partitioning and median-finding algorithms on unsorted lists to quickly partition the one unsorted list into two roughly (+/- 1) equal-sized unsorted lists of data with the following property. The elements in the left list are less than the median element in a given dimension (the x-coordinate for instance) and the elements in the right hand list are greater than or equal to the median element. Non-adaptive simply means that the data structure will partition on a different discriminators per level of this special binary tree. The discriminator is the x component of the data at level 0, the root. At level 1 the discriminator is the y component and at level 2 the discriminator is the z component. The discriminator is again the x component at level 3, and so on. The discriminator determines which component of x, y, or z we will partition the data into two sets of equal size. By being an optimized KD tree, the height of the binary tree is logarithmic in number of the particles. This is true because we split the size of the data (n) into two lists of size $n/2$. The building time of the KD tree is directly analogous to computing a Random QuickSort algorithm for existing 3D data. The search time is obviously $O(log\ n)$ since the tree is a full binary tree, with the exception that some leaves of the tree may be missing, depending upon the total count in the original linked list.

For more detailed information about various types of KD trees the reader can refer to [Same89].

5.4.1 Problems Encountered with Using KD Trees

A major difficulty in using KD trees as the underlying spacial data structure is the fact that KD-tree construction is not an incremental algorithm. In other words, it is impossible to update a KD tree for a change in the position of a single particle. When a single particle changes its position the entire KD tree needs to be rebuilt in order to be current. Because building KD trees is an expensive ($O(n\ lg\ n)$) operation, we had originally tried to build KD trees as infrequently as possible. We accomplished this by finding all the neighbors of a given particle within a range which was the maximum possible distance that a given particle could travel in n iterations of the system. This gave us neighbor lists which were unfortunately larger than they needed to be, and forced us into having to check whether any given neighbor was indeed in the interaction range for its corresponding particle. However, we were able to build KD trees as infrequently as every n iterations.

One exception to building every n iterations occurred whenever we create a new particle. Thus when we use the UI tools, described in the UI section, and create a new particle or move particles too fast then we were required to rebuild the entire KD tree. This is not a problem for the iterative spatial subdivision algorithm described in next section.

The introduction of heterogeneous particles into the system leads to another problem with using KD trees. In the case of homogenous particles, the Lennard-Jones potential gives us a unique inter-particle spacing at which the Lennard-Jones potential will contribute nothing to the total energy of the system at any given instant. We use this natural interparticle spacing, multiplied by some constant (about 2.0), to give us a range which is used to search for neighbors in the KD tree. Since the KD tree stores particles by their centers, we can find all particles whose centers are inside the given search range.

However, in the case of heterogeneous particles we are no longer able to do this because no single natural interparticle spacing exists. Every unique two particle size combination has its own unique inter-particle spacing. Therefore, if we try to find all the neighbors of a particle *a* of size 1, we invariably run into the problem of knowing how far away these particles can be. In the case that all of *a*'s neighbors are of a size close to 1, we can get away with searching at a distance of *r(1,1)* times some constant (say 3.0), and then checking any given neighbor *b* to see whether it is indeed $2.0 \times r(s_a, s_b)$ away from particle *a*. Unfortunately, this approach simply does not work for any of *a*'s neighbors which happen to be of size greater than 5, since $r(1,5) = 3.0 \times r(1,1)$. We simply do not detect that these larger particles are neighbors of *a* because their centers lie outside of the search range.

5.5 Spatial Subdivision

Because of the above reasons, we felt it necessary to implement a spatial data structure better suited to the demands of our system. For an overview of spatial data structures and collision detection algorithms, the reader can consult Chapter 3 in [Turk 90]. We implemented a spatial data structure very similar to uniform spatial subdivision as presented in [Turk 90].

Turk's algorithm essentially partitions the space which contains the particles into cubes of a given size. We store the location of the cubes by their upper left hand co-ordinates. Also, the cubes are positioned in integer multiples of cube size away from the origin in each of the three co-ordinate axis. In other words, we may have a cube at (0,0,0), at (*s*,0,0), at (*s*,*s*,0), and so on, where *s* is the size of the cube edges. Each particle *a* is treated as a sphere of radius $r(s_a, s_a)$. We then compute the cubes which intersect this particle. (Figure) provides an illustration of the 2D analog of this method.

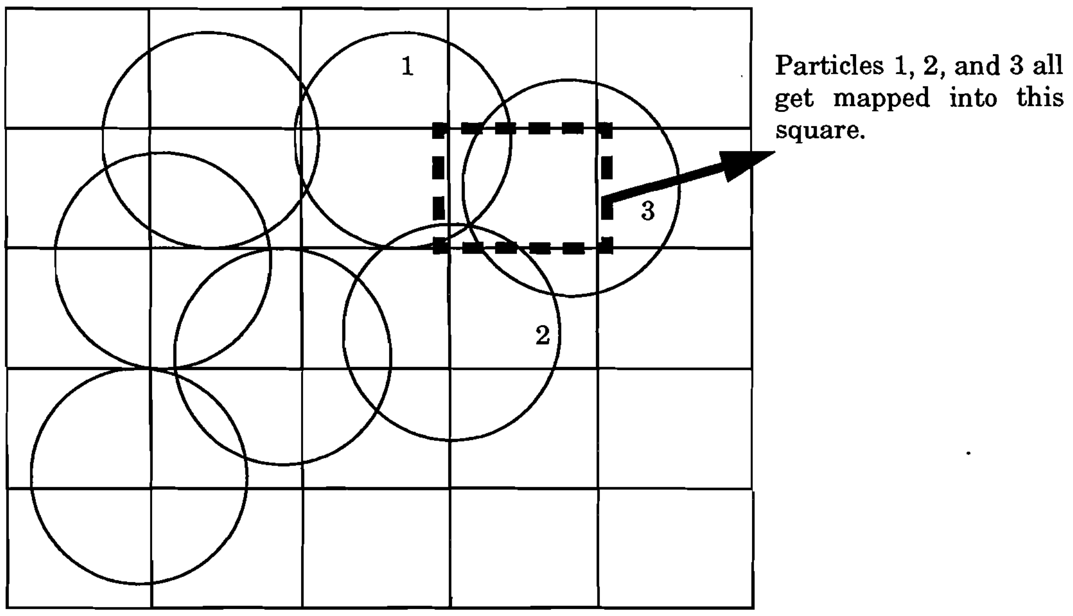

FIGURE 16. 2D analog for 3D spatial subdivision. Each circle represents a particle and it partitioned into its containing squares.

Every cube which contains part of the sphere represented by a given particle gets hashed into a table (currently with 4096 entries). Each entry in the space table contains the set of particles which have bounding cubes which map into that location. For any given particle a, we then compile a list of all particles which are present at the table locations that the bounding cubes of a get mapped into. Clearly, this list will contain all particles which share any bounding cubes with a. However, it may also contain particles bounded by cubes physically far away from a which get have the same hash value as some of the bounding cubes of a. Therefore, we perform a check to make sure that the particles returned by the algorithm are indeed neighbors of a. In practice, we found that the majority of the returned particles are indeed neighbors.

We use a modification of a hash function suggested in [Corm92]:

$$f(x, y, z) = (3x + 113y + 2311z)\left(\frac{\sqrt{5}-1}{2}\right)$$

We then extract the decimal part of $f(x, y, z)$ as follows

$$d(x, y, z) = f(x, y, z) - \lfloor f(x, y, z) \rfloor$$

Finally we obtain the hash value from

$$h(x, y, z) = N \times d(x, y, z) \text{ where } N \text{ is the table size} \quad \text{(EQ 34)}$$

Updating the spacial subdivision table for any given particle consists of first cleanly deleting that particle from the table by telling all its previous neighbors that it is no longer their neighbor and then removing the particle from all the table locations in which it was previously contained. Once the particle's new bounding cubes are recomputed, it is added to the particle lists at the table locations which those cubes are hashed to. The neighbor list for this particle is easily computed from the union of all particles already in those table locations. This particle is then added to the adjacency lists of all its new neighbors. Thus this process is completely incremental. This allows us to perform computations only for those particles which have moved sufficiently away from their previous locations to merit a table update. Since interaction in a given particle system tends to be localized to small areas, this provides a significant speed up in neighbor computations and spatial structure maintenance. Furthermore, this approach is asymptotically superior to KD trees since building a table is linear in the number of particles (provided that the worst case number of bounding cubes does not exceed some reasonable threshold). And search times are likewise constant. This compares well to the *O(n lg n)* build times and *O(lg n)* search times of KD trees.

Finally this approach solves the problem of finding large neighbors of small particles since any two physically proximate particles will be reported as being neighbors by this algorithm. Also, in the case that a large particle is surrounded by many smaller ones, this approach will report significantly fewer neighbors than the KD tree approach.

One drawback of this approach, stemming from the fact that particles may be of varying sizes, is that we may find that a given particle has an overly large (in the hundreds) number of bounding cubes when the cube size is substantially smaller than the biggest particle radius. Alternatively, we may find that an overly large number of particles are contained within a single cube when the cube size is substantially bigger than the smallest particle radius. The second case leads to neighbor lists which are larger than we would like. Fortunately, most of our particle sizes (and thus radii) only vary by a factor of at most 10 or so. Furthermore, the generation of our heterogeneous particle system ensures that the smallest particle is of size 1. This avoids the problem of having many particles hashing to one cube. Thus the decoupling of the sampling and filtering size for generating a heterogeneous particle system is independent from the spatial subdivision cube size and allows us to keep the numerical integration step size-independent as well. For more details please refer to the section on "Volumetric Data and Variable Sized Sampling" on page 55.

A possible speed up to the current approach would be to treat each particle as a disc in 3 space, as opposed to treating it as a sphere. This idea fits in quite well with the concept that each particle is really only a piece of a surface (a planar disc) in 3 space. Since we would only look for the cubes which contain a disc, and not a sphere, the method would clearly reduce the number of bounding cubes for the large particles, and would thus allow us to have cube sizes only slightly larger than the radii of the smaller particles.

5.6 Automatic Particle Generation

In modelling surfaces it sometimes is quite useful to have gaps which arise in the surface be automatically filled in. For example, when stretching the surface we could have particles automatically generated once gaps arise. This is analogous to the behavior exhibited by the grow sphere. However, that technique is limited to the sphere tool. This section discusses two other, more general, particle creation techniques. Both are variations on methods of the same name as presented in [Szel 92].

5.6.1 Growing

The growing heuristic works as follows. The user specifies a given size for the particles which are to be grown. An area of the surface is then highlighted to indicate where growth should occur. For a given highlighted particle a, we divide the circumference of that particle by the diameter of the growth particles. This tells us approximately the number of particles, N, that ought to be placed around a. Then we attempt to place growth particles equidistantly spaced about the boundary of a at a distance of $rad(a) + rad(g)$ where g is the growth particle away from the center of a. This is accomplished by placing a particle every $\frac{360}{N}$ degrees away from an arbitrary starting point around a. For each potential growth particle, we check to see if that particle falls within any other already existing particle. If so, then the particle is not created. If not, then we create the new particle.

Clearly this approach works best when the size of the growth particles is either the same or smaller than the size of particle a. As a trivial extension to our algorithm, we could enlarge a highlighted area by making the size of the growth particles for any given highlighted particle equal to the size of that particle. Alternately, the growth particle size can be uniform, and we will only grow around highlighted particles which are larger than the growth particles.

5.6.2 Stretching

This section discusses the stretching heuristic. Stretching can be used to fill in gaps created during user manipulation. Instead of tearing, particles will be created in gaps that occur while a user tugs and pulls at various surface areas using the various user interface tools (see section on UI tools). The heuristics that worked well for us are described next.

First, we only look at particles which are currently active in the system[1]. Next, the stretching rule checks to see if two neighbors have a large enough opening between them to add a particle. If the distance between two particles is such that $dist_{lower} < opening < dist_{upper}$, then we will attempt to create a particle. Having $dist_{lower} = (1.8 \times r(1, 1))^2$ and $dist_{upper} = (2.5 \times r(1, 1))^2$ is not a sufficient condition for particle generation. It is also necessary to limit the affine angle between normals of two particles that satisfy the distance requirement. This ensures that the particles created do not add to the noise level present in a particle system. Generating new particles in this manner creates a surface without holes in the region where stretching is taking place, whereas adding particles

1. Recall that in order to prevent a complete surface settling, only a subset of the particles is active at any given time.

without checking this angle may result in exponential particle growth in noisy areas. The "critical" angle that worked quite well for was 82 degrees.

We next need to check and see if we can safely add a particle. If a there exists at least one particle within a given distance[1] from the average of the two particles then no particle is created and we continue to check the list of all particles. Currently we accomplish this by performing a range search into our spatial data structure looking $0.8 \times r(1, 1)$ in 6 directions centered about the particle. If there are no particles already in this area in the spatial data structure, then we have a candidate particle to create. There is only one more step which involves range searching $0.8 \times r(1, 1)$ into the newly created particle list. If no particle exists in this list, then we can finally and safely create and add this new particle to the new particle list. The normal of this newly created particle is determined by taking the average of the two normals of the particles.

The overall computation appears to be n^2, for we need to find all pairs of neighbors. But actually it is merely only *O(n)*, where *n* is the number of particles. The reason is that we keep an adjacency list for each particle which contains the neighbors for this particle. For a uniform size = 1 particle system, we simply look $3.0 \times r(1, 1)$ to find all neighbors for a given particle, where defined previously $r(1, 1)$ is the interparticle spacing for two particles of size 1[2]. For those interested in result of stretching pictures please refer to [Szel92].

For temporally growing particles during a morph please refer to the micro morphing section which gives the details of how and why we felt it was important to automatically create particles when two systems have different number of particles.

6.0 User Interface and Interaction

Our system is designed to provide a real time interactive environment in which the user can create

1. It's not really a spherical distance, instead we only check to see if a particle exists in a cube centered about the particle point in space.
2. Incidentally, $r(1, 1) = 1.73205$.

and configure various surfaces by using an oriented particle system. We have extended the techniques presented in [Szel 92] for interacting with the particle system; furthermore, we have developed several new interactive techniques which will be described in this section. We have improved the triangulation algorithm presented in [Szel 92]. Thus the user is able to create fairly robust polygonal models of surfaces and objects created with the particle system. Our system allows the user to create and then manipulate an oriented particle system by sampling volumetric data at various levels of detail. Our volumetric sampling algorithm creates a heterogeneous particle system approximation to the given volumetric data in order to decrease the number of particles needed for such a representation. Finally, we have developed a 3D morphing algorithm which, with minimal user input, can create an animation sequence showing one particle system being transformed into another. Thus the user is able to quite simply create many complex animations with the aid of our system.

6.1 User Interface and the Scripting Language

The user interface allows direct interaction with the oriented particle system. The user may manually select and position particles by selecting them with the mouse. We use virtual trackball and camera abstractions to allow the user to view the world from arbitrary positions and perspectives. Furthermore, our scripting language provides a direct interface to many system functions without the need for interaction with the user interface. The entire system has been developed and implemented in the framework of Brown's UGA environment [Zele91].

6.1.1 A Description of UI Functionality

The user interface consists of two windows, the main menu which is shown in (Figure 17), and a window which is used to display the particle system and in which the user may interact with the particles. This section describes the functionality of the user interface.

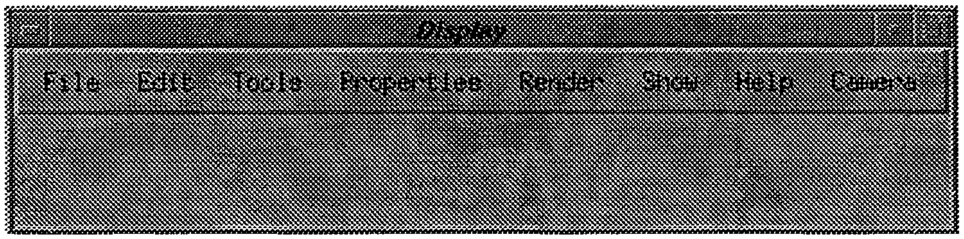

FIGURE 17. The user interface menu widget.

The file menu allows the user to

open an existing particle system

save the current particle system

sample a volumetric data set

quit the program.

The edit menu contains the following options:

Default Surface - creates a default surface which is a flat sheet based 10X10 grid of particles.

Heat Selected - activates particle interaction for the currently selected particles. This may in turn cause other particles to be activated depending on their proximity to the selected particles.

Delete Selected - deletes all currently selected particles.

Cool Selected - turns off particle interactions for currently selected particles.

Interpolation - pops up interpolation dialog box. This box contains 2 toggle buttons for turning on and off interpolation heuristics for auto creation of new particles in the system. The heuristics are *Stretch* used when pulling surfaces so the surface stretches as you pull it, and *Grow* which is used for growing particles in sparsely populated areas of the system (i.e., a sparsely populated volumetric data sample.).

Unorient Selected - turns off orientation potential functions for selected particles, these particles still care about their distance from other particles but no longer care about their orientations to neighboring particles (i.e., used for putting a crease in a surface.)

Select All - selects all particles in the system, highlighting them in the drawing area.

Unselect All - unselects all particles in the system returning the to there original color.

The tools menu contains the following options:

No Tool - removes any tool currently being used and returns to mouse interaction with the particle system.

Push Sphere - places a sphere in the drawing window, which when dragged into particles applies forces to the particles in the direction of the sphere's movement.

Select Sphere - places a sphere in the drawing window, which selects all particles it comes in contact with.

The properties menu contains the following options:

Particle - pops up a dialog box that allows control over the particle potential functions (*co-planarity, co-circularity, co normality, and LJ-distance*) and allows control over the force dampening coefficient and the integration step size for the differential equations.

Surface- pops up a dialog box which contains current surface properties. This box currently only contains a count of the number of particles in the system.

The rendering menu, assuming the surface is currently triangulated, allows the given particle system to be displayed in shaded or wireframe form.

The show menu allows different viewing options for the system, these being:

particles - displays particles only as surfels (small octagons).

surface - triangulates surface, and displays as a triangle mesh.

all - displays triangle mesh and overlays particles.

The help menu option is self-explanatory. The camera menu either resets the camera to its original position and orientation, or brings up a dialog box via which the user may move and rotate the camera.

6.1.2 The Mouse

This section describes the mouse functionality.

Mouse Button 1 - on particles selects the particles, queues the user by highlighting the selected particle.

Mouse Button 1 Drag - when no tool is present applies an external force to the selected particle in the direction of the mouse movement. When push sphere is active moves the sphere but does not affect particles.

Shift+Mouse Button 1 - selects multiple particles.

Shift+Mouse Button1 Drag - applies external forces to the selected particles in the direction the mouse is moving. When the push sphere is active, this applies forces to the particles in the direction of the sphere's movement. When the select sphere is active this selects particles which are inside the sphere.

Ctrl+Mouse Button 1 Drag - when push sphere is active, activates automatic particle creation when applying external forces to the selected particles in the direction the mouse is moving to fill in gaps.

Mouse Button 2 Drag - selects all particles the mouse runs over.

Shift+Mouse Button 2 Drag - unselects all particles the mouse runs over.

Mouse Button 3 Drag - virtual sphere rotation.

6.1.3 Scripting Language

In order to allow for a greater user control over the particle system, the control scripting language may be used in conjunction with the user interface. Furthermore, the addition of this scripting language allows us to write ASCII scripts that we source to set up object creation prior to attempting morphs. Our initial reason for having a scripting language was that we felt it might be necessary if we wanted to compute multiple morphs while not being physically present. For a complete overview of the scripting language the reader should consult Appendix A. We should note here that the UI and the scripting language grew up at different times, and therefore do not provide access to the same functionality. Since this is not a commercial quality product, we do not see this inconsistency as problematic. We implemented the parser for the language using Lex and YACC, and were able to read characters, in addition to catching other system events, by using UGA's built in event-handling facilities. The most important aspect of our scripting language was the ability to script other portions of the language from ASCII readable files. See Appendix A for a complete listing of the scripting language.

6.1.4 A Hypothetical User Session

The user starts the system and brings up the user interface. Via the scripting language, a given volumetric data file can be sampled to create a heterogenous system at some user specified frequency (see section on volumetric sampling). Once this is done, the user has a heterogeneous oriented particle system representation of the volumetric data set. The user is then able to shape and manipulate the surface by either direct mouse interaction, or by using the push sphere tool (see section on the tools). Once the user has satisfactorily modified the particle system, the surface may be triangulated in order to obtain a polygonal model. The latter may then be exported into some other polygonal modelling system. Furthermore, the user can load another volumetric data set into the system, and then specify a morph between the two sampled objects. The intermediate frames of the morph can then be viewed in order to create a visual transformation between the two objects.

6.2 Modelling Tools

One tool available to the user is a sphere which may be used to highlight large numbers of particles easily. The user selects the sphere via the menu and is then able to move the sphere around the particle world with the effect of highlighting all those particles which fall inside the volume of the sphere.

This is useful in order to quickly select a large number of particles, a task which would be quite cumbersome if each particle had to be individually selected with the mouse.

The push sphere (which is similar to the tool introduced in [Szel 92]) allows the user to push a solid sphere through the surface of an object with the effect of placing all particles which fall within the upper hemisphere of the sphere (along the direction in which the sphere is moving) onto the sphere surface. Furthermore, each affected particle's normal is set equal to the surface normal of the sphere at the center of that particle.

More interesting effects can be achieved with the grow sphere. This tool has similar behavior to the push sphere. However, in pushing particles onto its periphery, it checks to see whether any gaps develop on its surface. When this occurs, the sphere places a user-specified size particle into that gap. Again, the normal of the created particle will be equal to that of the sphere surface normal at the center of the particle. This tool allows the user to quite simply create interesting new surface features. (Figure 18) shows an example of a bump created using the grow sphere tool.

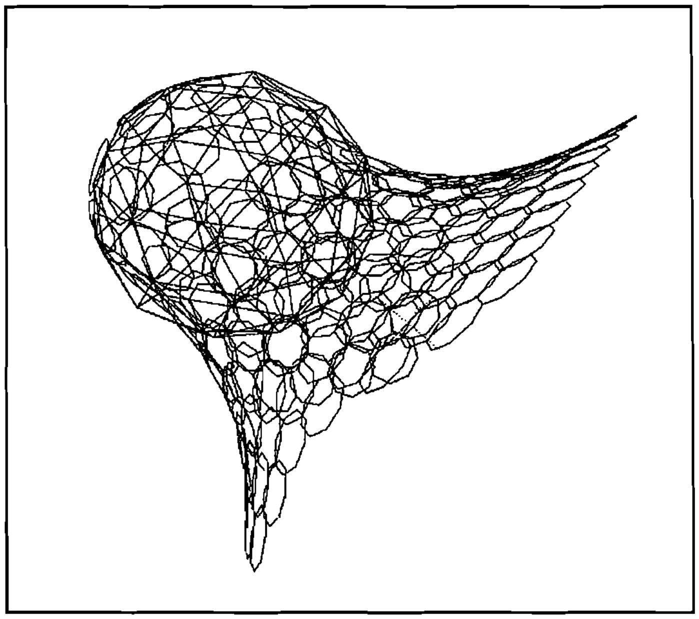

FIGURE 18. Creating a bump with the push sphere tool. New particles are generated along the sphere surface and are placed into the particle sheet as space allows.

As we noted before, when the user interacts with any particles those particles and their neighbors within some specified range are allowed to interact, while the remainder of the surface remains static. Again this is necessary so as not to lose fine surface detail.

A tool presented in [Szel 92], which we did not implement, but which may be useful for certain user shaping tasks is a knife tool which cuts the surface by physically separating adjacent particles in order to create a surface rip. Another interesting tool which is quite easy to implement is the clamp tool, which grabs any particles inside its volume and as it is moved, moves the clamped particles along with it.

The oriented particle system paradigm provides a very powerful and flexible method for creating complex 3D objects; therefore, an interesting and provocative future area of research would be to develop a more complete tool set for user interaction with the particle system.

7.0 Volumetric Data and Variable Sized Sampling

This section describes the "adaptive volumetric sampling and generation" of a homogeneous or heterogeneous particle system from volumetric data. As mentioned previously, we want the ability to accurately represent the fine levels of detail present in a given set of volumetric data. However we also noticed that not all objects are made up of a fine levels of detail, so we also wanted the ability to lower the overall particle count in areas that were basically planar. We had access to a Brown UGA library package CUBIE that implements the Marching Cubes algorithm [Lore87] on volumetric data sets and generates polygonal output. So we use the CUBIE marching cubes algorithm to read in volumetric data and then we filter the generated polygonal Marching Cubes output in the following manner.

1) First we attempt to generate a homogeneous oriented particle system from a given volumetric data set of 27,000 voxels with a user-specified particle size which allows us to compute the spacing between particles as a filter to avoid placing particles too close.

We do this by thinking of the *n*-sided polygon as being composed of "*n-2*" triangles. Thus computing the average of the vertices of the reported triangles for determining what we call the centroid of the reported n-sided polygon.

As previously mentioned, our natural interparticle spacing for size "1.0" particles is computed to be 1.73205. So a user specifying a particle size of .5 corresponds to a verifying that no particles should already exist within in a spacing of .866025. So we look at a point in space and basically look to make sure no particles exist within a "cube" centered at this point in space being considered. We decided to use a "cube" as it is a quick approximation to a sphere as this gets called very frequently.

If the spacing is large enough that we could fit in a particle, next we compute the average of the normals of the n-2 triangles in the n-sided polygonal. Now that we basically have an approximation an to the original n-sided polygonal we use the computed position and the computed normal to safely add a particle to the system.

3) Once we have a finely sampled homogeneous oriented particle system, we avoid numerical instability with the small particles by employing a transformation. We scale all particles such that the smallest particle has size "1.0" in our system. Thus the discrete time step for the physical integration can remain the same and remain independent of the particle sample size. Also it allows our spatial subdivision algorithm to avoid the problem case previously mentioned that occurs when too many particles hash to the same cubes.

This entire process described above happens much faster if we first turn off the dynamics (physical integration) of the system to keep the system interactive during the generation of surfels.

Next, we allow the particle system forces to settle the surface the number of iterations specified by the user. The amount of smoothing passes over the data is user-specified and we found that a number between 5 and 10 works quite well for the head volumetric data in (Figure 15).

If the user has only asked for a homogeneous particle system, then we are done with the volumetric generation algorithm. However the user could then ask to grow particles in the gaps using the particle growing described in "Growing" on page 47. The user could also generate a conventional polygonal model at this point, as we can Delaunay triangulate the oriented particle system to generate a polygonal surface which can be shaded and illuminated.

4) Else if the user has asked for the generation of a heterogeneous oriented particle system, then we continue the algorithm described in the next section.

7.1 Heterogenous Particle Generation

A user-specified threshold for what it means to be planar is employed in the next portion of the algorithm. This threshold is merely the allowable standard deviation, representing curvature at a given particle point of view, allowed between the normals of a collection of neighboring particles some distance from a given particle. We think of this threshold as a loose definition of the "what is means to be planar" at a given area of the oriented particle system.

The low level generation of a heterogeneous oriented particle system is as follows.

First all particles are labeled "UNMARKED" in the current particle system list. Then we walk the current list of particles considering only particles which are still labeled "UNMARKED" by the time we consider them in the algorithm.

For each particle we compute the standard deviation of the neighboring particles within a radius of "1.5 * interparticle spacing."

If the computed standard deviation of the particle normals is lower than the user-specified threshold, then we recompute the standard deviation of larger distance (The original distance weighted by 1.25). We remember both this and the previous collection of particles and continue looking out farther from this point in space until we have surpassed the user threshold. At this point use the previous collection of particles satisfying the user specified threshold is retained and we mark all of these particles as "SAME" and generate one new conglomerate particle in a future particle list. This new larger-sized aggregate particle is given a radius equal to the last computed radius that satisfied the standard deviation condition and is given a normal equal to the average of the particles in the "SAME" list of the given particle. The particles marked "SAME" are the particles which will no longer exist in the future particle list.

We continue the process looking for particles in the current system until all particles are no longer "UNMARKED." Any particles including a particle's neighbors who fail the first test because the computed standard deviation is larger than the threshold are added to the future particle list. Also all particles that have only themselves as neighbors are added to the future list.

We attempt multiple passes until the resulting heterogeneous particle system is well represented by heterogenous particles. We perform the same process described above. (Figure 19) displays the difference in the resulting picture.

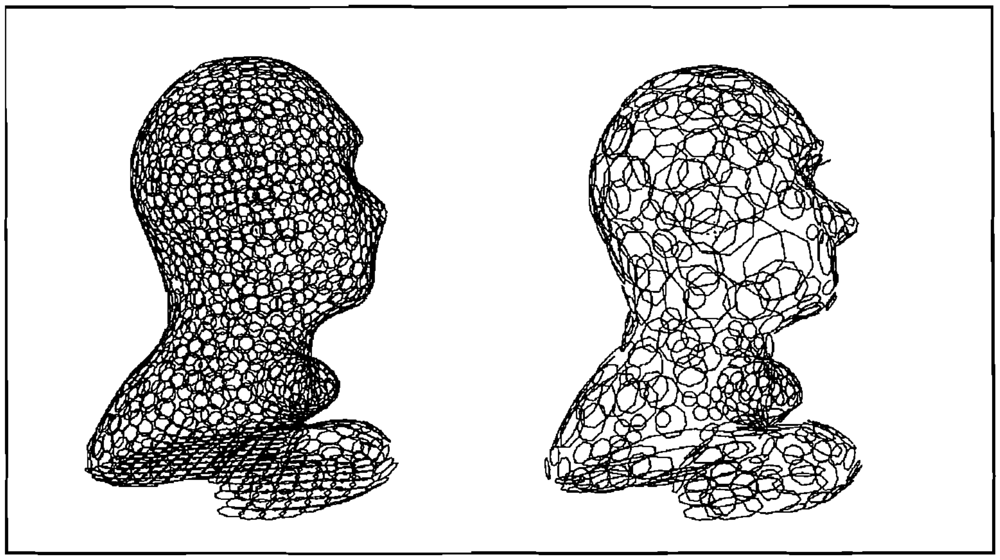

FIGURE 19. Volumetric head sampled with particle s=.5 and a standard deviation threshold = .25. The resulting particle counts are 1191 for uniform particle size head and 340 for the heterogeneous head.

7.2 Volumetric Results

This section shows the reduction of the number of particles by adjusting the particle size and the threshold on various data sets. A volumetric bust of a head and a volumetric teapot were used as data sets.

TABLE 2. Volumetric Head

Particle size	Standard Deviation	Particle Count (heterogeneous / homogeneous)
.25	.1	(2201 / 2380)
.5	.15	(563 / 1191)
.5	.25	(314 / 1191)
.25	.25	(1292 / 2380)

TABLE 3. Utah Teapot

Particle size	Standard Deviation	Particle Count (homogeneous/heterogeneous)
.25	.15	(1838 / 2584)
.5	.15	(557 / 1269)

8.0 Rendering

We render our oriented particle system by drawing an oriented octagonal planar disc (with the same surface normal as the particle) to represent each particle. The size of the octagonal disc is determined by the radius of the given particle. We can also triangulate the surface and thus create a conventional polygonal model. The details of how we do this are discussed in the section below. The polygonal model representation can be shaded and illuminated in a traditional fashion on any workstation. Within the UGA [Zele91] environment at Brown, this is done automatically once we specify a triangulation of the surface. Other rendering techniques, such as ray tracing, could be employed to render oriented particle systems. For examples reader may consult [Blin 82a, Reev85, Sims90, Blin 82b,Wyvi86, Hers87, Mill89, Tonn89].

8.1 Triangulation of the Oriented Particle System

A triangulation of a collection of vertices consists of a set of triangles which fully covers the surface defined by the vertex points. Triangles may be created from any three points from the vertex collection. We extend the triangulation algorithm presented in [Szel91], which is based on the Delaunay triangulation [Bois84]. In a Delaunay triangulation, a triangle is acceptable if it is the case that no other vertices lie inside the circle circumscribing that triangle. [Szel91] extend this concept to three dimensions by checking the circumscribing sphere which contains the triangle to make sure that no other vertices lie inside of it. If a triangle satisfies this criterion, we then proceed to check whether it is a *degenerate triangle* or not.

Our test for degenerate triangles consists of making sure that no side of a triangle exceeds a certain length (we use $3.0 \times rad$ where *rad* is the radius of the particles which serve as the triangle vertices). We should note that our approach works only for our homogeneous oriented particle system. Therefore the value of rad will be constant for any pass of the triangulation algorithm. We have not developed a triangulation algorithm for heterogeneous oriented particle system. Furthermore we check to see that all three triangle angles fall between 25 and 130 degrees. If both of these criteria are met, the triangle in question is not a degenerate triangle and may be added to the triangulation.

Since we have surface normals at each of the three triangle vertices (each one is an oriented particle), and since these must (because of the nature of the oriented particle system) all lie in the same half-space, we know which is the up side of the triangle and which is the down side. Therefore we are able to store the edges of the triangle in sorted counter-clockwise order around the triangle surface normal. Furthermore we are able to give each edge an orientation along the direction of the rotation. This fact allows us to perform a second pass in which we add triangles which the first pass as described above failed to generate.

We look for every three edge cycle which is composed only of edges each of which belongs to one triangle. The existence of such a cycle implies a triangular region which was missed in the original triangulation process. We then proceed to create a new triangle composed of the three edges in question to fill this region. Now each of these edges becomes a part of two triangles. Thus our second pass algorithm will not identify this opening any more. (Figure 20) illustrates this procedure. We found that this method found missing triangles in almost all cases where such triangles existed.

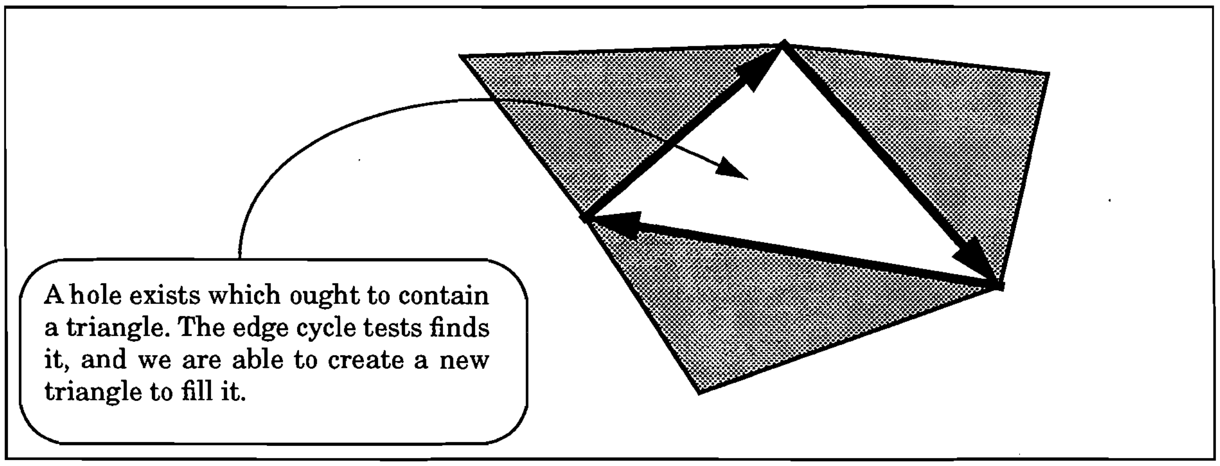
A hole exists which ought to contain a triangle. The edge cycle tests finds it, and we are able to create a new triangle to fill it.

FIGURE 20. A case where three edges, each of which belongs to only one triangle, form a cycle. We are able to find cycles because each triangle has a surface normal associated with it. Thus we know what counter-clockwise means for that triangle. Once the hole has been found, we create a new triangle to fill it. Assume all three shaded triangles have surface normals which point in the same direction outwards from the page.

9.0 Conclusion

In this paper, we have developed a three dimensional surface morphing algorithm which uses the oriented particle system as its underlying object representation. We divided our morphing process into two steps: the macro and the micro morphs.

The macro morph consists of a rigid body transformation alignment of the initial object to the destination object in order to achieve a user specified best fit. The transformation is computed by aligning the three major axis of the initial object with the three major axis of the final object.

The micro morph consists of transforming the initial system into a very close approximation of the destination surface, which we refer to as the destination surface. The process consists, roughly, of the establishment of a correspondence between the particles of the two systems, and then an interpolation where the initial system particles are moved over into their destination system counterparts. During this process, particles may be added or deleted as necessary. The addition of particles effects the creation of new correspondence pairs, causing the initial correspondence to evolve through out the course of the morph; therefore initial particle correspondences do not have to be exact. This allows us to morph between objects of very disparate sizes.

Additionally, by using the oriented particle system, an inherently topology -free object representation, we were able to avoid the pitfalls involved in morphing between objects of different genera and different topologies. For example we can successfully morph a sphere into a torus or a torus into two spheres, something which most previous approaches to 3D morphing were unable to achieve. We believe our morphing approach to be both intuitive and effective. It allows the user, with only minimal input, to specify very complex shape transformations.

Furthermore we have extended the oriented particle system in [Szel92] by introducing the concept of heterogeneous particles. This allows for surface regions with high levels of detail to be represented by smaller sized particles, while the rest of the surface may be modelled with larger particles. In this way we are able to represent surface regions previously composed of many smaller sized particles with only several larger sized ones. Thus the resultant system contains far fewer particles than the original one. Clearly this speeds up system animation times.

We also implemented a spacial subdivision algorithm, based on the one in [Turk90], in order to be able to perform rapid neighbor computations. The new algorithm is superior to KD trees in that search times are constant, and not logarithmic. Furthermore, an entire space hash table may be constructed in linear time in the number of particles, where as the construction of a KD tree requires $O(n \lg n)$ time where n is the number of particles in the system. One of the biggest benefits of the spatial subdivision approach, however, lies in the fact that it is entirely iterative. Thus the change in state of a single particle requires updates to be performed only for those table entries associated with the particle in question. A KD tree, on the other hand, would have to be entirely rebuilt in order for it to be accurate. Thus maintenance times for spacial subdivision are constant, where as for KD trees an entire tree construction is required, making the process $O(n \lg n)$. Another major benefit of the spatial subdivision approach lies in that it allows us to find large neighbors of smaller particles, a task which KD trees can not do as efficiently.

10.0 Future Directions

One of the problems with our current implementation of the particle system is that it is still too slow. We lose real-time user interaction at about one hundred particles. Fortunately many of the computations performed in animating the system are independent of one another, and could be performed in parallel. For example, each particle may individually compute the forces which act upon it in any given iteration. Thus parallelizing the particle system, especially since it lends itself so nicely to a partitioning computation, should yield a significant performance increase. Furthermore, new user interaction techniques could be developed to simplify the creation of surfaces with oriented particles. A tight parallel implementation, coupled with a good set of user tools, could result in an oriented particle system which is a very effective and highly flexible surface modelling method.

New rendering methods for oriented particle systems might provide better visual feedback. For example it would be nice to have the ability to ray trace an oriented particle system (something which should not be very hard to do, requiring only the ability to intersect a ray with the particle system). In this way, we could lay texture maps over the oriented particle system to produce visually realistic objects. Furthermore, a method for the triangulation of heterogeneous particle systems clearly needs to be developed. Again, we do not anticipate that this method should differ very greatly from the one we currently use. Furthermore a simple z-buffer could be used to perform hidden surface elimination so that particles obstructed by other particles would not be visible to the viewer. One of the problems of viewing a sphere composed of particles, for example, is that it is very difficult to determine which particles are in the foreground, and which ones are in the background. Also, particles could be treated as solid colored objects, where the colors would be representative of certain particle properties. For example, surface areas with high standard deviations of particle normals could have a different coloring than flatter surface areas.

A rather simple, but probably very beneficial addition to our morphing approach, would be to incorporate a texture map interpolation from the initial to the final system. That is both systems begin with a texture map which is laid out over the surface designated by the particle system. As the morphing process proceeds, this texture map would be interpolated and mapped onto the intermediate objects in the shape transformation. This extension would allow for very visually pleasing morphs to take place

If speed during the 3D morph is required, then one might consider a speedier algorithm. To quickly display heterogeneous particles one might triangulate just the octagonal 2D discs into 6 triangles. Then by rendering particles in this fashion with different colors pertaining to 1 or 2 texture maps or a per surface or per triangle or per vertex color from the color table, context could be given to the metamorphosis process. A z-buffer could aid in the visualization process of this quilt-like work surface.

Since we are limited to modelling surfaces which are locally either planar or circular, we are unable to model things like sharp edges. Of course many objects which we would like to morph may, in fact, contain sharp edges. New potential functions could be developed to extend the modelling capacity of oriented particle systems. Doing this would allow a user to model a wider array of shapes. Thus morphs could take place between objects with sharp corners, such as cubes and so on.

We did not employ heterogeneous particles in our morphing algorithm. We believe that this is an area which needs to be further explored. In other words, outside of the savings obtained in object representation, are heterogeneous particles useful in three dimensional morphing. We believe that this might very well be the case, if only for the simple reason that particle size could become one more feature which could be linearly interpolated, thus adding greater flexibility 3D morphing with oriented particle systems.

A problem with using oriented particles for morphing lies in that only surfaces may be modelled, and not actual solids. When a surface fully covers an object, this not particularly relevant since the impression exists that the object is solid, and its inside is filled and not empty. However, in the course of a morph, it is possible that a surface may rip. This happens, for example, in the transformation of a large single sphere into two smaller spheres. When this rip occurs, it becomes quite obvious that what is really being morphed is not a solid object but only a portion of a surface which can have holes due to ripping. In many cases this is not problematic. Unfortunately, in some cases this is not the visually intuitive result. Ways of remedying this phenomenon could be addressed by deriving an algorithm to fill in gaps due to ripping surface.

Furthermore it may be beneficial to extend our morphing approach to make sure that it is an exact one. In other words, it may be desirable to transform the initial system into an exact replica of the destination system. As already pointed out, our current approach does not guarantee this.

11.0 Appendix A - The Scripting Language

(1) ANIMATE <number_of_iterations> -

This command calls the animation routine to execute the differential equations the user specified number of iterations.

(2) CAMERA RESET -

This command resets the camera position and orientation to be the initial camera specification in world space.

(3) COMPUTE CORRESPONDENCE -

The command calls the algorithm to compute the micro correspondence partitioning algorithm.

(4a) CORRESPOND MOPS_OFF -

This command sets UI mode to the default. mouse mappings defined in the UI section.

(4b) CORRESPOND MOPS_MACRO - This command makes the UI mode to macro.

(4c) CORRESPOND MOPS_MICRO - This commands makes the UI mode to micro

(4d) CORRESPOND MOPS_BOTH - This command makes the UI mode: both(macro and micro).

(4e) CORRESPOND PARAMETERS MOPS_MACRO <submode> -

This command sets the MACRO MORPH submode to either EIGEN or RIGID. EIGIEN, computes a solution using the eigenvector method. The RIGID, 0, sets the MACRO submode to perform a rigid body transform based upon a vector alignment. EIGEN, 1, correspondences to aligning 3 orthogonal eigenvectors requiring no UI interaction.

(5) DEFAULT s= <sized_particle> -

The command brings up a plane of particles with the specified particle size.

(6a) DYNAMICS MOPS_ON -

This command enables the particle physics to apply forces to settle the particle system into a lower energy level.

(6b) DYNAMICS MOPS_OFF -This command disables the particle physics.

(7) FOCUS <integer> - This command provides focus to the user specified particle system.

(8) HIGHLIGHT <x> <y> <z> SIZE = <particle size> -

This function highlights all particles in the specified range.

(9a) INTERPOLATION GROW MOPS_ON - This command turns particle growing on.

(9b) INTERPOLATION GROW MOPS_OFF - This command turns growing particles off.

(9c) INTERPOLATION STRETCH MOPS_ON - This command turns stretch particle creation on.

(9d) INTERPOLATION STRETCH MOPS_OFF - This command turns stretch particle creation off

(10) LOAD "filename" - This command loads a particle system stored in the filename.

(11a) MAKE_TORUS <major_axis_size> <minor_axis_size> SIZE = <particle size> -

This command creates a torus with the specified particle size with a major and minor axis centered at the origin

(11b) MAKE_TORUS <major_axis_size> <minor_axis_size> <x> <y> <z> SIZE = <particle size> -

This command creates a torus with major and minor axis user specified located at x, y, z of user sized particles.

(11a) MAKE_SPHERE s =<size_particle> RADIUS_EQUAL <radius>

This routine generates a sphere composed of user sized particles. The location of the sphere is the origin in world space location.

(11b) MAKE_SPHERE <x> <y> <z> s =<size_particle> RADIUS_EQUAL <radius>

This routine generates a sphere of a user specified radius composed of user sized particles. The location of the sphere at a user specified location in world space location.

(12a) MORPH <how_many_iterations>

The function of this routine is to morph based upon the morph type (macro, micro, both).

(12b) MORPH "filename" <how_many_iterations>

The function of this routine is to morph based upon the morph type (macro, micro, both). The results of the morph will go to a file every time we reach the number of iterations asked for by the user. The resultant filenames will have the number of frame appended to the filename.

13a) PARTICLE <x> <y> <z> s=<size_particle> -

This command creates a <size_particle> particle located at x, y, z.

(13b) PARTICLE <x> <y> <z> <norm_x> <norm_y> <norm_z> s=<size_particle>-

This command creates a user sized particle located at x, y, z with normal components norm_x, norm_y, and norm_z.

(13c) PARTICLE NUMBER -

This command responds with the number of particles in the current system.

(13d) PARTICLE DELETE

The function of this routine is to

(14) STOP | GO

These two functions are useful to start and stop a morph currently in progress. This enables the user to change a morphing attribute to control a morph under progress.

(15) QUIT - The function of this routine is to quit the program execution.

(16) ROTATE AXIS <x> <y> <z> ANGLE <degrees>

This routine rotates, by a user specified number of degrees, the entire particle system about the user supplied axis.

(17) SAVE "filename"

The function of this routine saves a particle system with focus to a file.

(18) SCRIPT "filename"- The command reads in and executes a script file of this language.

(19) UI - This command brings up the User Interface.

(20) VISUALIZE "filename"

This function allows the user to cycle through a set of pre-cached morphing particle positions.

(21) VOLUME “filename” s=<size_particle> <ps_system_type> <threshold> <number_of_iterations>

This function calls the volumetric routine on a given file containing volumetric data. User’s can supply the size of the particle to use, as well as the heuristics of using volumetric data. The user specifies a homogenous(0) or heterogeneous(1) particle system in the ps_system_type. The user also specifies the allowable threshold for standard deviation and finally the user can specifies the number of settling iterations on the sampled volumetric data. For a full explanation of what these parameters are please refer to the section on volumetric processing.

12.0 Acknowledgments

First of all, we would like to thank our advisor, John "Spike" Hughes, whose input and suggestions proved to be invaluable in guiding our work. We also want to thank one of the original programmers, Marc Stevens, who worked with us to complete the initial surface modeling oriented particle system (SMOPS) at Brown University. Many thanks go out to Richard Szeliski and Dave Tonnesen whose original work inspired us. Finally, we thank the members of the Brown Graphics Group for their input and for their writing most of the libraries that we utilized.

This thesis was partly supported by Digital Equipment Corporation through the Graduate Engineering Education Program. I also want to thank my parents and friends. And of course, I extend many thanks to my wife Joanne for without her continued support, patience, and wonderful cooking this would not have been possible. Also I want to thank Kristen, my daughter, whose calls to the office kept me going. -Matt

I would like to thank my Mom and Dad, my brother Isaac, and my grandparents for their support and encouragement throughout my education and throughout my life. I would like to thank Kelly. For everything. And of course Alex and Dave. Without all of these people, I can't imagine why anything would be worth doing. -Harry

13.0 References

[Barr84] Barr, Al, "Global and Local Deformations of Solid Primitives", *Computer Graphics,* Vol. 18, No. 3, July 1984, pp.21-30.

[Beie92] Beier, Thaddeus and Shawn Neely, "Feature-Based Image Metamorphosis", *Computer Graphics (SIGGRAPH 1992),* Vol 26, No. 2, July 1992, pp.35-43.

[Beth89] Bethel, E. Wesley and Samuel P. Uselton, "Shape Distortion in Computer-Assisted Keyframe Animation," *State-of-the-art in Computer Animation: Proceedings of Computer Animation 1989,* N. Magnenat-Thalmann and D. Thalmann, eds., pp. 215-224.

[Bois84] Boissonat, J.D., "Representing 2D and 3D shapes with Delaunay triangulation," in *Seventh International Conference of Pattern Recognition (ICPR '84),* pp. 745-748, Montreal, Canada, July 1984.

[Burd93] Burden, Richard and J. Douglas Faires, *"Numerical Analysis,"* Plus-Kent Publishing Company, Boston, 1993.

[Chen85] Chen, Eric and Richard Parent, "Shape Averaging and Its Applications to Industrial Design", *IEEE Computer Graphics and Applications,* Vol. 9, No. 11, January 1989, pp. 187-196.

[Cocq90] Cocquillart, Sabine, "Extended Free-Form Deformations: A Sculpting Tool for 3D Geometric Modeling," *Computer Graphics (SIGGRAPH '90),* August 1990, pp.187-196.

[Corm90] Cormen, Thomas H. , Charles E. Leiserson, and Ronald L. Rivest, "Introduction to Algorithms," MIT Press, 1990.

[Ferg92] Ferguson, Helaman, Alyn Rockwood, and Jordan Cox, "Topological Design of Sculptured Surfaces, "SIGGRAPH 1992, pp. 149-156.

[Foley90] Foley, James D., Andries van Dam, Steven K. Feiner, John F. Hughes, *"Computer Graphics Principles and Practice, Second Edition,"* Addison-Wesley Publishing Company, 1990.

[Haumann88] Haumann, D. and Parent, R. "The Behavior Test-Bed:Obtaining Complex Behavior from Simple Rules, *"Visual Computer 4, 6,"* December 1988. pp. 332-347.

[Hugh92] Hughes, John F., "Scheduled Fourier Volume Morphing," *SIGGRAPH 1992*, pp. 43-46.

[Kent91] Kent, James, R. Parent, and W. Carlson, "Establishing Correspondences by Topological Merging: A New Approach to 3-D Shape Transformation", *Graphics Interface '91*, pp.271-278.

[Kent92] Kent, James R., Wayne E. Carlson, Richard E. Parent, "Shape Transformation for Polyhedral Objects," *SIGGRAPH 1992*, pp. 47-54, 1992.

[Knep92] Knep, Brian, "Mapping a 3-D Surface to the UV Plane for Texture Mapping, Patchifying and Metamorphosing," *Master's Thesis: CS-92-M1*, Brown University, 1992.

[Lore87] Lorensen, William E. and Harvey E. Cline, "Marching Cubes: A High Resolution 3-D Surface Construction Algorithm," *SIGGRAPH 1987*, pp. 163-169.

[Peachy85] Peachy, Darwyn, "Solid Texturing of Complex Surfaces", *SIGGRAPH 1985*, Computer Graphics, volume 19(3), pp.279-286.

[Platt87] Platt, John, Demetri Terzopolous, Alan Barr, and Kurt Fleischer, "Elastically Deformable

Models," *Computer Graphics (SIGGRAPH '87)*, July 1987, 20(4):205-214.

[Pres88] Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., "Numerical Recipes in C," *The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1988.

[Reev83] Reeves, W.T., "Particle Systems - a technique for modeling a class of fuzzy objects," *ACM Transactions on Graphics, 2(2)*, April 1983, pp. 91-108.

[Same89] Samet,, H., "The Design and Analysis of Spatial Data Structures," *Addison Wesley*, Reading Massachusetts, 1989.

[Sims90] Sims, Karl, "Particle Animation and rending using data parallel computation", *Computer Graphics (SIGGRAPH '90), 24(4)*, August 1990, pp. 405-413.

[Szel93] Szeliski, Richard, David Tonnesen, Demetri Terzopoulos, "Modelling Surfaces of Arbitrary Topology with Dynamic Particles," *CVPR '93*.

[Szel92] Szeliski, Richard and David Tonnesen, "Surface Modeling With Oriented Particle Systems," *SIGGRAPH 1992*, pp. 185-194.

[Szel91] Szeliski, Richard, and David Tonnesen, "Surface Modelling with Oriented Particle Systems," *Technical Report 91 / 14, Digital Equipment Corporation*, Cambridge Lab, December 1991.

[Terz87] Terzopolous, Demetri, John Platt, Andrew Witkin, and Michael Kass, "Symmetry Seeking Models and 3D Object Reconstruction," *International Journal of Computer Vision*, October 1987, (3):211-221.

[Terz88] Terzopoulos, Demetri and Fleischer, K., "Modeling Inelastic Deformation: Visoelasticity, Plasticity, Fracture", *Computer Graphics,* Vol. 22, No. 4, August 1988, pp.269-278.

[Turk90] Turk, Greg, "Interactive Collision Detection for Molecular Graphics", *Tech Report TR90-014*, University of North Carolina at Chapel Hill, 1990.

[Wolberg90]Wolberg, G. "Digital Image Warping" *IEEE Computer Society Press*, Los Alamitos, CA, 1990.

[Zele91] Zeleznik, Robert C., D. Brookshire Conner, Matthias M. Wloka, Daniel G Aliaga, Nathan T. Huang, Philip Hubbard, Brian Knep, Henry Kaufman, John F. Hughes, and Andries van Dam. "An object-oriented framework for the integration of interactive animation techniques," *Computer Graphics (SIGGRAPH 91)*, 25(4):105-112, July 1991.