Shape Interpolations with Unions of Spheres

VISHWA RANJAN and ALAIN FOURNIER
Department of Computer Science, University of British Columbia

Abstract

Shape interpolation is the process of transforming continuously one object into another. This is useful in applications such as object recognition, object registration and computer animation. Unfortunately, "good" shape interpolation is as ill-defined as "shape" itself. To be able to control the process in a useful way, we need a representation for the objects using primitives which capture at least some aspects of their shape, with methods to convert other representations to this one. We present here a method to interpolate between two objects represented as a *union of spheres*. We briefly describe the representation and its properties, and show how to use it to interpolate. Once a distance metric between the spheres is defined (we show different metric producing controlled effects), the algorithm optimally matches the spheres in the two models using a bipartite graph. The transformation then consists in interpolating between the matched spheres. If the union of spheres has been simplified, the other spheres are matched as a function of their positions within their representative cluster. Examples are shown and discussed with two- and three-dimensional objects. The results show that the union of spheres helps capture some notion of shape, and helps to automatically match and interpolate shapes.

Keywords: shape interpolation, stability, volume interpolation, union of spheres, matching, metamorphosis, registration

1 Introduction

Shape interpolation or object interpolation consists in continuously deforming one object into another. In addition to its aesthetic appeal (e.g., in "morphing" or metamorphosis [1, 5]), it can be used in visualizing the evolution of objects (e.g., growth of a tumor), in the registration of two views of the same object, or in automatic in-betweening in key-frame animation. The union of spheres (UoS) representation in which the set union of overlapping spheres is used to represent 3D objects is a computer representation of an object which has useful features for many applications [7]. In this paper we present an algorithm to interpolate between two UoS representations of the same or different object(s). The approach is to first match the spheres from one UoS representation to the other and then interpolate between the matched spheres.

2 Shape transformation problem

We want to define a continuous transformation between two objects A (the source object) and B (the destination or target object). Both objects are represented by a set of primitives, $\{a_1, a_2, \ldots, a_n\}$ for A and $\{b_1, b_2, \ldots, b_k\}$ for B. Generally the primitives a and b are assumed to be of the same type, but both the number of primitives n and k, and the topologies of A and B can be different. Without loss of generality, we assume $n \leq k$ in this paper.

It is difficult to measure the quality of or compare shape transformation algorithms in terms of the intermediate shapes that are generated, since it is already difficult to compare shapes themselves. Generally, the quality criteria are application dependent. For example, for effect animation it may be the aesthetic appearance of the transformation, for the visualization of the growth of a tumor, it might be the plausibility of the intermediate shapes and the evolution of the shape. However, we can list a few desirable properties (not necessarily all required at the same time):

- The surface of the interpolated objects should stay smooth and continuous, without tears or self-intersections.
- There should not be unnecessary distortions. This means for example that a large volume of one object should not transform into a small volume of the other.
- The algorithm should not generate too many intermediate primitives (only of the order of the number of primitives in the objects).
- It should allow some (intuitive) control over the transformation. The user should be able to select or modify some aspects of the interpolation.
- It should preserve as much as possible what is common between the two objects. For instance if A and B have the same topology, the intermediate shapes should have it too; if A and B are concave, so should be all the in-between objects.
- Neighbourhood and context should be preserved. If two matched pairs of primitives are adjacent, they should remain so in the interpolation.
- It should allow to interpolate other attributes such as color or texture whenever possible (sometimes different shape or topology make it impossible).
- It should be efficient and easy to implement.

3 Current approaches

Algorithms for both surface (facet-based, polygonal, polyhedral) and volumetric (scalar field on a discrete 3D grid) descriptions of 3D objects have been described in the literature [2, 5, 4, 3, 6]. Essentially, there are two classes of algorithms to perform 3D shape interpolation. Some algorithms first establish a correspondence between the primitives of the two objects (often after bringing them to some canonical position) and then interpolate between the matched primitives. The other class of algorithms do not match explicitly, but use a global transformation acting on all the primitives at once. The main difficulty with the first class is that the matching process is difficult and can be unstable. The main difficulty with the second is that there is little control over the intermediary shapes. We will discuss here a few representative methods.

3.1 Hong et al. (1988): Matching faces

This approach works on facet-based surface descriptions of objects (polyhedra). The faces in the two objects are matched by first computing the centroid of each face followed by an attempt to minimize the sum of distances of the centroids in the absolute or L_2 norm by a minimal dynamic displacement algorithm [2]. If there is not an equal number of faces in the two objects, multiple copies of some faces are created in the object with less faces. Once the face matching is done, the vertices in the matched faces are matched and the matched vertices are finally interpolated. The biggest problem with this approach is that there is no guarantee that the intermediate shapes will have a continuous surface.

3.2 Kaul and Rossignac (1991): Using Minkowski sum

One interesting way to do shape transformation is through the use of the Minkowski sum of the objects represented as polyhedra [4]. The Minkowski sum is in some sense the convolution of two objects. If $A+B$ is the sum of the two objects, the interpolation is then conveniently expressed as $tA + (1-t)B$. As the parameter t changes from 1 to 0, the object changes from A to B. This approach requires no explicit matching and works regardless of the topology of the two objects. Kaul and Rossignac show how the interpolated object can be computed and prove that the boundary of this object is always continuous. The approach is very storage and computation intensive. Another problem is that even if the two objects are similar, the intermediate shapes can be very different.

3.3 Kent et al. (1992): Matching the vertices

This approach works only with polyhedra of genus 0. The algorithm consists in mapping the polyhedra to a sphere, followed by a merging of the two maps so that the two models have the same number of vertices. Finally, the vertices are interpolated [5]. The quality of the intermediate shapes is strongly dependent on the quality of the vertex matching on the sphere. Neighborhood, connectivity and other information have to be taken into account during the matching step. One possible advantage is that other surface properties (color, texture etc.) can be interpolated. The main disadvantage is that it works only for objects of genus 0.

3.4 Payne and Toga (1992): Distance field manipulation

In this approach, the surface models are first converted to a volumetric representation (voxels) by calculating the distance from the boundary of the object for every grid point of an overlaying 3D grid. To interpolate between the two models, the discrete values in the scalar distance field are interpolated. At every step the intermediate object is defined as an isosurface in the interpolated distance field [6]. The approach does not require matching and is simple to implement, but the method requires an isosurface computation at every step, and that can be expensive.

3.5 Hughes (1992): Using Fourier analysis

This method works on source and destination objects both represented by volumetric data. First a Fourier transform is applied to the data and the high frequencies are removed from the source object. This is followed by interpolation between the low frequencies of the source object to the low frequencies of the destination object. Finally, the high frequencies of the destination object are slowly introduced [3]. The objects can be displayed using volume rendering techniques or by isosurface extraction. The method works for objects of any topology but is computation intensive.

3.6 Using specialized matching primitives

This is probably the oldest and most popular way. The primitives can be strokes, skeletons, members of an articulated model, or segments drawn by the user who determines the matching. This has been used in key-frame interpolation [11], in-betweening [12] and morphing [1]. This is also used in the context of hierarchical models [13].

It is clear from this brief survey that none of these approaches deal with “shape” primitives, and as a result there is little control over the result, except in the last category, where the control is obtained by direct human intervention in defining and/or matching the primitives. We will see that the UoS representation is a step in the right direction.

4 Shape interpolation using union of spheres

4.1 The Union of Spheres (UoS) representation

One of the basic properties that a representation should have to be related to shape is that of *stability.* Stability means that a small variation in the input data will result only in a small variation in the representaion. While it is a necessary condition, it is not a sufficient one.

The combination of stability and the use of *volume primitives* is the motivation behind the UoS representation. Ranjan and Fournier showed how we can obtain stable UoS models for objects from volumetric (e.g., CT or MRI scans) data [7], from range-finder data, and from a ray-traceable model of an object. This includes all object representations currently used in computer graphics. We propose to show that the UoS representation can be used advantageously for shape interpolation. Here are some of the reasons:

- As said above, we can transform all other representations into UoS.
- The UoS can be simplified, and therefore the interpolation can be made at various level of details.
- Since spheres are closed primitives, problems of tearing and self-intersection are avoided.
- The UoS is stable. This means that the intermediate shapes will be independent of noise etc. that is introduced during the model acquisition phase.
- The intermediate shapes will have the same number of spheres as that of B (the one with larger number of spheres). This is generally more compact than the number of vertices and faces.
- It is possible to define metrics that affect the quality of the interpolation in predictable ways. Also feature-level matching is possible because the distance measure can take each sphere's context into account.

4.2 The Algorithm

In most applications, it is useful to distinguish between a transformation, where one object is translated, rotated, scaled, or generally transformed globally to bring it in registration with the other object, and the interpolation, where the shape of one object evolves towards the shape of the other. This distinction is of course arbitrary, being mostly semantic, but it corresponds to different actions in most approaches. Symbolically we can call T_r the transformation to bring the objects into registration, and T_i the interpolation. These two transformations can either be actually separated, or put together in the interpolation step. If we restrict T_r to the class of affine transformations, then the goal might be to minimize the distance between objects by affine transformations alone (which can be done by least square fit after matching [8]), and then do the rest with T_i. Of course T_r itself can be interpolated. In particular, if A and B are the same objects (differ only in scale, rotation, translation, and resolution), T_i is identity.

Our approach in summary consists of the following steps. First we compute the UoS representations for the two objects, and simplify them to a level where they have approximately the same number of spheres (this number is user specified). More details on these steps can be found in [8]. Then we transform the objects to bring them into a common canonical position. The next step is to assign distances between every sphere $a \in A$ and every sphere $b \in B$ according to a pre-defined metric. Then the matches between the two sets of spheres are obtained by formulating the problem as a minimum weight bipartite graph matching (or "assignment") problem. Finally, we interpolate between the matched spheres. The last four steps will now be described in more details.

4.2.1 Step I: Object transformation

This can be manual or automatic. In the manual case, the user is allowed to scale, rotate, and translate the two models in the world space. In the absence of user intervention, we use the method of central moments to align the two objects [9]. The objects are translated such that their centers of masses match, they are scaled such that their first moments or volumes match and they are rotated such that their local axes established from second order moments match. To estimate the moments and volumes we scan convert (i.e., overlay a 3D grid over) the two representations keeping track of grid points inside the object. The center of mass of the object is estimated by taking the average of all the interior grid points.

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i, \quad \overline{z} = \frac{1}{N}\sum_{i=1}^{N} z_i$$

The eigenvectors of the following "covariance" matrix are used to attach a local coordinate system to the objects.

$$\boldsymbol{M} = \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} (y_i-\overline{y})^2+(z_i-\overline{z})^2 & (y_i-\overline{y})(x_i-\overline{x}) & (z_i-\overline{z})(x_i-\overline{x}) \\ (x_i-\overline{x})(y_i-\overline{y}) & (x_i-\overline{x})^2+(z_i-\overline{z})^2 & (z_i-\overline{z})(y_i-\overline{y}) \\ (x_i-\overline{x})(z_i-\overline{z}) & (y_i-\overline{y})(z_i-\overline{z}) & (x_i-\overline{x})^2+(y_i-\overline{y})^2 \end{bmatrix}$$

Because the matrix M is symmetric, its eigenvectors are orthogonal to each other. The eigenvalues of the matrix are used to put an ordering on the three eigenvectors. The z-axis is aligned with the eigenvector corresponding to the smallest eigenvalue (axis of least inertia), y-axis to the next, and x-axis to the axis of largest inertia. This method works well in most cases. We should note that it is the same preliminary step used to register two objects with UoS, and that we are currently studying ways to match features of the two UoS.

4.2.2 Step II: Distance computation

After alignment, we calculate the distances $d(a,b)$ between every $a \in A$ and $b \in B$. We define a bipartite graph where nodes correspond to the spheres in A and B respectively, and the weight on the edges are the distances between them (Figure 1a). In our examples the distance between two spheres is a function of their location (coordinate of centers after transformation) and size (radius after transformation). Specifically:

$$d(a,b) = \alpha((x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2) + (r_a - r_b)^\beta \tag{1}$$

where, α and β are parameters. By varying α and β, we can assign different weights to the size and location of primitives. This function strictly depends on the geometry of the spheres. When $\beta = 0$ it uses position alone, and when $\alpha = 0$ it uses size alone. Clearly other functions can be used for different effects. Also, for some problems, the distance function might be suggested by experimental evidence.

4.2.3 Step III: Primitive matching

The problem here is to find a *maximum match* such that: $\sum_{a\in A, b\in B} d(a,b)$ is minimum. A maximum match in a bipartite graph means that all n nodes of A should get matched to exactly n nodes of B. This problem has been widely studied and can be solved exactly in $O(n^3)$ time by reducing it to a network flow problem [10], where n is the number of nodes (in our case, the number of spheres in the representations) in the bipartite graph. The remaining $k - n$ spheres of B can be matched in different ways. In most cases matching them to the sphere in A which matches their nearest neighbour in B causes those extra spheres to "disappear" during the interpolation. After matching, the graph is a bijection from A to B, i.e., every sphere of A is matched to at least one sphere of B, and every sphere of B is matched to exactly one sphere of A (Figure 1b).

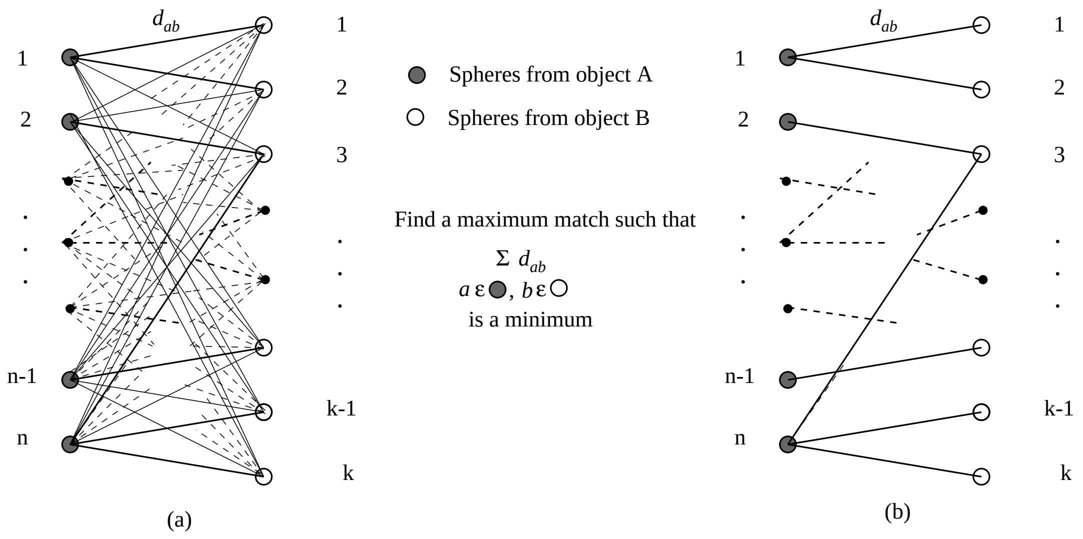

Figure 1: (a) Bipartite graph after distances are assigned. (b) Bipartite graph after matching.

4.2.4 Step IV: Interpolation

To interpolate between object B and object A we interpolate between the matched spheres. This interpolation affects both the centers and radii of the spheres, and the rate at which each changes can be chosen at will. Some constraints can be to vary the radii linearly, the volume of each sphere linearly (i.e. the cube of the radius), or even the total volume of the UoS linearly. The latter is expensive, since it is very difficult to go from the volume to the position of spheres (there is in general no unique solution). There can be external reasons, from domain specific knowledge, to determine these rates, as for instance for experimentally determined growth curves. In the absence of such knowledge, we found that linear interpolation between the radii of matched spheres produces decent results.

4.3 Observations

The complexity of the overall algorithm is $O(n^3)$, where the matching step dominates. The matching step can be trivialized by selecting the nearest neighbor in the other set for all spheres, but the subjective quality of the resulting interpolation is low. Our method is intermediary between direct matching and global interpolation, since the spheres overlap, and the large spheres are representatives of the more global aspects of the shape, while the small spheres provide the details.

5 Implementation and results

The programs for shape interpolation in 2D and 3D have been implemented in the C programming language using the GL graphics library and forms, and compiled for SGI and IBM RS/6000 platforms. We illustrate with two examples: (i) a rectangle transforming to a slice of brain (obtained from MRI), (ii) a rabbit (obtained from multiple-view range data) changing into a human head (obtained from multiple-view ray-traced data). In 2D, the two objects are represented as union of circles (rectangle: 62 circles, brain: 228 circles), and in 3D as union of spheres (rabbit: 1046 spheres, human head: 262 spheres) (Figures 2a, 2f, 3a, 3f). These objects were obtained and simplified using the algorithms described in [8].

Step I: In our implementation the alignment can be manual or automatic or a combination of the two. For results in 2D the alignment was automatic—it took about 0.1 second in total to do the alignment. For 3D, the alignment was semi-automatic. First, we aligned the objects by moments and then rotated one of the objects to proper (that is semantically appropriate) orientation. The estimation of moments for both objects took about a total of 1.5 seconds.

Step II: A number of metrics have been tried.

We experimented with varying α and β in equation 1. For both our examples we chose the metric with $\alpha = 1$, and $\beta = 1$. Calculating the distances took a total of about 8 seconds for 2000 spheres in total. Whereas this metric seems to work quite well, other context or feature-based metrics will do better for specific puposes, for instance to match the rabbit's ears to the human/vulcan ears.

Step III: The most time consuming part of our algorithm is the matching step. For matching a total 2000 spheres it took about 1.5 minutes on an IRIS Crimson.

Step IV: Figures 2b-e show 4 images in the interpolation from rectangle to brain. Similarly, Figures 3b-e show the in-betweens for the rabbit interpolating to a human head. We chose the linear interpolation in radius so that the linear dimensions of the object increase by about equal amounts for each time step. We found this to be more pleasing than an interpolation where the volumes of the primitives is increased linearly. While in 2D the display for above examples is near real-time, in 3D it is not—the display rate for 1000 spheres (100 polygons per sphere, Gouraud-shaded) is about 2 frames per second on our Crimson. Since the matching and interpolation in our examples is done on the simplified UoS representation, the rendered version shown here lacks details. In 3D, the display of the union of spheres can be smoothed by using blending functions (i.e., using blobby display [7]). The accompanying video shows these two "morphings" as smooth transformations.

6 Conclusion

We showed that the *union of spheres* representation can be easily and significantly used to interpolate between objects, whether they are very different in shape or not. In previous work we showed that this representation can be computed from just about all other representations, such as volumetric data, polygonal meshes, range finder data, parametric and implicit surfaces. The essential steps are transforming the objects to a canonical position, matching the spheres, and interpolating between the spheres. The correspondence between the spheres of the two objects is established by performing a minimum weight bipartite graph matching using a user-specified distance metric. Since this representation is linked to the shape of an object, and the user has control over the distance metric, we feel that the quality of the intermediate shapes can be adequately controlled. More weight can be given to distance between spheres or to differences in radius. Increasing the former stresses locality, while increasing the latter stresses a form of shape matching. One problem with our algorithm is that it is slow for large number of spheres because the time complexity of matching is $O(n^3)$. However, this problem can be lessened because the UoS representation can be simplified, and because we can speed of the process by not requiring optimal matching. We are currently investigating the use of feature matching both to determine the geometric transformation (also an essential step in registration, that is matching two versions of the same object) and to provide a different distance metric for the interpolation. The features are defined by a scalar field deduced from the UoS itself. Another direction of research is to match the original colour, texture and surface normals of the objects as they are simplified, transformed and interpolated.

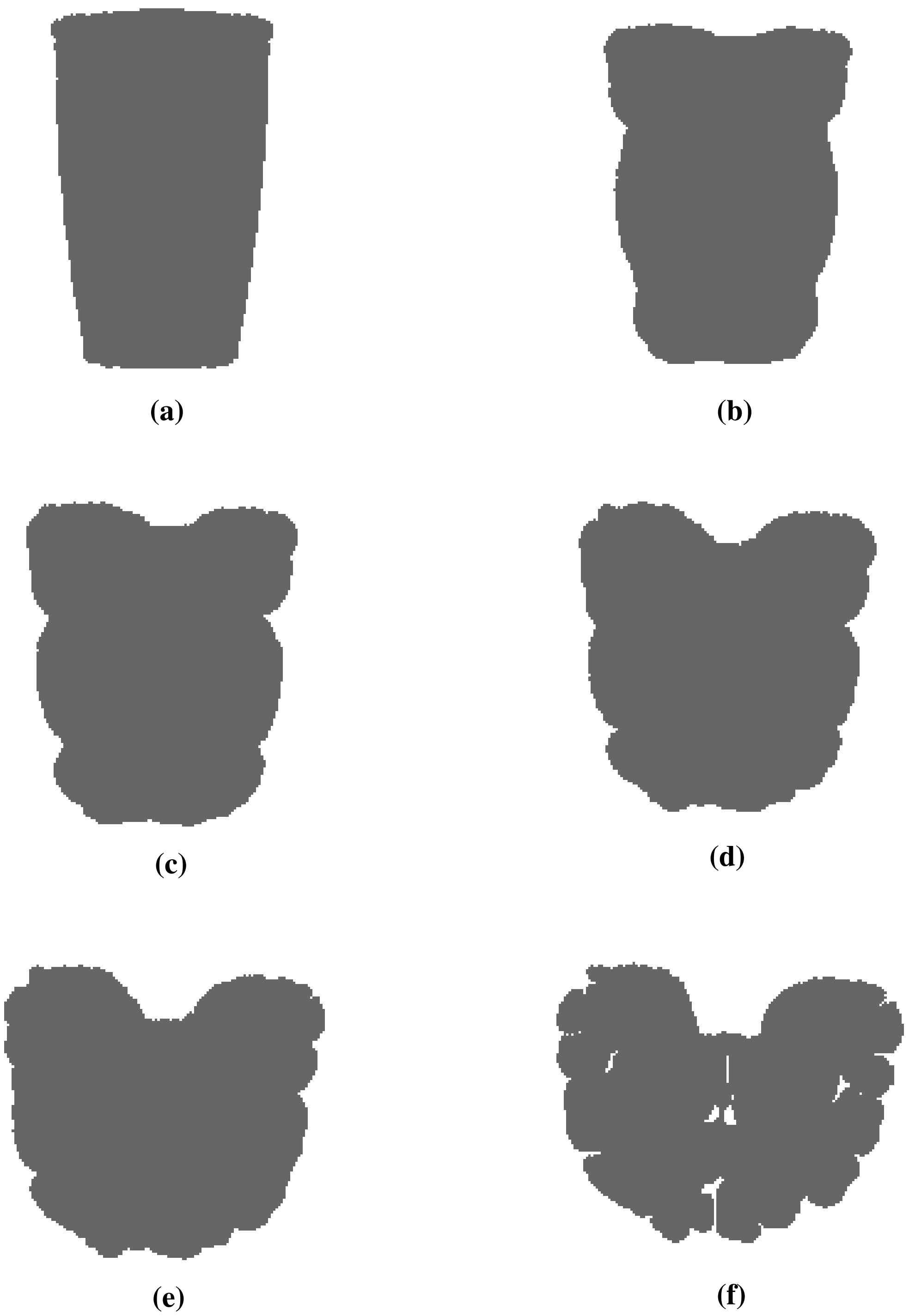

Figure 2: Transformation from a rectangle to a slice through brain.

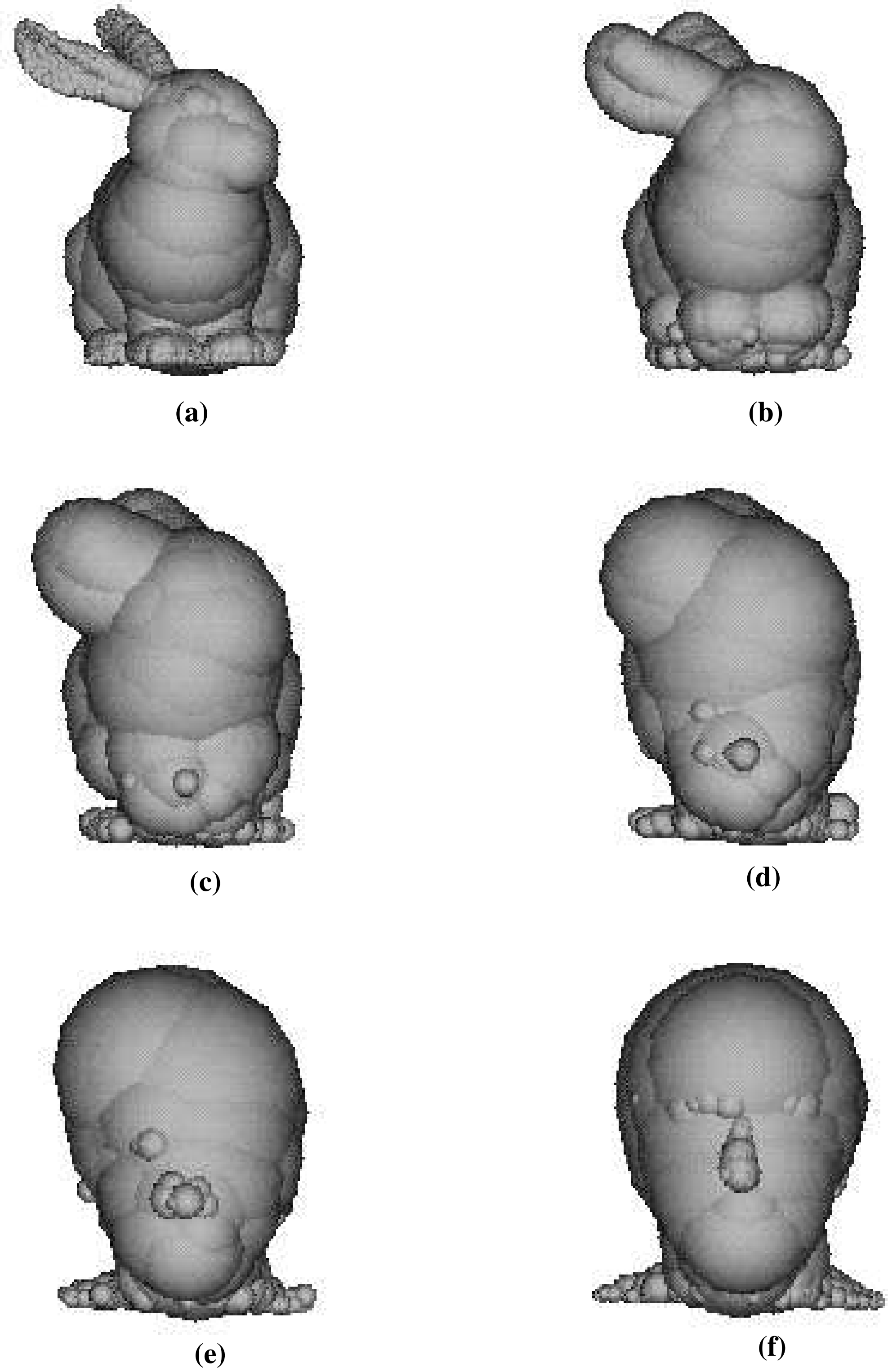

Figure 3: Transformation from a rabbit to a human head.

Acknowledgements

We gratefully acknowledge the help of the Natural Sciences and Engineering Research Council of Canada and the Government of British Columbia for their generous support. IBM Canada donated to GraFiC the RS/6000 workstations on which most of the animation illustrating this paper was computed, and we are grateful. Robert Kennedy kindly provided the code for solving the assignment problem. Cyberware and Stanford University provided the range data for the rabbit and for Spock head.

References

[1] T. Beier and S. Neely, "Feature-based image metamorphosis", Proceedings of SIGGRAPH '92, *Computer Graphics*, Vol. 26, No. 2, July 1992, pp. 35–42.

[2] T.M. Hong, N. Magnenat-Thalmann, and D. Thalmann, "A general algorithm for 3-D shape interpolation in a facet-based representation", Proceedings of *Graphics Interface '88*, June 1988, pp. 229–235.

[3] J. Hughes, "Scheduled Fourier volume morphing", Proceedings of SIGGRAPH '92, *Computer Graphics*, Vol. 26, No. 2, July 1992, pp. 43–46.

[4] A. Kaul and J. Rossignac, "Solid-interpolating deformations: construction and animation of PIPs", Proceedings of *Eurographics '91*, pp. 493–505.

[5] J.R. Kent, W.E. Carlson, and R.E. Parent, "Shape transformation for polyhedral objects", Proceedings of SIGGRAPH '92,*Computer Graphics*, Vol. 26, No. 2, July 1992, pp. 47–54.

[6] B. Payne and A. Toga, "Distance field manipulation of surface models", *IEEE Computer Graphics and Applications*, Vol. 12, No. 1, Jan. 1992, pp. 65–71.

[7] V. Ranjan and A. Fournier. "Volume models for volumetric data", *IEEE Computer*, Vol. 27, No. 7, July 1994, pp. 28–36.

[8] Ranjan, V., and Fournier, A., "Union of spheres (UoS) model for volumetric data", short communication, to appear in the proceedings of the *Eleventh Annual Symposium on Computational Geometry* to be held in Vancouver, Canada.

[9] F. Solina and R. Bajcsy, "Recovery of parametric models from range images: The case for superquadrics with global deformations", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, Feb. 1990, pp. 131–147.

[10] R. E. Tarjan, *Data-structures and network algorithms*, SIAM, Philadelphia, 1983.

[11] N. Burtnyk and M. Wein, "Interactive skeleton techniques for enhancing motion dynamics in key frame animation", *Communications of the ACM*, Vol. 19, 1976, pp. 564–569.

[12] W.T. Reeves, "Inbetweening for computer animation utilizing moving point constraints", *Computer Graphics (SIGGRAPH '81 Proceedings)*, August 1981, pp. 263–269.

[13] D. Forsey, "A surface model for skeleton-based character animation", *Eurographics Workshop on Animation and Simulation*, Sept. 1991, pp. 55–73.