Volume-based three-dimensional metamorphosis using region correspondence

G.M. Treece, R.W. Prager and A.H. Gee

CUED/F-INFENG/TR 379

April 2000

Cambridge University Engineering Department
Trumpington Street
Cambridge CB2 1PZ
England

E-mail: gmt11,rwp,ahg@eng.cam.ac.uk

Abstract

The metamorphosis (commonly known as morphing) of one image into another is a well studied subject that has frequently been used to create impressive visual effects. There has been significant work recently in three-dimensional (3-D) metamorphosis, where a surface is transformed into another surface. One approach is to construct discrete volume representations of the surfaces and interpolate between them to create intermediate volumes. With the advance of volume graphics, these volumes can be directly rendered in real time, or alternatively one of the many isosurface extraction techniques can be used to construct a polygonal mesh from each volume for rendering using standard hardware. This technique is independent of the surface topology, which enables morphing between surfaces of completely different shape — however it can also lead to unrealistic intermediate surfaces which have different topology from the originals. We present a method for correcting this problem by automatically calculating region correspondence derived from a representation of each surface as a set of spheres. We also show how this can be combined with a simple method for manually defining correspondence. In combination, this proves to be a fast and flexible method for morphing surfaces, as demonstrated on a wide range of examples.

Contents

1 Introduction

Image metamorphosis (or morphing) is the gradual transformation of one image to another, usually by a combination of warping and interpolation, recently reviewed in [8, 29]. Such techniques are popularised by their use in the film industry, where the morph is used to create the effect that one image 'becomes' the other. However, morphing can also be viewed as data interpolation, and similar techniques have been used within the medical imaging community to interpolate between parallel slices of data [21, 22, 23].

Three-dimensional (3-D) metamorphosis is the gradual change of one *surface* to another. There has been less work on this subject, and it has been reviewed only recently [15]. 3-D morphing can be used for shape transformation (morphing between different objects), animation (interpolating intermediate surfaces between the same object in two different positions) and surface synthesis (editing an existing shape by morphing it with a new one). There is also a close link between the interpolation of 2-D cross-sections to form 3-D surfaces, and the morphing of 3-D surfaces to form 4-D time series.

A major advantage of 3-D over 2-D morphing is its independence of viewing and lighting parameters — the model itself is morphed, and the rendering is performed on the morphed models. In 2-D morphing, the rendering has already been done in creating the initial images, and hence it is impossible to maintain correct lighting, or move the viewpoint, during the morph. There are also situations where the aim is to create intermediate *models* rather than images, for instance improving a model of a beating heart by interpolating sampled data, and in this case there is no option other than to use a 3-D morph.

We present a method for morphing 3-D objects which is an extension of a technique already used to interpolate surfaces from 2-D contours [27]. This technique is relatively fast, and integrates automated and manually defined object correspondence in order to improve the definition of the morphing sequence. It is a discrete volume based approach, where each volume element (voxel) is defined in terms of the minimum distance to the surface. Although volume graphics (the storage and display of volumetric data as opposed to polygonal surfaces) has advanced considerably over the last decade [14, 24], most surfaces are still defined as polygonal models, and most graphics hardware is designed to render such models. Hence we also describe techniques for converting between polygonal models and volume data.

In Section 2 we review the classes of 3-D morphing techniques, concentrating on volume based approaches, and explain the connection to surface interpolation from 2-D cross-sections. Section 3 contains an overview of our approach, which is explained in detail in Section 4. The user interaction is an important aspect of this design, and the pertinent features are explained in Section 5. The algorithm is tested on a range of surfaces in Section 6 and conclusions are drawn in Section 7.

2 Background

2.1 3-D volume metamorphosis

3-D morphing algorithms can be split into three major categories [8, 15], according to the underlying representation of the surfaces being morphed. Polygonal representations can be morphed by interpolating the location of the surface points [1, 9, 16]. Implicit surfaces (i.e. those defined as an isosurface of a function generated from a set of primitives) can be morphed by interpolating the underlying primitives. Surfaces defined as isosurfaces of volume

data can be morphed by interpolating the volume data. Each strategy has its own strengths and weaknesses and in general it is possible to convert between representations in order to take advantage of these. In particular, most models are initially defined as polygonal meshes, and these can be converted to implicit representations (e.g. the skeleton representation [2] and union of spheres representation [20]), or to volume data [24].

Since volume representations are topology and shape independent (the underlying arrangement of the data stays the same — only the values change) they are inherently suited to handling complex morphing sequences. Each voxel contains a signed minimum distance to the surface, such that positive distances are considered to be inside the surface and negative outside. The surface itself is defined as the zero isosurface (or level set) of this volume. Intermediate volumes can be created by linear interpolation [19, 28]. Alternatively, new surfaces can be created by isosurfacing the initial volumes at a different value, hence generating *offset* surfaces [4].

Unfortunately, this independence from topology and shape also makes it difficult to constrain the intermediate surfaces to have the same topology and shape as the originals. Intermediate objects can only exist in regions where the original objects overlap; where this is not the case, one object disappears and the other appears during the morph, creating a gap in the sequence. The visual effect is similar to cross-dissolving an image — one object 'dissolves' into another, but there is no sense of movement of any part of either object. The challenge in volume based morphing techniques therefore lies in controlling the way in which the volumetric data is interpolated in order to overcome this problem.

In one approach, the volumes are transformed into the frequency domain using either wavelets [10] or Fourier transforms [11]. Interpolation is then performed in the frequency domain before being transformed back to the spatial domain for viewing. The main advantage of these methods is in allowing information at differing frequencies to be transformed at different rates, usually with the emphasis on the low frequencies, which can improve the quality of the morph. General correspondence between the objects can also be established from the low frequency data. However, it is difficult to control the high frequency distortions that can be introduced, and there is little or no scope for user interaction to control the morphing sequence — both the cited methods are entirely automatic. Furthermore, the transformation to and from the frequency domain is both memory and time consuming.

The alternative approach is to warp the source and target volumes before interpolating them, or equivalently to vary the interpolation direction across the volume. In this case, the correspondence between the two volumes is defined manually, either with matching feature 'elements' [17] (consisting of points, lines, rectangles and boxes), or with matching oriented discs [5] or matching points [6]. Correspondences between various elements are combined by simple inverse distance weighting [5, 17] or used to calculate a smooth warping field [6]. In each case, the quality of the morphing sequence is entirely dependent on the manual definition of correspondence.

In this work, we aim to prevent unrealistic morphs by providing automation of correspondence, as in the former approach, whilst maintaining the user interaction inherent in the latter.

2.2 3-D metamorphosis as an extension of surface interpolation

The reconstruction of surfaces from cross-sectional data is a very well studied problem. Some of these techniques involve discretisation and interpolation to form intermediate cross-sections,

which can then be reconstructed to form the object surface [27]. These techniques can be adapted to 3-D volume morphing simply by extending the interpolation from 2-D to 3-D, thus creating a 4-D (time varying 3-D) sequence from a 3-D volume, rather than a 3-D volume from a 2-D slice. There are, however, some differences in assumption and philosophy that complicate this otherwise natural extension.

Firstly, when generating a morphing sequence between two objects, it is generally not desired that objects, or parts of objects, should appear out of nothing, or alternatively disappear into nothing, since this has no parallel in reality and hence tends to make the morph unrealistic. However, it is frequently the case that two *cross-sections* of the same object have parts that should not be connected; consider for instance two horizontal cross-sections of the letter 'J' — the upper consists of a single circle, and the lower of two, but only the right hand circle should be connected to the upper cross-section. This is because objects, or parts of objects, can be (and generally are) disjoint in *space*, hence cross-sections of these objects will also be disjoint. However, objects are never disjoint in *time*. The assumptions underlying the calculation of correspondence of objects are therefore not the same as those which underly the correspondence of cross-sections.

Secondly, although both problems are highly under-determined and have many possible solutions, it is accepted that in surface interpolation there is some 'optimum' surface (even if we do not in fact know what that optimum is), and user interaction in generating this surface is undesirable. This is because the cross-sections are generally of some physical structure, and hence there is a 'right' solution to the surface interpolation problem. Many applications of morphing, however, are not representative of real situations, and the 'right' solution can be most easily defined as 'what the user wanted'. The issue here is therefore to provide users with an intuitive way of describing the morph, rather than making the decisions for them. Having said this, there are many morphs which everyone would agree look 'wrong' and there is hence much scope for some automation to ensure the user is selecting between 'right' morphs rather than trying to avoid 'wrong' ones.

The third difference is related to the second, in that the desirability of user interaction introduces a new challenge to the problem of 3-D morphing, especially since that interaction is with 3-D time varying data and most display and input technology is designed for 2-D. This applies equally to the manual definition of correspondence, and to the assessment of the morphing sequence itself.

3 System overview

Figure 1 shows the main steps in the morphing of two polygonal models, and at which points the user can interact with this process (though not in the same order as is presented to the user — see Section 5). The first step is to create the discrete volume representations from the polygonal models. The accuracy of this process determines the quality of the volume data, and hence also the final morphing sequence: or conversely it determines the size of volume that will be required for a given model quality. It is at this point that the alignment and scale of the two models is set[1] — this is controlled by the user. Secondly, sphere representations are calculated from the two volumes. The user can control the coarseness of this representation, although in practice the default value is usually acceptable.

[1] Alignment and scale could equally be determined after conversion to the volume representation, however realigning the volume data introduces sampling errors which realigning the polygonal models avoids.

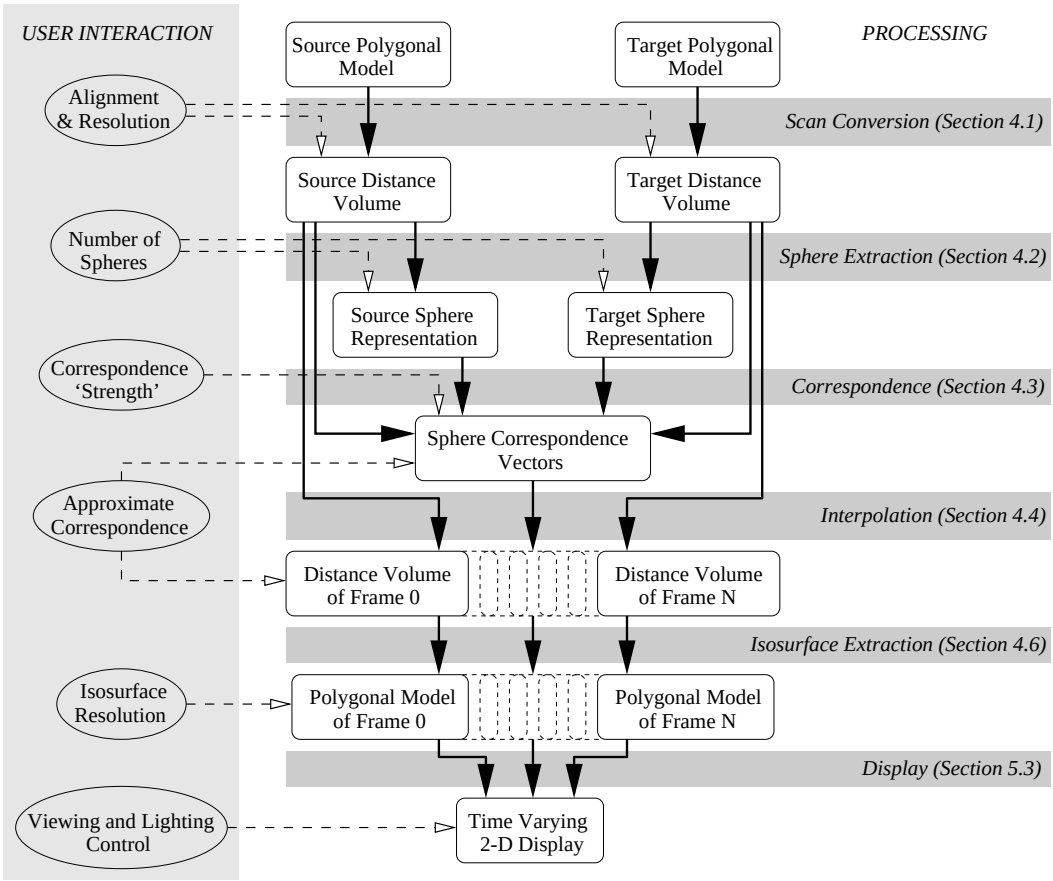
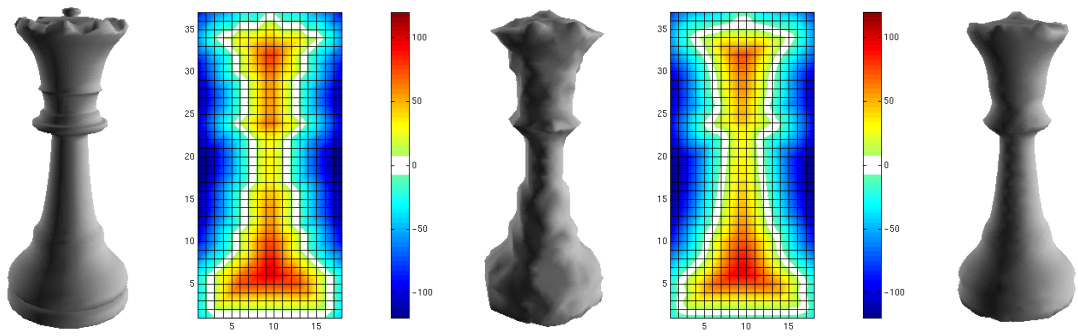


Figure 1: **System overview.** The morphing process begins with two polygonal models (top of diagram) and ends with a real time display of the morphing sequence. User interaction with this process is shown on the left. The processing steps (and sections within this report where they are explained) are shown on the right.

Correspondence between the two objects is then established by using the sphere representations. There are two important features of this correspondence. Firstly, it is not a mapping between spheres as in [20], but rather a correspondence *vector* is established for each sphere, which is affected by all spheres in the other object. Secondly, since the correspondence is between spheres and not between the original objects, this correspondence is calculated at a *regional* level — i.e. it is not simply a mapping of one object to another. The correspondence calculation can therefore improve the morph even if there is only one object in each model, by determining which parts of which object should be connected. The user controls how 'connected' the morph is allowed to be by varying the 'correspondence strength': see Section 4.3.

Any number of intermediate volumes can then be interpolated from the originals, using the sphere correspondence to guide the interpolation. These are isosurfaced (in addition to the source and target volumes) to create polygonal models which can be rendered. In practice, the isosurface triangulation is performed as each intermediate volume is interpolated, and it is this, rather than the volume itself, that is stored. Real time display of the morphing sequence



(a) Original (b) Binary scan converted (c) Anti-aliased

Figure 2: **Effect of model conversion.** (a) The original model is defined as a polygonal mesh containing 3600 polygons. (b) and (c) show the results of scan converting this into a volume of only $18 \times 18 \times 37$ voxels, using binary and anti-aliasing techniques respectively. This is represented, in each case, by a slice through the volume (with linear interpolation of the values at each voxel), and the re-triangulated zero isosurface (containing approximately 2000 polygons).

can then be achieved by looping through the sequence of models each time the display is rendered, during which the viewing and lighting can be interactively adjusted by the user.

Each of the steps in this process is discussed in more detail in Section 4, together with how manual definition of approximate correspondence is combined with these to allow more control over the morph.

4 Algorithms in detail

4.1 Conversion to volume-based representation

The signed minimum distance to surface representation which we have described is not the only possible volume based representation, indeed others exist which can represent non-closed surfaces within a volume [24]. In this case, however, the morphing algorithm requires that the surface must be closed, and it is convenient to define the surface at the zero (rather than some other value) threshold. Intensity-defined volume data can be converted to this representation by thresholding followed by distance computation, or more accurately converted by using Constrained Elastic Surface Nets [7]. Implicit surfaces are generally already defined by a distance function, and this function simply has to be evaluated at each voxel location. The conversion of polygonal meshes is examined in the next section.

4.1.1 Conversion of polygonal meshes

In order to be representable as a distance volume, a polygonal model must itself be closed, i.e. it must separate space into *outside* and *inside* regions. In addition, the surface must not be self-intersecting, since this implies a contradictory definition of signed minimum distance at the intersection. The triangles must also be correctly oriented (i.e. the vertices are defined

in a consistent order with respect to the direction of the outward surface normal)[2]. These criteria add constraints to the definition of polygonal models which might be considered good practice, but are not necessarily required if the model has only to be rendered.

Given that a polygonal model is suitable for representation as a distance volume, the simplest way to convert it is by using a two step process. In the first step (scan conversion), which voxels in the volume are inside and which are outside the surface is determined, and in the second (distance transformation), the distance of each inside voxel from the nearest outside one, and vice versa, is estimated. Both scan conversion [13] and distance transformation [3] can be performed very efficiently, the former by a single pass through the polygons and then the volume, the latter by two passes through the volume. Figure 2(b) shows the results of this process for the generation of a low resolution volume from the source model in Figure 2(a). The binary nature of the process is immediately apparent in both the distance volume itself and the triangulated isosurface. The distance values are calculated from voxel borders rather than the original surface, and hence the voxels are themselves apparent on the surface, although this effect is reduced by the use of a high quality isosurfacing algorithm.

The quality of the surface could be improved by calculating the distance volume at a much higher resolution, however this also dramatically increases both the storage requirement and processing time for the morph. It is much more efficient to use an anti-aliasing algorithm to initialise the distance volume, rather than binary scan conversion — this results in the much improved distance volume and surface of Figure 2(c). The algorithm we use is a two step technique which initialises the voxels near to the surface with the exact (Euclidean) signed minimum distance, then propagates this information to the remainder of the volume using a chamfer estimate [3]. The two step nature of this process is similar to a technique employed for Constructive Solid Geometry [4]. We use an algorithm which, like binary scan conversion, only requires one pass through the polygons making up the model (rather than testing each of the polygons at each voxel location, as in [12]).

For each polygon, each voxel inside the 3-D bounding box surrounding the polygon is examined, and the distance to the triangle calculated, up to a pre-determined maximum. If it is over this maximum, the voxel remains un-initialised. The closest distance to a polygon in 3-D space can be either to the plane containing the polygon, one of the vertices of the polygon, or one of the edges. It is most efficient to calculate the distance to the plane containing the polygon, then project the point onto that plane and calculate the remaining distances in that plane. An additional advantage of this technique is that if the distance to the plane is already greater than the maximum initialisation distance, there is no need to proceed any further. The distance from a voxel to a polygon is stored if it is less than the initialisation distance, and the voxel is not initialised or contains a greater distance value. In practice, an initialisation distance of 1.1 (with respect to the width of a voxel) has been found to be sufficient to correctly define the surface.

The sign of the distance is derived from the orientation of the polygon vertices, and it is in the correct determination of this sign that the subtlety in the algorithm lies. For example, Figure 3 shows a point which is closest to the edge of two triangles. There are two regions in which the distance to each of these triangles is exactly the same, but the sign is not, and in these regions the angles α and β must also be considered to determine the correct sign. Simply deciding that the point is outside if either of the triangles indicate such is not sufficient, since although this strategy would work in the case of Figure 3, it would not be correct in

[2]This criterion does not apply to binary scan conversion

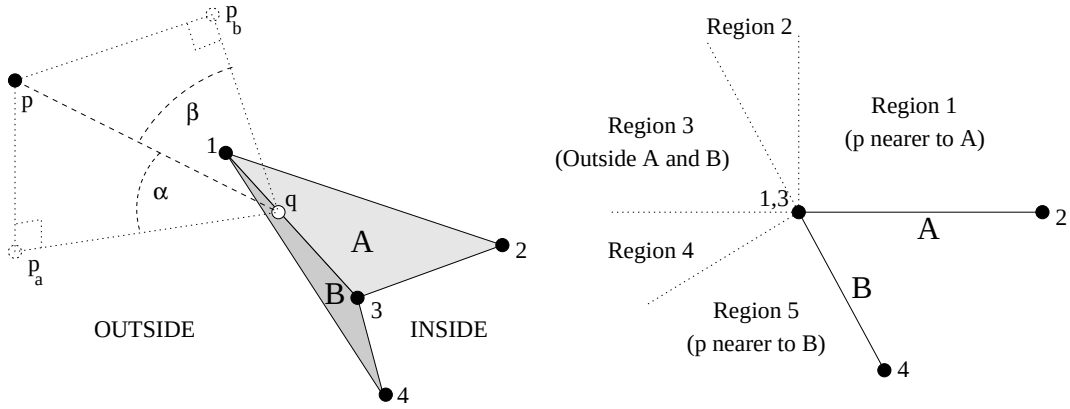


Figure 3: **Calculation of signed distance to surface.** If the nearest surface point q to a point p lies on the edge of two triangles A and B, the sign information from each triangle (i.e. whether the point p is outside or inside the surface) can be contradictory. The diagram on the right is a view of the triangles looking along the line from point 3 to 1. In regions 2 and 4 the distance to each triangle is the same, but the sign is not. In this case, the sign can be determined by considering the angles α and β which $\vec{pq}$ makes with the planes containing A and B respectively — the triangle with the largest of these angles has the correct sign.

the complementary case where the inside and outside are swapped. In order to perform the conversion in one pass through the polygons, this angle must be stored at each voxel, along with the distance. This makes the complete test for distance initialisation as follows:

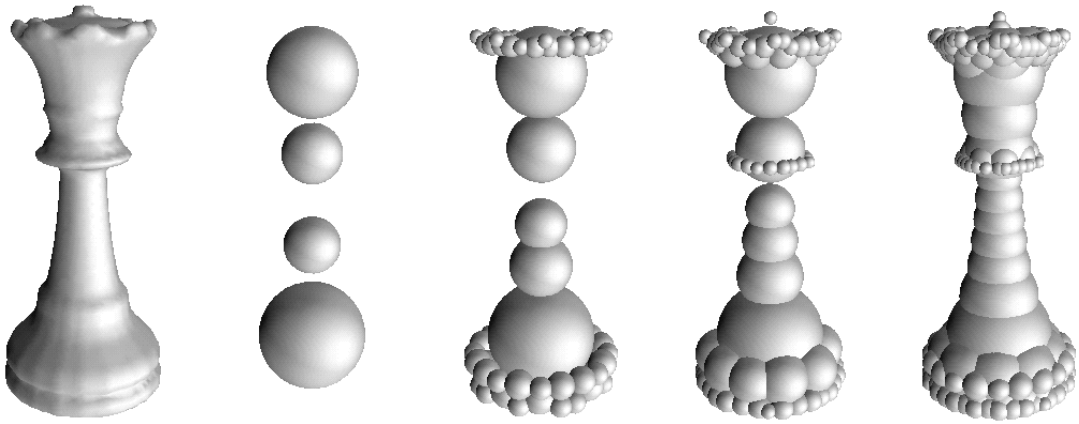
- If the voxel is not initialised, store the new distance and angle.
- Otherwise, if the absolute value of the new distance is less than the absolute value of the stored distance, store the new distance and angle.
- Otherwise, if the absolute value of the new distance is equal to the absolute value of the stored distance, set the sign of the distance to that with the smallest angle, and store this angle.
- Otherwise keep the stored distance and angle.

This is very similar to the technique used in [24], assuming that the 'line parameter' mentioned there is used in much the same way as the angle stored here.

The distance transformation used to propagate this information throughout the volume is described in detail elsewhere [3, 26]. The process described above serves as an initialisation to this algorithm. We use a 22-31-38 chamfer code (i.e. the width of one voxel is 22, the diagonal across a side is 31 and the double diagonal across the voxel is 38) to give reasonable accuracy whilst storing the distance in only two bytes.

4.1.2 Surface colour

As in [4], other surface properties may be stored throughout the volume in addition to the distance, for instance the colour. In our implementation, the colour of a voxel close to the surface is initialised from the polygon closest to that voxel, and propagated throughout the



(a) (b) (c) (d) (e)

Figure 4: **Level of detail in the sphere representation.** (a) shows the isosurface of the distance volume for the object in Figure 2, containing 8300 polygons. (b) to (e) show the results of progressively increasing the number of spheres in the representation.

volume during the distance transformation process. Voxels farther from the surface inherit the colour of the voxel from which the distance was propagated. This greatly enhances the eventual morphing sequence by allowing the gradual change of colour in addition to shape.

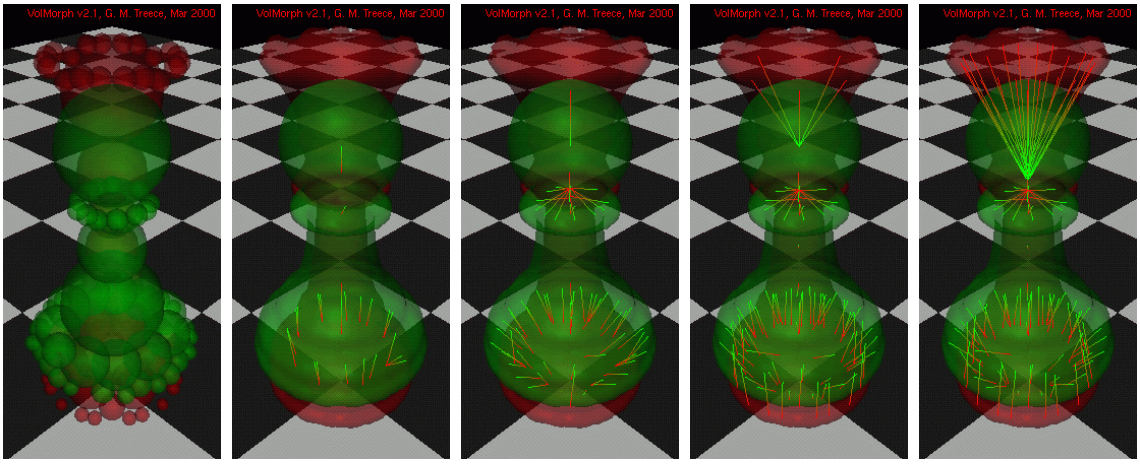
4.1.3 Distance field estimation outside the volume

The morphing process involves variation of the interpolation direction across the distance volume, and there is no guarantee that the points to be interpolated will lie within this volume. We therefore require an estimate of distance from the surface (and colour) at any point in space. This estimate does not need to be very accurate, since these points will generally have little effect on the interpolated surface, being themselves far from the original surfaces. To calculate the distance at an arbitrary point, we use a simple rule:

- If the point is contained within the volume, the distance (and colour) is tri-linearly interpolated from the surrounding points in the volume.
- If the point is outside the volume, the intersection is found of the line joining it to the centre of the volume, with the side of the volume. The returned distance is the sum of the distance *to* this intersection point (using the same distance metric as used in the distance transformation) plus the distance *at* the intersection point. The colour is inherited directly from the intersection point.

4.2 Representation by a set of spheres

Sphere extraction from the 3-D distance volume is performed in exactly the same manner as disc extraction from a 2-D distance image [26]. At each voxel, the sum of the difference of the distance value from each of the 26 neighbouring voxels is calculated. If this is positive,



(a) (b) (c) (d) (e)

Figure 5: **Effect of sphere correspondence strength.** (a) to (e) show examples of the user interface for defining sphere correspondence. In all cases the source and target objects are rendered in translucent red and green, respectively. (a) shows the sphere representation. (b) to (e) show the sphere correspondence vectors, with gradually increasing correspondence strength. The colour of each vector varies from red to green to show the direction of correspondence over time.

the voxel is at a change of gradient, and is therefore considered to be a candidate for a sphere centre. The sphere radius is simply the value of the distance field at that voxel. This set of spheres is thinned by starting with the largest, and removing all spheres within a certain fraction of that sphere's radius, then iterating.

The number of spheres (and hence the coarseness of the sphere representation) can be controlled by adjusting this fraction. Figures 4(b) to (e) show the sets of spheres extracted from the surface in Figure 4(a), whilst progressively decreasing this fraction. Since the spheres are only used to determine correspondence, and not to define the surface itself, it is only necessary to have enough spheres to distinguish the important features of the object — Figure 4(d) is the most appropriate choice in this case. User manipulation of this parameter is described in Section 5.

4.3 Calculating sphere correspondence

Once again, sphere correspondence is calculated in the same way as disc correspondence for the 2-D case [26], except that the tolerance for connecting spheres from differing objects is extended so that no regions remain unconnected. Essentially, each sphere from one object is tested against all other spheres from the other object. A correspondence vector is calculated for that sphere, which is a weighted sum of the vectors connecting its centre with each other sphere. The weight is determined by examining to what extent the distance field at the centre of the first sphere dominates that at the centre of the second, and vice versa. More

specifically, the weighting for the pair of spheres A and B, ω_{ab}, is:

$$\omega_{ab} = \begin{cases} \frac{1}{({\varepsilon_a}^2+\mu)} + \frac{1}{({\varepsilon_b}^2+\mu)} & \text{if } \varepsilon_a < x \text{ and } \varepsilon_b < x \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\varepsilon_a = \left|\vec{l}_{ab}\right| - |r_a - d_b|\,, \quad \varepsilon_b = \left|\vec{l}_{ab}\right| - |r_b - d_a| \tag{2}$$

where $\vec{l}_{ab}$ is the distance between the spheres, r_a and r_b are the radii of the spheres, and d_a and d_b are the distance values at the centre of each sphere on the *opposite* volume. The division in equation (1) is conditioned by a small constant μ (set to the square of half the width of one voxel), and x is the *correspondence strength* — it is this parameter that controls the tolerance for connecting spheres from differing objects.

Figures 5(b) to (e) show the correspondence vectors for the two sphere sets in Figure 5(a), representing a queen and a pawn. The effect of increasing correspondence strength is twofold: more correspondence vectors are defined (up to the maximum of one per sphere) and these vectors tend to connect across a larger change in shape.

4.4 Interpolation of volume data using sphere correspondence

The sphere correspondence vectors are used to guide the direction in which the volume data is interpolated. The interpolation vector for a point p is calculated from the weighted sum of all the sphere correspondence vectors. The weight ω_{pa} for each sphere A is given by:

$$\omega_{pa} = \frac{1}{({\varepsilon_p}^2 + \mu)}, \quad \varepsilon_p = \left|\vec{l}_{pa}\right| - |r_a - d_p| \tag{3}$$

where ε_p and μ have similar definitions as in equation (1), $\vec{l}_{pa}$ is the distance from p to the centre of sphere A, and d_p is the distance value at the point p, in whichever volume (source or target) contains sphere A.

Whereas ε_a and ε_b in equation (1) ensure that sphere correspondence vectors connect similar regions, ε_p in equation (3) ensures that the vectors used at a specific location are from spheres which dominate the shape at that location. The effects of both of these parameters on the 2-D case are examined in detail in [27]; essentially they tend to generate correspondence between regions which are close in both shape and distance.

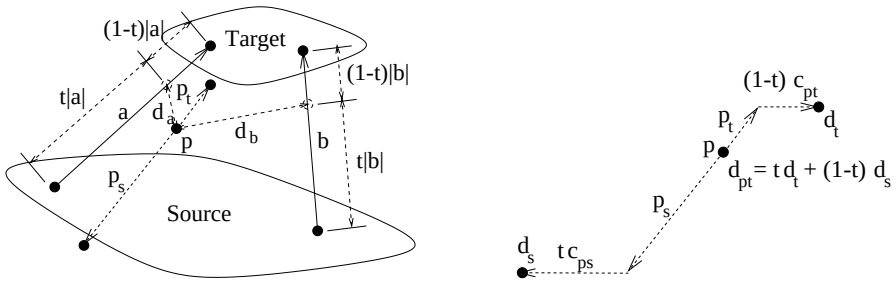
Once the point interpolation vector has been determined, the interpolated distance value at point p for frame i of the morphing sequence is given by:

$$d_{pt} = t d_t\left(\vec{p} + (1-t)\,\vec{c_p}\right) + (1-t)\, d_s\left(\vec{p} - t\vec{c_p}\right) \tag{4}$$

where the time t varies from 0 to 1, $\vec{c_p}$ is the interpolation vector at the point p, and $d_s(\vec{p})$ and $d_t(\vec{p})$ are the distance field at point p in the source and target volume respectively.

4.5 Manual guidance of automatic correspondence

Sections 4.3 and 4.4 demonstrate how correspondence between the source and target objects can be calculated automatically and used in the morphing sequence. For very complex morphs, for instance Figure 10, or to add more user control to the morphing sequence, this automatic calculation can be combined with a manual definition of correspondence. Since the sphere



(a) Manual correspondence (b) Manual and automatic

Figure 6: **Manual correspondence.** (a) Vectors $\vec{a}$ and $\vec{b}$ connect points in the source and target objects. For any point p at time t, two vectors are calculated, $\vec{p_s}$ and $\vec{p_t}$, which give the points corresponding to p for the source and target objects respectively. These points are used to interpolate the value for point p at time t (t varies from 0 to 1). The calculation of $\vec{p_s}$ and $\vec{p_t}$ is by weighted sum, based on inverse distances d_a and d_b to points along the vectors $\vec{a}$ and $\vec{b}$, which also depend on the time t. (b) These can be combined with the automatic correspondence vectors $\vec{c}_{ps}$ and $\vec{c}_{pt}$ to give d_s and d_t, from which the interpolated value d_p for point p is calculated.

correspondence ensures that low level connectivity is maintained across the objects, it is only necessary to define high level correspondence manually (e.g. which limb is connected to which limb in Figure 10).

4.5.1 Defining manual correspondence

Manual correspondence is defined by a set of vectors that connect points in the source object to points in the target object (the user interface for defining such vectors is described in Section 5). Figure 6(a) shows an example where two such vectors have been defined. These vectors are combined with a simple inverse distance scheme to give two correspondence vectors, $\vec{p_s}$ and $\vec{p_t}$, which relate the point p to corresponding points in the source and target volume respectively. These vectors are dependent on both the position of the point p being interpolated, and the time t in the morphing sequence. Rather than calculating the inverse distance to the points at the ends of each manual correspondence vector, it is calculated to a point along the vector, which moves with time from the source to the target end of the vector. This ensures that locations for which manual correspondence vectors have been defined are guaranteed to be connected at all time points in the morphing sequence.

4.5.2 Combining manual and automatic correspondence

The manual correspondence method detailed above generates two vectors ($\vec{p_s}$ and $\vec{p_t}$ in Figure 6) for each point p in each time frame. Each of these vectors give the transformed points in the source and target volumes respectively, which are interpolated to generate the data for this point. If there is no automatic correspondence, the distance is linearly interpolated from these two points, as is the case in Figure 11(b).

If only a few manual correspondence vectors have been defined, this scheme alone is not sufficient to define correspondence over the entire object, and needs to be combined with the automatic correspondence vectors. This is done at two stages in the morphing process (Figure 1): when calculating sphere correspondence and when interpolating the intermediate volumes.

In calculating sphere correspondence, rather than projecting the centre of a sphere in the source volume to the target volume, as in equation (2), the target manual correspondence vector at time 0 is used to transform the sphere centre before sampling the target volume. Conversely, when considering spheres from the target volume, the source manual correspondence vector at time 1 is used to transform the sphere centre before sampling the source volume. The resulting sphere correspondence vectors must be added to the manual correspondence vectors in order to derive the actual correspondence for each point.

When interpolating each point at each time frame, the manual correspondence vectors are first used to transform the point to the source and target volumes. These transformed points, rather than the initial point, are used to calculate two sphere correspondence vectors, $\vec{c}_{ps}$ and $\vec{c}_{pt}$, for the source and target volume respectively. The manual correspondence vectors $\vec{p_s}$ and $\vec{p_t}$ are also used in the interpolation of each point, so equation (4) is replaced by:

$$d_{pi} = t d_t \left(\vec{p} + \vec{p_t} + (1-t)\,\vec{c}_{pt}\right) + (1-t)\, d_s \left(\vec{p} + \vec{p_s} - t\vec{c}_{ps}\right) \tag{5}$$

as shown in Figure 6(b). In effect, both the source and target volumes are warped by the manual correspondence vectors before either the sphere correspondence or interpolation is performed.

4.6 Isosurface extraction

In order to display the interpolated surfaces in real time on readily available hardware, a triangular mesh is generated from the zero isosurface of the interpolated volumes. The generation of this mesh has as much effect on the quality of the rendering as the conversion to volume based representation discussed in Section 4.1. Although Marching Cubes [18] is the best known of these algorithms, it does not in general produce high quality meshes, and we use Regularised Marching Tetrahedra [25] instead.

5 User interface

The user is presented with one window, but three 'states', which can be toggled as in Figure 7. Manual correspondence can be defined interactively in the first state, following which the automatic correspondence parameters can be adjusted in the second. Finally, the morphing sequence itself can be viewed in the third state.

5.1 State one — creation of manual correspondence vectors

Manual correspondence vectors define a 3-D correspondence between points in the source and target objects. However, both the display and the input device can only represent 2-D information. A technique for defining such vectors in 2-D is presented in [8] whereby the user first selects a point on the surface of an object (or on a pre-defined grid), following which the *direction* of a unit vector from that point is chosen, and finally the magnitude of this vector.

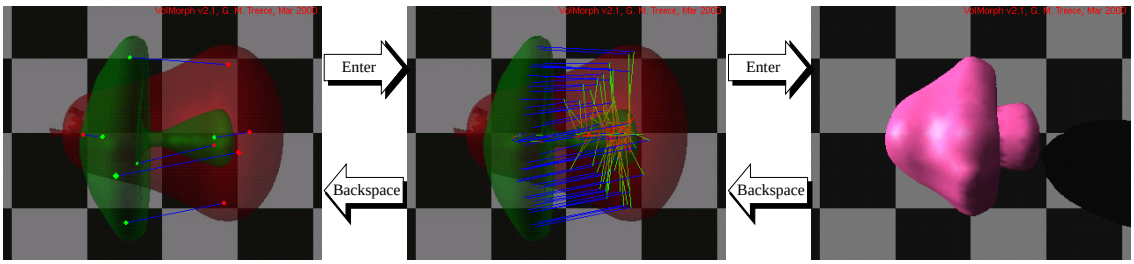


Figure 7: **Progression of the user interface.** The user is presented with a sequence of three displays — manual correspondence definition (on the left), sphere correspondence adjustment (centre) and the morph itself (right). This sequence can be traversed by using the 'Enter' and 'Backspace' keys. The intermediate surfaces for the morph are calculated on starting the final display.

There are two difficulties with this approach; firstly we would like to define points which are within (rather than on the surface of) the objects, and secondly we want an interface which is more natural for the user. Most 2-D drawing programs, for instance, use a click-and-drag approach to define vectors. We can mimic this behaviour by using the objects themselves to provide the extra dimension which is missing from the input device. The procedure is as follows:

- Both objects are displayed simultaneously in transparent red (for the source object) and green (for the target), as in the left hand diagram of Figure 7.
- Clicking and dragging *outside* the objects changes the viewpoint and can be used to examine the objects.
- Once the objects are at an appropriate orientation where corresponding parts can be clearly seen, and are not obscured (either in front or behind), the user clicks on the source (red) object.
- A correspondence point is defined on this object, using the median distance between the surfaces underneath the selected point as the z-dimension.
- As the user drags the mouse, a vector is drawn from this point to a point on the target (green) object, providing that the mouse is over part of the target object.
- On releasing the mouse, the z-dimension of the corresponding point is similarly defined as the median of the z-dimensions of the surface under the selected point.
- The spatial location of this vector can now be examined by clicking and dragging outside of both objects, or deleted by right-clicking on the vector itself.

This interface provides a very fast way of defining corresponding points. These points are drawn in the same colour as the object they are defined for, with the connecting vectors in blue (as in the left diagram of Figure 7), such that it is easy to see which parts of which object are connected. In addition, the way in which the vectors are defined ensures that corresponding points are always defined near the centres of the objects that they connect.

The one exception to this is if the user clicks over a part of an object which has further parts behind it, in which case the median distance over *all* these surfaces is used, which may lie outside any one of the surfaces. This situation can be clearly seen due to the transparency of the objects, and if such vectors are defined, changing the viewpoint makes them apparent.

5.2 State two — selection of automatic correspondence parameters

There are two automatic correspondence parameters which can be adjusted to vary the eventual morphing sequence — the number of spheres and correspondence strength (see Section 4.3). Changing the former will cause both the sphere representation and the sphere correspondence to be updated, whilst the latter only affects sphere correspondence. In all but the most complex cases, both of these can be recalculated in real time, providing interactive control of both parameters, as in the middle diagram of Figure 7.

Both surfaces are once again rendered transparent red or green, but can be displayed either as the triangulated isosurface of the source and target volumes, or as their sphere representations, as in Figure 5(a). The correspondence vectors themselves are each displayed as lines gradually changing from red to green to indicate their sense, as in Figures 5(b) to (e). If manual correspondence vectors have also been defined, the part of each sphere correspondence vector due to the manual correspondence is shown in blue, and the remaining part in red to green, as in Figure 7(b). Both parameters can be changed using key presses, whereupon both the number and directions of the vectors are updated.

5.3 State three — display of morphing sequence

The interpolation of the morphing sequence is the most time consuming step in the process; the time taken can vary from a few seconds to a few hours, depending on the complexity of the correspondence, the isosurface resolution and the number of frames (or intermediate surfaces). This interpolation is performed on entering the third state of the user interface. Although the size of the volume is fixed at this point, the resolution of the isosurface compared to the volume (i.e. the approximate size of each of the triangles) is variable. Low resolution morphs can be calculated very quickly and used to assess the approximate behaviour before a high resolution morph is calculated.

The user can vary the scene and model position in addition to the lighting and speed of the morph, during the sequence. This enables sufficient interaction with the morph to determine if it is what the user wanted. If not, then the user can return to the previous states to adjust either the automatic or manual correspondence, then recalculate the morph once this has been done.

6 Metamorphosis examples

The algorithms described in Section 4 and user interface described in Section 5 have been implemented using OpenGL[3] in VolMorph[4], which is freely available for Irix, Linux and Windows platforms. The polygonal models used in the following sections can be downloaded

[3] `http://www.opengl.org`, Silicon Graphics Inc.

[4] `http://svr-www.eng.cam.ac.uk/~gmt11/software/volmorph/volmorph.html`

	Pawn ⇒ Queen	**Tricycle ⇒ Sphere**	**Dragon ⇒ Creature**	**Pear ⇒ Mushroom**
Source model polygons	2,500	40,000	110,000	900
Target model polygons	4,000	8,000	75,000	240
Dimensions of volume	$30 \times 30 \times 55$	$112 \times 105 \times 108$	$85 \times 136 \times 143$	$35 \times 38 \times 38$
Size of volume	0.19Mb	4.84Mb	6.31Mb	0.19Mb
Scan conversion time	00:00:03	00:00:35	00:01:06	00:00:01
User interaction time	00:00:30	00:01:00	00:30:00	00:00:30
Frames	20	20	20	20
Polygons per frame	$\approx 4,000$	$\approx 38,000$	$\approx 33,000$	$\approx 3,300$
Morph creation time	00:01:31	00:45:20	01:31:00	00:00:33

Table 1: **Morphing details.** The number of polygons, the size of the discrete volume and the times taken in conversion, user interaction and creating the morph are shown for the sequences in Figures 8, 9, 10 and 11. Times are given in hours:minutes:seconds, measured on a Silicon Graphics Indigo 2 R10000 workstation.

from the Geomview[5] distribution (pear and mushroom) or from 3D Cafe[6] (all other models).

Table 1 contains details of each of the following examples. Source and target volumes are stored as four bytes per voxel: two for the distance value and two for the colour. All examples are for twenty frame morphs (i.e. eighteen interpolated models), and the morph creation time given in the table is the time to create all of the intermediate models (including isosurface extraction). In each example, the new scheme is compared to a simple morph generated by linear interpolation of the distance values, with no calculation of region correspondence, which is equivalent to shape based interpolation [19, 21] of the objects.

6.1 Pawn to queen

Figure 8 shows a morph from a white pawn chess piece to a black queen chess piece. Shape based interpolation in Figure 8(a) results in a variety of unnatural effects. The colour of the green base bleeds up into the lower section of the intermediate shape, giving it a green tint. The crown of the queen unfolds from the top of the pawn head rather than growing from the side. Also, the collar from *both* models is apparent in the intermediate morphs, rather than a single collar in the average position.

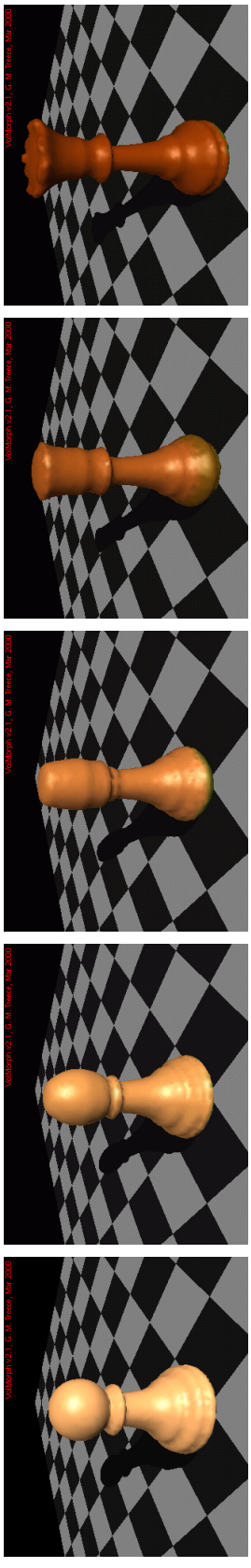
All these problems can be corrected by using automatic correspondence. The user interaction time for this morph in Table 1 is for adjusting the correspondence strength alone, and the whole process (from initial models to display of a twenty frame morph) is completed in only two minutes. Figure 8(b) shows the results — both the shape and colour change are now much more natural.

6.2 Tricycle to sphere

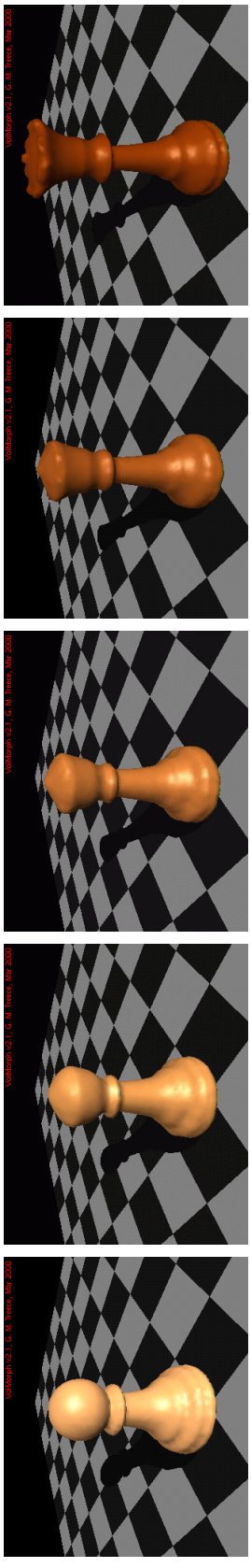
Figure 9 shows a morph from a coloured tricycle to a white sphere. The shape and topology of the objects are very different, and the simple strategy in Figure 9(a) is not able to cope

[5] `http://www.geom.umn.edu/software/geomview/`.

[6] `http://www.3dcafe.com`



(a) Shape based metamorphosis



(b) Shape based metamorphosis with region correspondence

Figure 8: **Pawn to queen metamorphosis.**

(a) Shape based metamorphosis

(b) Shape based metamorphosis with region correspondence

Figure 9: **Tricycle to sphere metamorphosis.** The objects and base rotate during the sequence, the light being fixed relative to the viewpoint.

with this — for instance the rear wheels and the handlebars disappear completely during the morph. In contrast, the morph of Figure 9(b) shows a gradual change of *all* parts of the tricycle to the sphere. The effect is as if the tricycle is gradually inflated; the components of the tricycle are visible at all stages of the morph. Interpolating colour as well as shape adds to this effect by giving visual clues as to which parts of the sphere correspond to which parts of the tricycle.

In this case, the user interaction time in Table 1 was for adjusting the number of spheres needed to represent the tricycle. In particular, both box sections (e.g. the back foot rest) and cylindrical sections (e.g. the handlebars) require a lower than usual density of spheres to determine good correspondence.

6.3 Dragon to creature

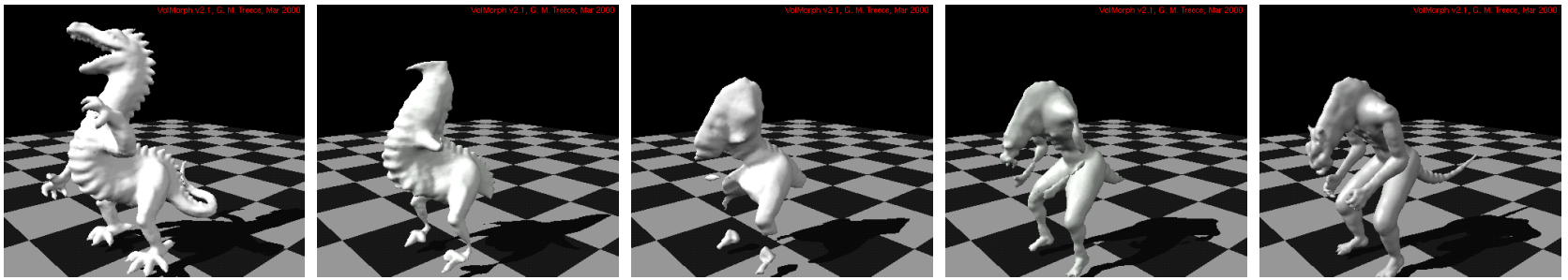
Figure 10 shows a shape-only morph from a dragon to an alien creature (neither of the models are coloured). Although both objects have the same number of limbs and general shape, there is considerable variation in both the detail and the poise. In fact, neither the limbs, tail nor the head are in the same position — as can be seen from the intermediate frame of Figure 10(a), where all the limbs have disappeared. In this case, automatic correspondence is not sufficient to define the morph, since both the left arm and the head are farther away from each other than from other body parts (the left arm, for instance, would otherwise be joined to the left thigh).

Three manual correspondence vectors were defined for each limb, placed at the major joints, plus three each for the tail, head and torso. This results in the morph of Figure 10(b) — the limbs are now connected where the manual correspondence has been defined, however they still disappear in-between these points. There are resulting gaps in the surface at the tail and right hand, as well as a lack of definition at the extremities. The addition of automatic correspondence in Figure 10(c) corrects most of these problems, however the large movement of the left arm in contrast with the relatively stationary left leg and torso leads to a lack of definition in the left hand which is difficult to overcome.

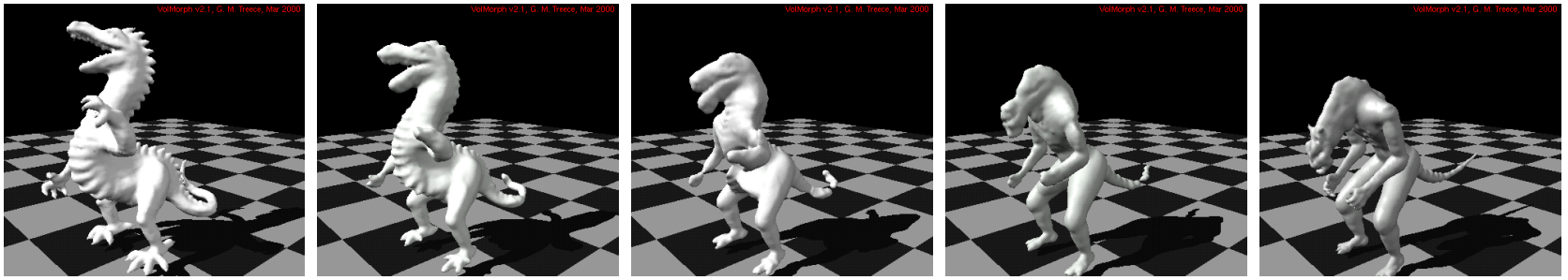
Since this morph is more complex, the user interaction time in Table 1 is greater; this time also includes the creation of low resolution morphs to check the effect of manual correspondence. The models themselves are also more complex, leading to a larger volume and hence a longer processing time. However, the entire processing was still performed within two and a half hours.

6.4 Pear to mushroom

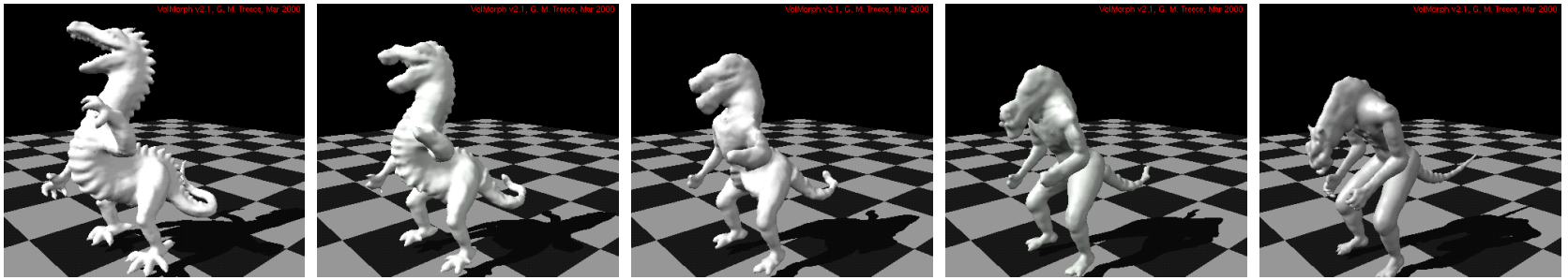
Figure 11 shows two possible morphs from a pear to a purple mushroom, demonstrating that there are many ‘correct’ solutions to the problem. Figure 11(a) is the result of using simple interpolation (the addition of region correspondence in this case generates the same result). Here the effect is as if the base of the pear has been squashed, resulting in the expansion of the top. On the other hand, Figure 11(b) was generated by defining a small set of manual correspondences, as in Figure 7. Now the effect is as if the top of the pear has been pushed downwards through the base. Either of these morphs could be the most appropriate in different circumstances.



(a) Shape based metamorphosis

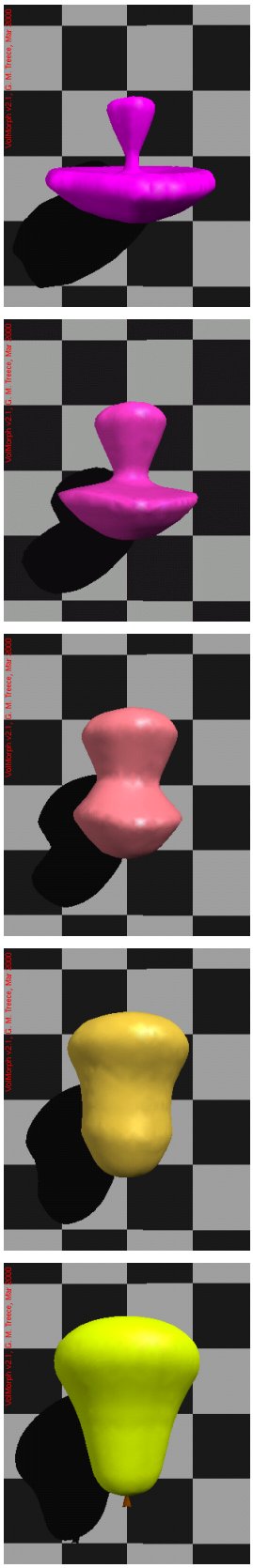


(b) Shape based metamorphosis with manual guidance

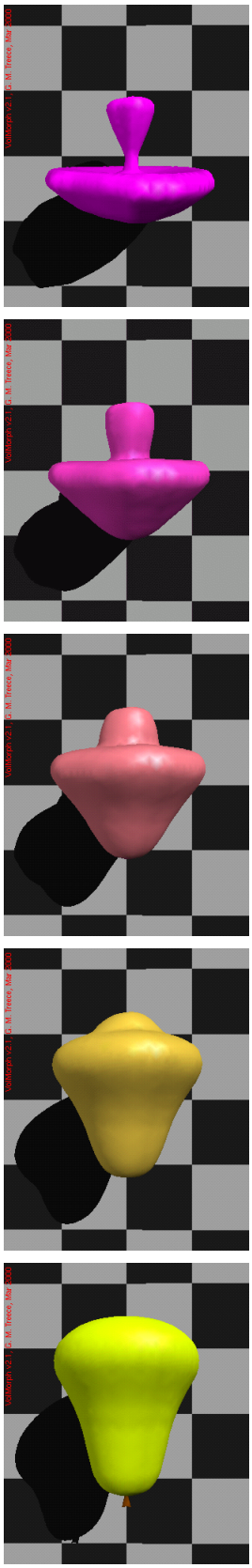


(c) Shape based metamorphosis with manual guidance and region correspondence

Figure 10: **Dragon to creature metamorphosis.** The manual guidance consisted of three correspondence vectors per limb, and a further three vectors for each of the tail, torso and head.



(a) Shape based metamorphosis



(b) Shape based metamorphosis with manual guidance

Figure 11: **Pear to mushroom metamorphosis.** The manual guidance for (b) is given in Figure 7.

7 Conclusions

It is important when morphing between two objects to maintain a natural level of user control over the effect of the morph, whilst automatically preventing morphs which are clearly incorrect. The use of region correspondence, calculated automatically from a sphere representation of the objects, automates the low level detail of the morph whilst leaving room for larger scale manual changes. In addition, the speed and versatility of this method in handling complex shape changes enable the user to easily interact with the morphing process — changing parameters has an immediate effect on the display of correspondence, and low resolution morphs can be calculated in minutes to assess the eventual effect.

Much of the processing speed of the algorithm is achieved by high quality scan conversion of polygonal models to a volumetric representation, and equally high quality isosurfacing to recreate the polygonal models for display. This can reduce the size of the volume for a given surface quality by more than an order of magnitude. The automation of low level detail allows a simple manual correspondence technique to be adopted, where high level correspondence vectors are defined by clicking and dragging.

The speed and flexibility of the method is demonstrated on a wide variety of examples — in some cases the entire process from initial models to real time morphing took only a few minutes.

Acknowledgements

Graham Treece is supported by an EPSRC studentship, and a Newton Trust award from the University of Cambridge Department of Engineering.

References

[1] M. Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.

[2] J. Bloomenthal and C. Lim. Skeletal methods of shape manipulation. In *Shape Modeling International '99*, International Conference on Shape Modeling and Applications, Aizu-Wakamatsu, Japan, 1999.

[3] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–345, 1984.

[4] D. E. Breen and S. Mauch. Generating shaded offset surfaces with distance, closest-point and color volumes. In *Proceedings of International Workshop on Volume Graphics*, pages 307–320, March 1999.

[5] M. Chen, M. W. Jones, and P. Townsend. Volume distortion and morphing using disk fields. *Computers and Graphics*, 20(4):567–575, 1996.

[6] D. Cohen-Or, D. Levin, and A. Solomovici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998.

[7] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of 1998 IEEE Symposium on Volume Visualization*, pages 23–30, October 1998.

[8] J. Gomas, L. Darsa, B. Costa, and L. Velho. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers, Inc., 1999.

[9] A. Gregory, A State, M. C. Lin, D. Manocha, and M. A. Livingston. Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 15(9):453–470, 1999.

[10] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of Visualization '94*, pages 85–92. IEEE Computer Society Press, 1994.

[11] J. F. Hughes. Scheduled fourier volume morphing. In *SIGGRAPH '92*, volume 26 of *Computer Graphics*, pages 43–46, 1992.

[12] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[13] A. Kaufman. Efficient algorithms for scan-converting 3D polygons. *Computers and Graphics*, 12(2):213–219, 1988.

[14] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, pages 51–64, July 1993.

[15] F. Lazarus and A. Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, 14:373–389, 1998.

[16] A. W. F. Lee, D. Dobkin, W. Sweldens, and P. Schröder. Multiresolution mesh morphing. In *Proceedings of SIGGRAPH 99*, pages 343–350. ACM SIGGRAPH, 1999.

[17] A. Lerios, C. D. Garfinkle, and M. Levoy. Feature-based volume metamorphosis. In *SIGGRAPH '95*, volume 29 of *Computer Graphics*, pages 449–464, 1995.

[18] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–168, July 1987.

[19] B. A. Payne and A. W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, pages 65–71, January 1992.

[20] V. Ranjan and A. Fournier. Shape transformations using union of spheres. Technical Report TR-95-30, Imager Computer Graphics Laboratory, Department of Computer Science, University of British Columbia, 1995.

[21] S. P. Raya and J. K. Udupa. Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging*, 9(1):32–42, 1990.

[22] D. Ruprecht and H. Müller. Deformed cross-dissolves for image interpolation in scientific visualization. *The Journal of Visualization and Computer Animation*, 5:167–181, 1994.

[23] W.-S. V. Shih, W.-C. Lin, and C.-T. Chen. Morphologic field morphing: Contour model-guided image interpolation. *International Journal of Imaging Systems and Technology*, 8:480–490, 1997.

[24] M. Sramek and W. E. Kaufman. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):251–267, July 1999.

[25] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*, 23(4):583–598, 1999.

[26] G. M. Treece, R. W. Prager, A. H. Gee, and L. Berman. Fast surface and volume estimation from non-parallel cross-sections, for freehand 3-D ultrasound. *Medical Image Analysis*, 3(2):141–173, 1999.

[27] G. M. Treece, R. W. Prager, A. H. Gee, and L. Berman. Surface interpolation from sparse cross-sections using region correspondence. Technical Report CUED/F-INFENG/TR 342, Cambridge University Engineering Dept, March 1999.

[28] R. T. Whitaker and D. E. Breen. Level-set models for the deformation of solid objects. In *Proceedings of 3rd International Workshop on Implicit Surfaces*, pages 19–35. Eurographics, June 1998.

[29] G. Wolberg. Image morphing: a survey. *The Visual Computer*, 14:360–372, 1998.