

Morphing the BlobTree

Eric Galin, Antoine Leclercq
L.I.G.I.M
Université Claude Bernard Lyon 1
69622 Villeurbanne Cedex
[egalin-leclercq]@ligim.univ-lyon1.fr

Samir Akkouche
L.I.G.I.M
Ecole Centrale de Lyon
B.P. 163, 69131 Ecully Cedex
samir@ec-lyon.fr

Abstract

Implicit surfaces have proved to be a particularly well suited and efficient model for animating and morphing shapes of arbitrary topologies. The BlobTree model is characterized as a hierarchical combination of skeletal primitives organized in a tree. The nodes hold blending, boolean and warping operators, which allows the design of complex objects.

In this paper, we address the metamorphosis of the BlobTree. This appears a difficult task as the tree data-structures of the initial and final shapes are completely different in the general case, and consequently cannot be matched easily. We propose an original technique that solves the correspondence process and creates an intermediate generic BlobTree model whose instances interpolate the initial and final shapes.

The animator may control the correspondence between features and can specify both the speed of transformation and the trajectory of the nodes and the leaves of the generic BlobTree model. This provides the end user with a tight control over the transformation so as to achieve good visual effects.

Keywords: animation, BlobTree, implicit surfaces, metamorphosis, warping

1. Introduction

Metamorphosis (or morphing) can be defined as the process of smoothly transforming an initial shape into a final shape. Metamorphosis has a vast variety of applications. It has been successfully applied to some modeling systems, and is extensively used for generating special effects in the movie industry.

The metamorphosis between two shapes may be defined as the continuous evolution of a time varying shape interpolating an initial and a final shape. Given two shapes, there are an infinity of transformations that create the metamorphosis sequence. Therefore, the visual aspect of the transformation is often the only relevant criterion to evaluate the quality of the transformation in a key-frame animation system.

It seems essential that the metamorphosis should avoid unnecessary shape distortion or changes of the topology. In general, the transformation should be smooth and continuous. Shape coherence should be maintained whenever possible so as to preserve the characteristic features of the

source and target shapes. Amorphous (*i.e.* featureless) transitions that cannot be avoided in complex morphing sequences should be limited in time. Although some techniques focus on automating the metamorphosis process (which may be effective for restricted class of shapes or models), user control proves to be fundamental to produce convincing animations.

1.1. Previous work

Lazarus¹⁹ has presented an interesting survey of existing metamorphosis techniques. The proposed classification exhibits two major categories according to the underlying shape model : techniques focusing on mesh models and volumetric methods. Lazarus further separates volumetric methods into two sub-categories : voxel based approaches and implicit surface techniques.

In the following overview, we will address those last two methods separately as the control over the transformation is achieved in a completely different fashion.

1.1.1. Boundary representation approaches

Methods that directly work on the boundary representation of polygonal meshes need to solve the *vertex correspon-*

dence problem. Although many approaches have been proposed, no general solution has been presented so far. Kent¹⁶ first proposed to merge the topologies (*i.e.* the adjacency graph) of star shaped polyhedral objects. Lazarus¹⁸ later generalized that technique to a wider class of objects. Kaul¹² has proposed to tackle the vertex correspondence problem by using the weighted Minkowski sum of argument polyhedra which works well for convex objects.

Kanai^{14, 15} gives an alternative method based on a user decomposition of the meshed models into patches that can be paired and morphed. Even though a good control can be achieved, the decomposition process can become slow and tedious for large meshes. Recently, Lee²⁰ has presented a multi-resolution mesh morphing approach that overcomes this problem.

The very limitation of boundary representation based morphing methods is that the source and target models should share the same topology (*i.e.* have the same genus). Moreover, the computation of intermediate shapes becomes the more expensive as the number of vertices increases, and the control of the transformation may be tedious for complex models.

1.1.2. Voxel based methods

Unlike mesh based techniques, volumetric methods can handle the metamorphosis of shapes of different topologies easily. Another advantage of the volumetric approach is that almost any model may be converted into a voxel.

Lerios²¹ first extended the two-dimensional morphing algorithm proposed by Beier³, warping and cross-dissolving matched density volumes. Hughes¹¹ and He¹⁰ proposed a method based on the Fourier and on a wavelet decomposition of space respectively. Voxel based approaches are memory consuming and computationally demanding however. Good visual results cannot be obtained unless using a fine sampling of the objects. The computation cost becomes the more prohibitive as the size of the sampling grid increases.

1.1.3. Implicit surface techniques

In contrast to voxel based methods, implicit surface techniques avoid the computation of a fixed sized sampling grid and directly compute the metamorphosis by interpolating the parameters of the field functions representing the objects.

Pasko²² has proposed a general morphing scheme, using a linear field function interpolation to define the time varying field functions characterizing the transformation. Wyvill⁵ has proposed an original technique for morphing implicit surfaces built from point skeletal elements, also known as *blobs* or *soft objects*. A pre-processing correspondence step, namely cellular matching and hierarchical matching, ensures all the elements of the initial and final shapes have been paired, possibly creating null components whenever necessary. The time varying blob is defined by interpolating the

parameters characterizing the paired primitives. However the visual aspect of the morphing sequences are often poor.

We have generalized and extended Wyvill's morphing technique to implicit surfaces built from convex skeletons of any dimension⁷. Our method provides a good control over the transformation. The animator may freely pair sets of primitives of the source and target models, and even control the trajectory⁸ of intermediate primitives to avoid amorphous or featureless transitions.

1.2. Contributions

Recently, Wyvill has presented the BlobTree model²⁸ that can be thought of as an extension of blobs. The BlobTree is characterized by a hierarchical combination of skeletal primitives organized in a tree data-structure whose nodes hold blending, boolean operators and warping operators. The key feature of the BlobTree is that complex models can be built with a small number of skeletal primitives using arbitrary combinations of blending, warping and boolean operations.

In this paper, we address the metamorphosis of the BlobTree model. We rely on the fundamental concepts of the existing blob morphing technique and extend our previous work to the more general BlobTree model. Specifically, we make the following contributions :

- We put the emphasis on the creation of a generic BlobTree model that characterizes the whole transformation. For instance, genericity enables us to save interesting intermediate shapes for later use, and combine interpolations so as to define a Bézier-like metamorphosis where control knots are replaced by control BlobTree shapes.
- We provide both low and high level tools to achieve coarse and fine control over the transformation. The animator may control the correspondence between features of the BlobTrees and may specify both the speed of the transformation and the trajectory of the nodes and the leaves of the generic BlobTree model.

The remainder of this paper is organized as follows. To make it self-contained, we recall the fundamentals of the BlobTree model in section 2. In section 3, we present our blob morphing technique and put the emphasis on the transformation of the skeletal elements. Section 4 addresses the metamorphosis of the BlobTree: we show how to adapt the existing blob morphing technique to the tree structure of the BlobTree model. Eventually, we present some morphing sequences in section 5.

2. The BlobTree

Let us recall that implicit surfaces built from skeletal elements, also known as *blobs* or *soft objects*²⁶, are characterized by a scalar field $f(x, y, z)$ generated by summing the

influences of n scalar field elements $f_i(x, y, z)$.

$$f(x, y, z) = \sum_{i=1}^{i=n} f_i(x, y, z)$$

The surface of the object may be characterized from this global potential field $f(x, y, z)$ as the points of space whose potential equals a threshold value denoted as T .

$$\Sigma = \left\{ M(x, y, z) \in \mathbb{R}^3, f(x, y, z) = T \right\}$$

The potential field contributions f_i are decreasing functions of the distance to a skeleton $f_i = g_i \circ d_i$ where $g_i : \mathbb{R}_+ \rightarrow \mathbb{R}$ is the field function, and $d_i : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ refers to distance to the skeleton. Thus an element i will be fully characterized by its skeleton S_i , its distance function $d_i(x, y, z)$ and its field function $g_i(r)$, and denoted as :

$$\{S_i, d_i(x, y, z), g_i(r)\}$$

Both the skeleton S_i and the distance function d_i characterize the shape of the element, whereas the field function g_i defines the way elements blend together. In our implementation, we restricted skeletons to polytopes (*i.e.* convex polygonal shapes of any dimension) for some reasons that will be discussed in the next section. Throughout this paper, skeletal elements will refer to points, line segments, convex polygons and convex polyhedra.

Several field functions have been proposed in the literature. An overview by Bloomenthal et al.⁵ describes great many possible field functions. Although our morphing technique may cope with any kind of field function, our own implementation relies on the piecewise rational polynomial proposed by Blanc⁴.

$$\begin{cases} g(r) = 1 - \frac{9r^4}{k + \left(\frac{9}{2} - 4k\right)r^2} & 0 \leq r^2 \leq \frac{1}{4} \\ g(r) = \frac{(1-r^2)^2}{\frac{3}{4} - k + \left(\frac{3}{2} + 4k\right)r^2} & \frac{1}{4} \leq r^2 \leq 1 \end{cases}$$

This function is both computationally efficient and its shape is controlled by a restricted set of parameters that are the maximum *intensity* I , the *radius of influence* R and possibly a *stiffness* coefficient k .

Although any kind of distance function might qualify, we also restricted our study to the distance derived from the L^p metrics⁹, which provide an extension of the usual Euclidian distance.

The BlobTree model²⁸ can be thought of as an extension of those *soft objects*. Unlike blobs that are characterized by

a mere set of elements that blend in the same way[†], the BlobTree is characterized by a hierarchical combination of primitives organized in a tree data-structure. The nodes of the tree hold blending, boolean operators and warping operators, whereas the leaves are characterized as skeletal elements (figure 1).

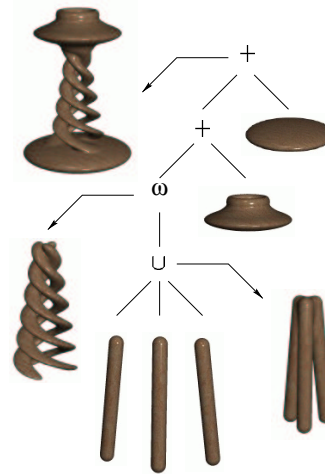


Figure 1: Candlestick created by blending the twisted union of three cylinders with tapered spheres

The evaluation of the field function in space is achieved by recursively traversing the BlobTree, either evaluating the field functions at the leaves of the tree or combining the field function values returned by the children of a given node.

The blending operator is defined by summing the field functions of the contributing elements :

$$f_+(x, y, z) = \sum_{i=1}^{i=n} f_i(x, y, z)$$

Therefore, blobs form a restricted sub-class of the BlobTree model. As suggested by Wyvill et al.²⁸, boolean operators may be defined by using Pasko's set theoretic functions²². In our own implementation however, we use the minimum and the maximum instead.

$$f_U(x, y, z) = \max_{i \in [1, n]} f_i(x, y, z)$$

$$f_{\cap}(x, y, z) = \min_{i \in [1, n]} f_i(x, y, z)$$

The BlobTree includes warping operators at its nodes that distort the shape of the implicit surface by warping space in its neighborhood. Generally speaking, a warp is a continuous

[†] A restricted class of *blobs* rely on a graph that hold blending relationships to avoid unwanted blending between neighboring elements

function $w(x, y, z)$ that maps \mathbb{R}^3 into \mathbb{R}^3 . The field function in space is defined as :

$$f_{\omega}(x, y, z) = f \circ \omega^{-1}(x, y, z)$$

Although affine transformations can be thought of as special cases of warp operators, they have been coded separately so that consecutive affine transformations may be concatenated out of efficiency. In our own implementation, warps have been chosen from the Barr operators¹ and affine geometric transformations.

As it is the case in constructive solid geometry, different BlobTree models may result in the same implicit surface. Therefore, two models will be said to be *equivalent* if and only if they characterize the same implicit surface. Two BlobTrees will *overlap* if their tree structures overlap, *i.e.* if the leaves and nodes at the same levels can be bijectively matched.

3. Blob metamorphosis

In this section, we recall the principles of the blob metamorphosis technique^{7,8}. The metamorphosis between two shapes relies on the definition of a graph of correspondence matching elements of the initial and the final models. This graph specifies which parts of the models should undergo metamorphosis, and may be either provided by the animator or created automatically through some matching heuristics.

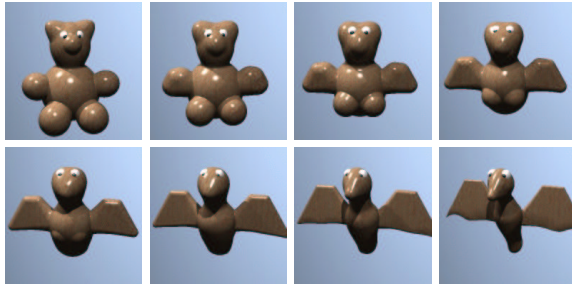


Figure 2: Metamorphosis between a bear and a bird

To metamorphose between an initial and final model, we define a correspondence between the elements representing the two models. As this initial correspondence is in general not a bijection, we define a new blob for each of the two models. Those new blobs have more nodes, and the correspondence can be made bijective on this set of nodes.

An intermediate *generic* sub-component is associated to each link in the bijective correspondence. The whole transformation itself is characterized by a *generic* model whose instantiations interpolate the initial and the final shapes throughout time.

The time varying skeletons of the generic elements are defined as the linear interpolation based on Minkowski

sum of the skeletons of the initial and final corresponding sub-components. As the direct interpolation of the distance and the potential functions generates complex equations, we tackle the transformation problem by using *classes* of parametrized functions. We characterize time varying distance and potential functions as specific members of those classes with interpolated parameters.

Thus, we achieve a concise *generic representation* that preserves shape coherence during the transformation. Moreover, the genericity property enables us to combine several interpolations together so as to define a Bézier-like metamorphosis where control knots are replaced by control blobs⁷. The different steps of the morphing process are further detailed in the next paragraphs.

3.1. The correspondence process

The creation of a graph of correspondence matching elements of the initial and final shape models is a key step in the morphing process as it constrains the animation. Several heuristics have been proposed, a good overview has been proposed by Wyvill⁵.

Techniques may be split into two categories : techniques matching elements according to their position in space (cellular matching), and techniques that require extra information about the elements (hierarchical matching). A pre-processing step ensures that both initial and final shapes share the same number of components by creating phantom elements whenever necessary. The characterization of phantom elements will be addressed in section 3.3.

Unfortunately, those heuristics often lead to a loss of shape coherence during the animation. For instance, the metamorphosis between a man and a rabbit of equal size, *i.e.* a transformation between two shapes of similar geometry and equal topology, yields intermediate shapes with disconnected components⁵.

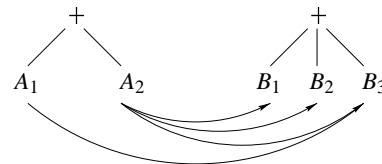


Figure 3: Graph of correspondence matching elements of the initial and final blobs

As suggested in Galin et al.⁷, the graph of correspondence may be generated automatically, simply by matching all the elements of the source and target models. However, this straightforward approach generates $n_A n_B$ elements in the time varying model, which often results in a loss of shape coherence during the transformation. In our implementation,

we prefer to let the animator define the graph of correspondence, with a view to tuning the transformation as needed (figure 3).

There are no restrictions over this graph of correspondence that characterizes which elements really undergo metamorphosis. As we will see in the next section, this is no longer the case for the transformation of the BlobTree.

3.2. The decomposition step

Since the graph of correspondence is not bijective in the general case, we split each multiply paired component into sub-components with a view to creating a new graph bijectively matching those sub-components.

Given a rough graph of correspondence matching parts of the initial and the final shapes A and B , *i.e.* sets of components A_i and B_j , we split the components of those shapes into sub-components $s_k(A_i)$ and $s_k(B_j)$ with a view to creating a new graph bijectively matching those sub-components (figure 4). Initial and final components that are not matched involve the creation of specific phantom elements ⁷.

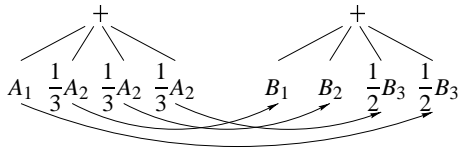


Figure 4: Creating a bijectively matching graph through component splitting

We have proposed to keep both the skeletons and the distance functions unchanged and to split the field functions into sub-functions. Therefore, the skeletons $S_{s_k(A_i)}$ and $S_{s_k(B_j)}$ are the same as S_{A_i} and S_{B_j} . The distance functions $d_{s_k(A_i)}$ and $d_{s_k(B_j)}$ are also the same as d_{A_i} and d_{B_j} respectively, whereas the potential functions $f_{s_k(A_i)}(r)$ and $f_{s_k(B_j)}(r)$ are weighted replications of $f_{A_i}(r)$ and $f_{B_j}(r)$, which preserves shape :

$$\begin{cases} f_{s_k(A_i)}(r) = \alpha_{ik} f_{A_i}(r) \\ f_{s_k(B_j)}(r) = \beta_{kj} f_{B_j}(r) \end{cases}$$

The weights α_{ik} and β_{kj} should form a partition of unity, formally :

$$\sum_{k=1}^{n_{A_i}} \alpha_{ik} = 1 \quad \sum_{k=1}^{n_{B_j}} \beta_{kj} = 1$$

Several heuristics for computing those weights α_{ik} and β_{kj} have been proposed⁷. A straight forward technique that works well in practice consists in setting weights as follows : $\alpha_{ik} = 1/n_{A_i}$ and $\beta_{kj} = 1/n_{B_j}$ where n_{A_i} and n_{B_j} refer to the

number of components linked with the initial and final components A_i and B_j respectively. The animations presented in this paper (figures 2, 11, 8) have been designed with such a scheme.

3.3. Phantom elements

In our own implementation, a phantom element is defined as a primitive with a null field contribution, *i.e.* with a field function characterized by null maximum intensity. Therefore, phantom elements can be handled as normal primitives by the morphing algorithm. The choice of the other parameters, such as the skeleton, the radius of influence and the stiffness coefficient, have a strong visual effect on the transformation.

In our implementation, the animator may set those parameters to control the animation. By default, a phantom element B_j associated to another element A_i is characterized by a null maximum intensity $I_{B_j} = 0$ whereas the skeleton S_{B_j} , the radius of influence R_{B_j} and the stiffness coefficient k_{B_j} are set equal to S_{A_i} , R_{A_i} and k_{A_i} respectively.

3.4. Characterization of the generic elements

The time varying blob model may be characterized by the evolution of generic components $C_{ij}(t)$ associated to each pair of initial-final sub-components (A_i, B_j) . Thus, intermediate shapes may be defined by their global field function as follows :

$$f_C(t) = \sum_{i,j} f_{C_{ij}}(t)$$

The surface of intermediate shapes is defined as the points of space whose potential equals an intermediate threshold value $T_C(t)$ that interpolates the initial and the final thresholds T_A and T_B .

The characterization of the time varying elements $C_{ij}(t)$ is the key of the algorithm. The computation of the skeleton $S_{C_{ij}(t)}$, the distance function $d_{C_{ij}(t)}$ and the field function $g_{C_{ij}(t)}$ is detailed in the next paragraphs.

3.4.1. Skeletons

The time varying skeleton $S_{C_{ij}(t)}$ is defined as the linear interpolation based on Minkowski sum of the skeletons S_{A_i} and S_{B_j} of the initial and final associated sub-components, formally :

$$S_{C_{ij}(t)} = (1-t)S_{A_i} \oplus tS_{B_j}$$

Let us recall the following fundamental results. The linear interpolation based on Minkowski sums implicitly interpolates polygonal shapes of any dimension. Whenever the argument shapes A and B are convex, the topology of the intermediate polytope $(1-t)A \oplus tB$ remains unchanged throughout time (there is neither creation nor destruction of vertices,

edges or faces). As the topology of $(1-t)A \oplus tB$ is the same as the topology of $A \oplus B$, we will address the characterization of $A \oplus B$ in the remainder of this paragraph.

Any vertex V_C of C may be written as $V_C = V_A \oplus V_B$, V_A and V_B are said to be the parent vertices of V_C , and the relationships between V_A , V_B and V_C is unique^{24, 12, 7}.

Therefore, $C = A \oplus B$ may be defined as the convex hull of the vertices V_C provided the argument polyhedra are convex. The transformation may be characterized by a *generic topology*, whose geometry is defined in terms of a reference table between the vertices of C and the parent vertices of A and B . As the topology remains unchanged throughout the transformation, both the adjacency graph and the reference table may be created once and for all.

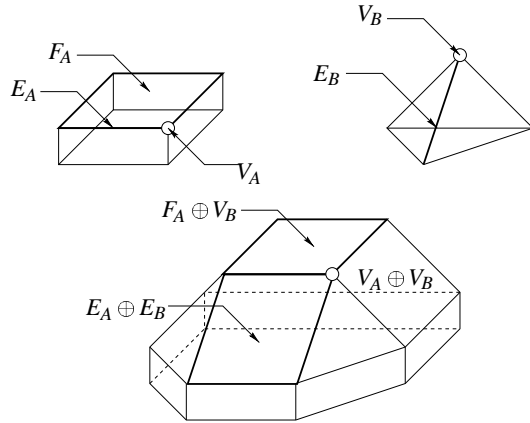


Figure 5: Minkowski sum of two polyhedra

Straightforward techniques consist in building the convex hull of candidates vertices $V_C = V_A \oplus V_B$, and require $O(n_A^2 n_B^2)$ computational time, where n_A and n_B stand for the number of vertices of A and B . Although the Minkowski sum of low dimension polytopes may be computed this way, computational time becomes prohibitive with complex polyhedra. In our own implementation, we rely on accelerated techniques that take advantage of the topology of the argument polyhedra. We rely on a divide and conquer technique²⁴ that lowers the overall complexity to $n_C \ln(n_C)$, where n_C stands for the number of vertices of C .

3.4.2. Field functions

The time dependent field functions $g_{C_{ij}(t)}(r)$ should interpolate the field functions of the sub-components $s_j(A_i)$ and $s_i(B_j)$. In the most general case, the direct linear interpolation may cope with any kind of field function. $g_{C_{ij}(t)}$ may be defined as follows, where the weights α_{ij} and β_{ij} result from the splitting process detailed in paragraph 3.2.

$$g_{C_{ij}(t)}(r) = (1-t)\alpha_{ij}g_{A_i}(r) + t\beta_{ij}g_{B_j}(r)$$

The computation of $g_{C_{ij}(t)}$ may not be straightforward however. Several problems arise when using piecewise field functions defined over different intervals or rational polynomials⁴. For instance, the number of intervals that partition the field function kernel increase. Moreover, summing rational polynomials of different denominators increases the degree of the resulting numerator polynomial. Therefore, analytic ray-tracing techniques^{27, 9}, that require polynomial root solving become computationally prohibitive.

Direct interpolation results in an unwanted loss of coherence in the intermediate field function definition (figure 6), which consequently results in a loss of shape coherence during the metamorphosis.

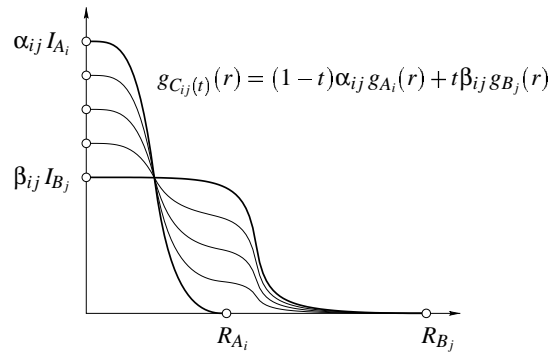


Figure 6: The direct linear interpolation of field functions results in a loss of coherence in the definition of $g_{C_{ij}(t)}$

We use an alternative technique, based on the characterization of field function *classes* to overcome those problems. Let us recall that in our implementation, field functions are piecewise rational polynomials controlled by a set of parameters. Those field functions share the same parametrized definition, *i.e.* have the same mathematical properties and thus form a class.

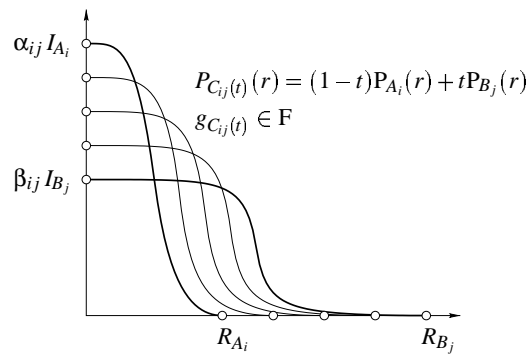


Figure 7: Interpolation of field functions that belong to the same class with parameter interpolation

Therefore, field functions g_{A_i} and g_{B_j} belong to the same

class F characterized by a set of parameters denoted P_A and P_B respectively. The intermediate field function $g_{C_{ij}(t)}(r)$ may be defined as a specific element of that class F , whose parameters $P_{C_{ij}(t)}$ are interpolating the parameters of the initial and the final field function $g_{A_i}(r)$ and $g_{B_j}(r)$.

The parametrized field function will be said to be *generic*, and time specific field function will be referred to as a precise *instantiation* of the model. The code needed for whatever computations (ray-tracing, polygonization) is always the same, which increases speed.

3.4.3. Distance functions

The transformation of distance functions problem is almost identical to the field function transformation problem. The direct interpolation provides us with a general means for transforming d_{A_i} into d_{B_j} :

$$d_{C_{ij}(t)}(x, y, z) = (1-t)d_{A_i}(x, y, z) + td_{B_j}(x, y, z)$$

This technique is computationally expensive however as the evaluation of $d_{C_{ij}(t)}$ requires twice as many distance function evaluation as usual for each sub-component.

Although any kind of distance function might qualify, we limited our study to the distance derived from the L^p metrics⁹, which provide an extension of the usual Euclidian distance. Let us recall the definition of the distance function based on L^p metrics :

$$d(x, y, z) = \left(|x|^p + |y|^p + |z|^p \right)^{\frac{1}{p}}$$

In our implementation, all distance functions are derived from a L^p metric characterized by a single parameter $p \in [1, +\infty]$. Therefore, we characterize the intermediate distance function by interpolating the exponent parameter, setting $p_{C_{ij}(t)} = (1-t)p_{A_i} + tp_{B_j}$.

3.5. Trajectory and shape control

As demonstrated in⁸, the use of the Minkowski sum implicitly generates a linear trajectory path for each time varying skeletal element. Therefore elements may cross during the transformation, which often results in a loss of shape coherence (figure 8).

Moreover, the Minkowski sum generally increases the dimension of the time varying skeletons. For instance the transformation between two non-collinear line segments results in a deforming polygon (figure 9), which adds up to the loss of shape coherence.

Formally, the computation of the Minkowski sum of two skeletons S_{A_i} and S_{B_j} of the elements is performed by computing the convex hull of the vertices $V_{S_{A_i}} \oplus V_{S_{B_j}}$ using the world coordinate system, denoted as R_0 . The deforming skeleton $S_{C_{ij}(t)}$ is defined as :

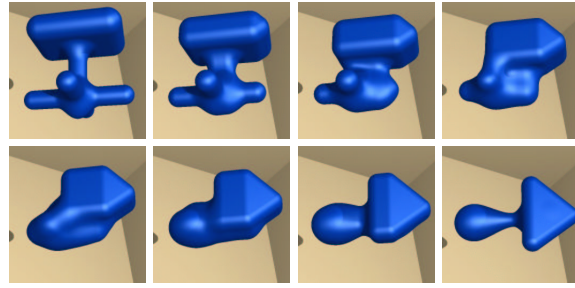


Figure 8: The direct interpolation based on the Minkowski sum of the skeletons results in a loss of shape coherence during the transformation

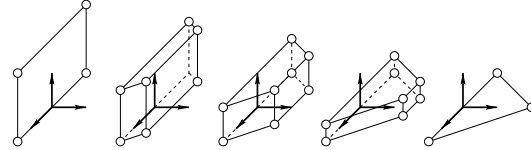


Figure 9: Deforming polyhedron resulting from the interpolation of non coplanar skeletal elements

$$S_{C_{ij}(t)}|_{R_0} = (1-t)S_{A_i}|_{R_0} \oplus tS_{B_j}|_{R_0}$$

We use a local frame system to control both the *dimension* of the intermediate skeletons and their *trajectory path*, which provides the animator with a tight control over the transformation. Therefore, an element A_i of a blob A will be fully characterized by its coordinate system R_{A_i} , its skeleton S_{A_i} , its distance function $d_{A_i}(x, y, z)$ and its potential function $f_{A_i}(r)$, and referred to as :

$$\left\{ R_{A_i}, S_{A_i}, d_{A_i}, f_{A_i} \right\}$$

Given two models A and B , we define the transformation between two skeletons S_{A_i} and S_{B_j} as follows :

$$S_{C_{ij}(t)}|_{R_{C_{ij}(t)}} = (1-t)S_{A_i}|_{R_{C_{ij}(t)}} \oplus tS_{B_j}|_{R_{C_{ij}(t)}}$$

The Minkowski sum is created in a time varying ghost coordinate system, denoted as $R_{C_{ij}(t)}$, with the sole constraint that it should interpolate the initial and the final coordinate systems R_{A_i} and R_{B_j} (figure 10). Several interpolation schemes have been proposed. A complete overview is beyond the scope of this paper, however we briefly present the technique we rely on.

The interpolation between R_{A_i} and R_{B_j} may be achieved by key-framing techniques where the animator controls the location and the orientation of intermediate coordinate systems (figure 10). In our implementation, the origin of the co-

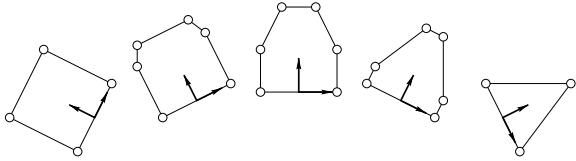


Figure 10: Constraining polygons in the same plane reduces the dimension of intermediate skeletons and creates a set of deforming polygons

ordinate system $R_{C_{ij}(t)}$ follows a cubic spline curve, whereas the smooth interpolation of the orientations involves quaternion sleeping^{25, 2}.

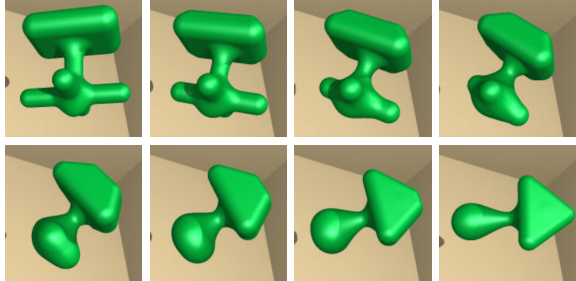


Figure 11: Frame interpolation avoids blobby intermediate steps in the animation and preserves shape coherence

The choice of a local coordinate system associated to the skeletons S_{A_i} and S_{B_j} provides the animator with a wide span of visual effects. However, the process may be automated with a view to systematically reducing the dimension of the varying skeletons. For instance, constraining that all line segments and polygonal skeletons should lie in the same plane reduces the dimension of intermediate skeletons and avoids the creation of deforming polyhedra (figure 11).

4. BlobTree metamorphosis

In this section, we address the metamorphosis of implicit surfaces characterized by a BlobTree data-structure. Unlike blobs that are characterized by a mere set of elements that blend in the same way, the BlobTree is defined by a *hierarchical* combination of primitives organized in a tree data-structure whose nodes are characterized as *heterogeneous operators*, such as blending, boolean operators or even warping. Therefore, we are confronted with the following issues:

- The correspondence process is no longer as straightforward as it used to be in for the blob model. Because of the hierarchical structure of the BlobTree, the nodes and the leaves of the source and target models may not be matched freely. In practice, the graph of correspondence must remain compatible with the tree hierarchy. For instance the

nodes at a given level should not be matched unless their ancestor nodes have also been paired.

- As the correspondence process matches nodes and leaves of the source and target models, we need to characterize transformations between heterogeneous operators. For instance we need to define the interpolation between a union and a blending operator.

As recalled in the previous section, the key idea of the blob morphing method is the decomposition of multiply matched elements into sub-elements⁷. This process allows the creation of a bijective graph of correspondence which eventually leads to the generic blob model. In the following paragraphs, we demonstrate that we can directly adapt this decomposition method to the union, intersection and blending operators, whereas the difference and warping operators deserve a special case. Instead of splitting the difference and warping nodes, the BlobTree will be updated into equivalent structures.

4.1. Overview of the algorithm

Given two BlobTree models A and B , we aim at creating two new *overlapping* BlobTrees A' and B' whose nodes and leaves can be bijectively paired. This involves the creation of a graph of correspondence matching two BlobTrees compatible with the tree structure of both the source and the target models.

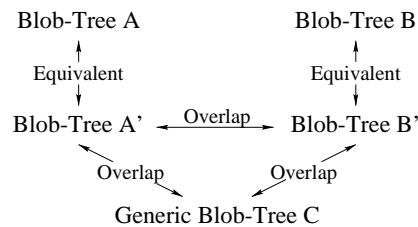


Figure 12: Generic BlobTree creation process

Starting from the roots of the BlobTree, we iteratively descend down the BlobTree data-structures and create a graph of correspondence between the nodes at the same level. Whenever a node N_A of A holds several correspondence links, it is split into sub-nodes N_{A_i} and the BlobTree structure A is updated into an equivalent form A' . The same process is simultaneously performed on the nodes N_B of B .

The correspondence process ends at the leaves of either BlobTree. Multiply matched leaves of the BlobTree are split as described in Galin et al.⁷. In practice, this correspondence process may be either controlled by the animator or achieved automatically through heuristics.

The *generic* BlobTree model C is implicitly created during the correspondence and the decomposition steps. The structure of C is the same as the overlapping structures of A' and B' . The nodes and the leaves of C are defined as time

varying tuples $\{N_{A'}, N_{B'}, s(t)\}$ where $s(t)$ refers to the speed of transformation between the nodes $N_{A'}$ and $N_{B'}$. As we will see in section 4.4, those time varying nodes will be denoted as :

$$N_{C(t)} = N_{A'} \xrightarrow{s(t)} N_{B'}$$

The operators of the BlobTree fall into two categories : the commutative operator set and the difference and warping operators that are not commutative.

Notations In the remainder of this paper, we will refer to the following terminology. The set of union, intersection and blending operators (the commutative operator set) will be referred to as $\{\cup, \cap, +\}$ for short. The other two sub-set will be denoted as $\{/ \}$ and $\{\omega\}$ respectively.

4.2. Node decomposition background

In this section, we present the decomposition background that will enable us to create the equivalent BlobTree models during the correspondence process. As we will see in section 4.3, the decomposition algorithm will be only applied to the commutative nodes (union, intersection and blending nodes). Warping and difference node will deserve a special case.

In this section, we will refer to the following prefixed notation : $N_A(N_{A1}, \dots, N_{An})$ will denote a node N_A taken in the whole set $\{\cup, \cap, +, /, \omega\}$ and its n children N_{Ai} . The following property will be involved in the decomposition algorithm.

Scaling nodes Let $\alpha > 0$ a real positive number, the scaled node αN_A is recursively defined as :

$$\alpha N_A = N_A(\alpha N_{A1}, \dots, \alpha N_{An})$$

If N_A is a leaf characterized by a potential field function f , then αN_A is the primitive characterized by $\alpha \times f$. For instance, the maximum intensity of the blending function of the skeletal element is scaled by α .

The interpretation in the case of operators relies on the function definition of the operators of the BlobTree. We restrict to binary nodes out of clarity. Let us define $\alpha \cup$. In this case, we use the distributivity of α with respect to the maximum :

$$\begin{aligned} (\alpha \cup)(f, g) &= \alpha \max(f, g) \\ &= \max(\alpha f, \alpha g) = \cup(\alpha f, \alpha g) \end{aligned}$$

The same demonstration applies to the blending, intersection and difference operators as well. In the case of warping operators, we have the following definition of $\alpha \omega$:

$$\begin{aligned} (\alpha \omega)(f) &= \alpha(f \circ \omega^{-1}) \\ &= (\alpha f) \circ \omega^{-1} = \omega(\alpha f) \end{aligned}$$

Decomposition algorithm Let A and B two BlobTrees. Let $N_A(N_{Ai})$ and $N_B(N_{Bi})$ two corresponding commutative nodes with n_A and n_B children respectively. Whenever a node N_{Ai} is multiply matched with $2 \leq n \leq n_B$ nodes N_{Bj} , we split N_{Ai} so that the decomposition will be neutral with respect to the parent node N_A :

- If the parent node N_A is a union or an intersection operator, N_{Ai} is replicated n times. This decomposition creates an equivalent model as this replication is neutral with respect to the union and the intersection.
- If N_A is a blending operator, the previous straightforward replication technique cannot be applied. Instead, we create n weighted copies $\alpha_i N_{Ai}$ with the constraint that the weights α_i should be positive and form a partition of unity as described in section 3.2.

As we will see in the next section, the decomposition never applies to difference and warping operators.

4.3. The correspondence process

Let A and B two BlobTrees whose structures do not overlap. We aim at transforming A and B into *equivalent* models A' and B' that do *overlap*. Our method iteratively parses the BlobTree data-structures A and B and simultaneously creates the equivalent BlobTrees A' and B' .

The algorithm starts by matching the roots of the argument BlobTrees. At each level, we apply the following two steps to each pair of matched nodes (N_A, N_B) . First we invoke a correspondence process that creates correspondence links between the children nodes of N_A and N_B . Then, we apply a decomposition scheme to the multiply matched children with a view to creating the new equivalent structures that will eventually overlap. The splitting process strictly follows a set of decomposition rules that depend on the types of the matched operators.

In the most general case, we need to solve the correspondence between all the possible operators of the BlobTree, *i.e.* we address the transformation of the whole set $\{\cup, \cap, +, /, \omega, \}$.

Difficult cases involve the correspondence between commutative and non-commutative operators. We propose to reduce the complexity by creating *equivalent* BlobTree structures whenever needed to avoid those difficult correspondences. This enables us to focus on the following three simpler correspondence problems : commutative operator correspondence $\{\cup, \cap, +\} \rightarrow \{\cup, \cap, +\}$, difference to difference $\{/ \} \rightarrow \{/ \}$ and warping to warping $\{\omega\} \rightarrow \{\omega\}$ correspondences.

For each tuple (N_A, N_B) of nodes paired by a link in the correspondence graph, the children of N_A and N_B may be paired according to the following rules.

Commutative nodes Whenever N_A and N_B are chosen from the subset union, intersection, and blending (*i.e.* we address the transformation $\{\cup, \cap, +\} \rightarrow \{\cup, \cap, +\}$), any child of N_A may be paired with any child of N_B . Automatic pairing may match all children of N_A with all children of N_B , however this results in the creation of $n_A \times n_B$ sub-nodes which often results in featureless intermediate shapes. In practice, the animator may freely control the correspondence process (figure 13).

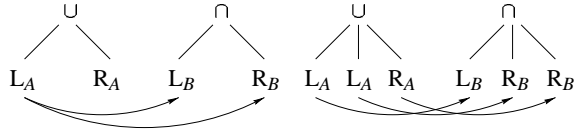


Figure 13: Decomposition of a left node L_A , whose parent node is a union operator, paired with two target nodes L_B and R_B . The right node R_B is also split accordingly

The correspondence process is recursively applied to the children of N_A and N_B .

Difference nodes This operator deserves a special case as it is a non-commutative binary node. No problem occurs whenever two difference nodes are paired, *i.e.* we address the transformation $\{/ \} \rightarrow \{/ \}$: we suggest that the left children and the right children should be automatically matched.

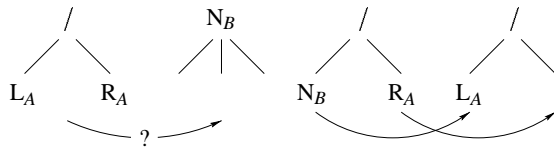


Figure 14: Matching a difference operator $N_A = /$ with a commutative operator N_B

The correspondence between the difference operator and a commutative operator cannot be performed directly in the general case. We propose to tackle the correspondence problem $\{/ \} \rightarrow \{\cap, \cup, +\}$ by creating equivalent BlobTree structures that will be transformed easily. Let $N_B \in \{\cap, \cup, +\}$, let N_A be the difference operator, and L_A and R_A the left and the right children of N_A respectively. We proceed as follows (figure 14):

- We insert a difference operator above N_B , whose left child is set to N_B , and whose right child is a phantom node, *i.e.* a null field function in practice.
- We keep the left and right children L_A and R_A of the difference node unchanged.

The correspondence and decomposition process are recursively applied to the leaves of the equivalent nodes. This

technique enables us to match a difference operator with a union, an intersection or a blending operator. The correspondence between a difference and a warping $\{/ \} \rightarrow \{\omega\}$ is addressed in the next paragraph.

Warping nodes Whenever two warping nodes are paired, the correspondence process is trivial as we only need to pair the unique child of N_A with the unique child of N_B . As for the difference operator, warping nodes deserve a special algorithm when we address the correspondence problem $\{\omega\} \rightarrow \{\cap, \cup, +, / \}$.

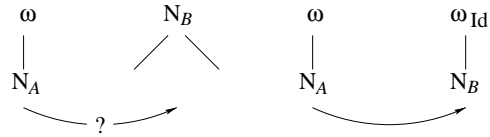


Figure 15: Matching a warping operator with a non warping operator

Let N_A the warping operator and N_B its paired counterpart. We perform the transformation shown figure 15, inserting the identity warping operator at the top of the right tree, to match the warp of the left tree. The node N_A of the right tree is matched with N_B in the left tree, and recursion proceeds further.

4.4. Evaluation of the generic BlobTree model

Let us recall that at this step of the metamorphosis algorithm, we have created two *overlapping* BlobTree models A' and B' whose nodes and leaves are bijectively matched. The intermediate *generic* BlobTree model is generated by traversing the trees and creating *morphing* nodes and leaves for each correspondence link.

In this section, we focus on the creation of the time varying nodes. The transformation of the skeletal primitives is performed as for blobs (see section 3).

Let N_A and N_B the nodes of two overlapping BlobTrees A and B . The metamorphosis between the nodes N_A and N_B at speed $s(t)$ will be referred to as :

$$N_{C(t)} = N_A \xrightarrow{s(t)} N_B$$

The generic BlobTree describing the metamorphosis is created by morphing all the nodes in correspondence. When the process reaches the leaves of the BlobTrees, we invoke the transformation of the skeletal primitives as for blobs (see section 3).

Let us recall that our correspondence and decomposition algorithms result in the following three correspondence sub-problems : the commutative node transformation problem $\{\cup, \cap, +\} \rightarrow \{\cup, \cap, +\}$, and the metamorphosis of two differences $\{/ \} \rightarrow \{/ \}$ and two warps $\{\omega\} \rightarrow \{\omega\}$.

Commutative operator metamorphosis Whenever the nodes N_A and N_B hold the same operator, we simply define $N_{C(t)}$ equal to N_A . In the general case, we propose the following definition of time varying nodes, which holds for whatever union, intersection or blending operators :

$$N_A \xrightarrow{s(t)} N_B = (1 - s(t))N_A + s(t)N_B$$

For instance, the function that evaluates the potential field of the morphing operator interpolating the union and the intersection operators may be written as :

$$f_{U \rightarrow \cap} = (1 - s(t)) \max_{i \in [1, n]} (f_i) + s(t) \min_{i \in [1, n]} (f_i)$$

As mentioned in Wyvill et al. ²⁸, super-elliptic blending may be used to smoothly morph the union and the blending operators. Let us recall that super-elliptic blending, referred to as f_n , is defined as follows :

$$f_n = \left(\sum_{i=1}^n f_i^n \right)^{\frac{1}{n}}$$

The standard blending operator $+$ proves to be a special case of the super-elliptic blend with $n = 1$. When n varies from 1 to infinity, it creates a set of blends interpolating between blending and union.

Morphing difference operators In the case $\{/\} \rightarrow \{/\}$, no problem occurs as we simply need to define the morphing operator as the difference operator.

Morphing warping operators Let ω_A and ω_B two warping operators. In the most general case, the time varying warping operator may be defined by interpolating the warping functions as for commutative operators :

$$\omega_A \xrightarrow{s(t)} \omega_B = (1 - s(t))\omega_A + s(t)\omega_B$$

Unfortunately, we have observed that this technique may create over-distorted intermediate shapes. We propose an alternative approach that consists in decomposing the warping correspondence into simpler sub-problems.

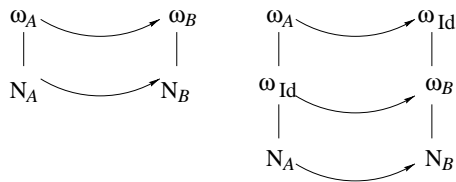


Figure 16: Decomposing the warping correspondence into simpler sub-problems

As mentioned in section 2, warps are chosen from the Barr operators ¹, namely twist, taper and bend, and affine geometric transformations. Those deformation tools are characterized by a few parameters (e.g. an axis and a rotation angle

for the twist). We proceed as follows. If the warps ω_A and ω_B are of the same type, we define $\omega_{C(t)}$ as the same type of warping with interpolated parameters. For instance the interpolation of two rotations results in a time varying rotation node, whose angle parameters are computed using quaternion slerping ². Otherwise, we need to split the global warp transformation into two simpler cases (figure 16).

The evaluation of the time varying warping operator may be written as follows :

$$\omega_A \xrightarrow{s(t)} \omega_B = \left(\omega_A \xrightarrow{s_A(t)} \omega_{Id} \right) \circ \left(\omega_{Id} \xrightarrow{s_B(t)} \omega_B \right)$$

This approach enables us to metamorphose any kind of warping operator, including geometric transformations that appear in the scene graph structure of the BlobTree.

5. Metamorphosis sequences

We have implemented our metamorphosis method on Pentium II 450Mz workstations running Linux and used it to produce the following animations.

Cube to pierced sphere This metamorphosis (figure 17) illustrates that our method can handle metamorphosis between shapes of completely different topologies.



Figure 17: Metamorphosis between a pierced sphere and a rounded box

The initial shape is characterized by a single skeletal element (the rounded cube), whereas the final shape is defined as the difference between a sphere and the union of three axis-aligned boxes.

As the root nodes of those two BlobTrees (a skeletal primitive and a difference operator respectively) cannot be matched automatically, the initial BlobTree is rewritten into an equivalent form as the difference between the rounded cube primitive and a phantom element (a sphere that the animator located at the center of the cube in that specific animation).

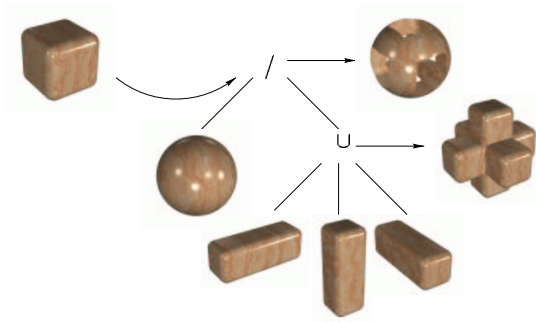


Figure 18: Correspondence problem between two BlobTrees of different structure

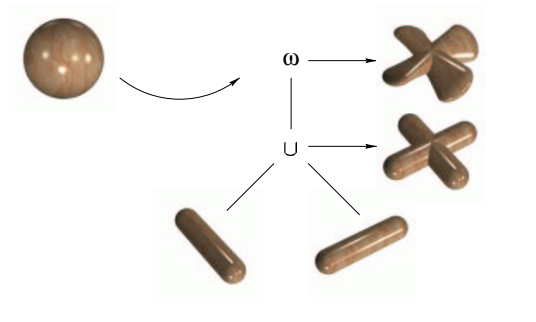


Figure 20: Correspondence problem between two BlobTrees of different structure

Sphere to Propeller This sequence (figure 19) shows the transformation of a sphere into an helix defined by two blended and twisted rods. During the correspondence process, our method automatically inserts a twisting operator at the root of the BlobTree model of the sphere with a null angle parameter.



Figure 19: Metamorphosis between a propeller and a sphere

In this example, the initial shape is a sphere whereas the final shape is defined as a twisted union of two rounded cylinders. The initial BlobTree needs to be rewritten, inserting a warping node at the root and splitting the sphere into the union of two spheres so that equivalent BlobTrees should overlap.

Tower to queen This sequence (figure 21) illustrates the transformation between two complex BlobTree models: a rook morphing into a queen. This animation exhibits the transformation between two difference operators: the top of the rook that is defined as the difference between a cylinder and the union of two boxes, has been matched with the top of the queen whose carvings have been defined by subtracting eight small spheres from the upper part.



Figure 21: Metamorphosis between a rook and a queen

6. Conclusion and future work

The key feature of the BlobTree is its ability to model complex objects with a small number of skeletal primitives combined with blending, warping and boolean operators. In this paper we have presented an original BlobTree morphing technique that provides the animator with both low and high level tools to achieve coarse and fine control over the transformation. The overall metamorphosis is described by a generic BlobTree model, whose nodes are characterized as time varying operators. In practice, the user may rely on pre-defined or user-defined interpolating operators. A global control may be performed by only matching the upper nodes of the source and target BlobTrees. A finer control can be achieved by specifying the trajectory and the transformation speed of the skeletal elements.

In our own implementation, we rely on Barr operators¹ that form a restricted class of warps. Future research could be pursued to generalize our metamorphosis technique to other morphing operators. We could extend warping operators to the Free Form Deformations proposed by Sederberg²³ or to Axial Deformations¹⁷. The integration of those deformation methods in our modeling and morphing environment would require the development of an extended interactive user interface.

References

1. A. H. Barr. Global and Local Deformations of Solid Primitives. *Computer Graphics (Siggraph'84 Proceedings)*, **18**: 21–30, July 1984.
2. A. H. Barr, B. Currin, S. Gabriel, J. F. Hughes. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics (Siggraph'92 Proceedings)* **26**: 313–320, July 1992.
3. T. Beier and S. Neely. Feature Based Image Metamorphosis. *Computer Graphics (Siggraph'92 Proceedings)*, **26**: 35–42, July 1992.
4. C. Blanc and C. Schlick. Extended field functions for soft objects. *Proceedings of Implicit Surfaces'95*, 21–32, Grenoble, April 1995.
5. J. Bloomenthal, C. Bajaj, J. Blinn, M. P. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*, Morgan Kaufmann 1997.
6. S. Coquillart and P. Jancene. Animated Free Form Deformation. *Computer Graphics (Siggraph'91 Proceedings)*, **24**(4): 23–26, July 1991.
7. E. Galin and S. Akkouche. Soft Object Metamorphosis based on Minkowski sums. *Proceedings of Eurographics'96*, **15**(3): 143–153, August 1996.
8. E. Galin and S. Akkouche. Shape Constrained Blob Metamorphosis. *Proceedings of Implicit Surfaces'96*, 9–23, October 1996.
9. E. Galin. Métamorphose et Visualisation de Blobs à Squelettes. *Ph.D. Thesis*, University Claude Bernard Lyon 1, July 1997.
10. T. He, S. Wang, A. Kaufman. Wavelet-Based Volume Morphing. *Proceedings of Visualization'94*, 85–91, 1994.
11. J. F. Hughes. Scheduled Fourier Volume Morphing. *Computer Graphics (Siggraph'92 Proceedings)*, **26**(2): 43–46, 1992.
12. A. Kaul and J. Rossignac. Solid interpolating deformations, construction and animation of pips. *Computer Graphics*, **16**(1): 107–115, January 1992.
13. J. R. Kent, R. E. Parent and W. E. Carlson. Shape Transformation for Polyhedral Objects. *Computer Graphics (Siggraph'92 Proceedings)*, **26**(2): 47–54, July 1992.
14. T. Kanai, H. Suzuki and F. Kimura. Three dimensional Geometric Metamorphosis based on Harmonic Maps. *The Visual Computer* **14**(4): 166–176, 1998.
15. T. Kanai, H. Suzuki and F. Kimura. Metamorphosis of Arbitrary Triangular Meshes with User-Specified Correspondence. *IEEE Computer Graphics and Applications* (to appear).
16. J. R. Kent, R. E. Parent and W. E. Carlson. Shape Transformation for Polyhedral Objects. *Computer Graphics (Siggraph'92 Proceedings)*, **26**(2): 47–54, July 1992.
17. F. Lazarus, S. Coquillart and P. Jancène. Axial deformations: an intuitive deformation technique. *Computer-Aided Design*, **26**(8), 607–613, August 1994.
18. F. Lazarus and A. Verroust. Metamorphosis of Cylinder-like Objects. *Journal of Visualization and Computer Animation*, **8**: 131–146, 1997.
19. F. Lazarus and A. Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, **14**(8/9): 373–389, 1998.
20. A. Lee, D. Dobkin, W. Sweldens and P. Schröder. Multiresolution Mesh Morphing. *Computer Graphics (Siggraph'99 Proceedings)*, August 1999.
21. A. Leros, C. D. Garfinkle and M. Levoy. Feature-Based Volume Metamorphosis. *Computer Graphics (Siggraph'95 Proceedings)*, 449–456, August 1995.
22. A. Pasko and V. Savchenko. Constructing functionally defined surfaces. *Proceedings of Implicit Surfaces'95*, 97–106, Grenoble, France, 1995.
23. T. Sederberg and S. Parry. Free Form Deformation of Solid Geometric Models. *Computer Graphics (Siggraph'86 Proceedings)*, **23**(3): 151–160, August 1986.
24. J. Serra. *Image Analysis and Mathematical morphology*, Academic Press Inc., 1982.
25. K. Shoemake. Animating Rotation with Quaternion Curves. *Computer Graphics (Siggraph'85 Proceedings)*, **19**: 245–254, July 1985.
26. B. Wyvill, C. McPheeters and G. Wyvill. Data Structure for Soft Objects. *The Visual Computer*, **2**(4): 227–234, 1986.
27. G. Wyvill and A. Trotman. Ray Tracing Soft Objects. *Computer Graphics International'90*: 469–476, 1990.
28. B. Wyvill, A. Guy and E. Galin. Extending the CSG Tree (Warping, Blending and Boolean Operations in an Implicit Surface Modeling System). *Computer Graphics Forum*, **18**(2): 149–158, 1999.