Blob-Tree Metamorphosis

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 Blob-Tree 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 Blob-Tree. 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 Blob-Tree 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 Blob-Tree model. This provides him with a tight control over the transformation so as to achieve good visual effects.

Keywords: animation, Blob-Tree, 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.

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

Lazarus has presented an interesting survey of existing metamorphosis techniques [15]. The proposed classification exhibits two major categories according to the underlying shape model : techniques focusing on mesh models and volumetric methods.

1.1.1 Boundary representation approaches

Methods that directly work on the boundary representation of polygonal meshes need to solve the *vertex correspondence problem*. Although many approaches have been proposed, no general solution has been presented so far. Kent [13] first proposed to merge the topologies (*i.e.* the adjacency graph) of star shaped polyhedral objects. Lazarus [14] later generalized that technique to a wider class of objects. Kaul [9] has proposed to tackle the vertex correspondence problem by using the weighted Minkowski sum of argument polyhedra which works well for convex objects.

Kanai [11, 12] 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, Lee [16] 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 size of the mesh increases, and the control of the transformation difficult.

1.1.2 Volumetric methods

Unlike mesh based techniques, volumetric methods can handle the metamorphosis of shapes of different topologies easily. Volumetric approaches further separate into two categories : voxel based approaches that sample space on a regular grid, and implicit surface techniques that work on the equations characterizing the volume of the objects.

Lerios [17] first extended the two-dimensional morphing algorithm proposed by Beier [2], warping and cross-dissolving matched density volumes. Hughes [8] and He [7] proposed a method based on the Fourier and on a wavelet decomposition of space respectively. Voxel based approaches are memory consuming, and 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.

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

Pasko [18] has proposed a general morphing scheme, using the linear interpolation to define the intermediate implicit functions characterizing the transformation. Wyvill [3] 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 point convex skeletons of any dimension [5]. Our method provides a good control of the transformation. The animator may freely pair sets of primitives of the source and target models, and even control the trajectory [6] of intermediate primitives to avoid amorphous or featureless transitions. Still remains the fact that blobs specifically model smooth objects, and are ill suited to create non-organic shapes, which consequently limits the scope of the proposed metamorphosis technique.

1.2 Contributions

Recently, Wyvill has presented the Blob-Tree model [21] that can be thought of as an extension of blobs. The Blob-Tree 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 Blob-Tree 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 Blob-Tree model. Specifically, we make the following contributions :

- We extend the blob metamorphosis technique to the more general Blob-Tree model. We put the emphasis on the creation of a generic intermediate Blob-Tree 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 Blob-Tree 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 Blob-Trees and may specify both the speed of the transformation and the trajectory of the nodes and the leaves of the generic Blob-Tree model.

The remainder of this paper is organized as follows. To make it self-contained, we recall the fundamentals of the Blob-Tree model in section 2. In section 3, we address the metamorphosis of the Blob-Tree. We first recall the keys of our blob morphing technique, and adapt exiting algorithms to the tree structure of the Blob-Tree model. In section 4, we present morphing sequences.

2 The Blob-Tree

Let us recall that implicit surfaces built from skeletal elements, also known as *blobs* or *soft objects* [20], 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 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 potential function, and $d_i : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ refers to distance to the skeleton.

The Blob-Tree model [21] 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 [1], the Blob-Tree 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).

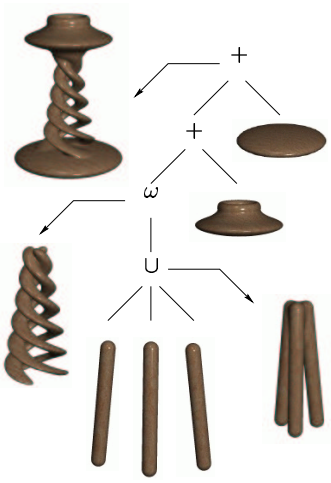

Figure 1: Candlestick created by blending the twisted union of three cylinders with tapered spheres

The evaluation of the field function in space is achieved by recursively traversing the Blob-Tree, 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 :

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

As suggested in [21], boolean operators may be defined by using Pasko's set theoretic functions [18]. In our own implementation however, we use the minimum and the maximum instead.

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

The Blob-Tree 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 function

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

$w(x, y, z)$ 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 taken among the Barr operators [1].

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

3 Metamorphosis

In this section, we address the metamorphosis of implicit surfaces characterized by a Blob-Tree data-structure. First, we recall the blob metamorphosis technique [5] that inspired the Blob-Tree metamorphosis method. Then, we address the creation of a generic Blob-Tree data-structure and put the emphasis on the key features inherited from the blob morphing approach.

3.1 Blob metamorphosis

As mentioned in the introduction, 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.

As suggested in [5], the graph of correspondence may be generated automatically by matching all the elements of the source and target models. Several heuristics for matching and interpolating soft objects based on skeletal elements have been proposed (Wyvill has presented a good overview in [3]). Those 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 null components whenever necessary.

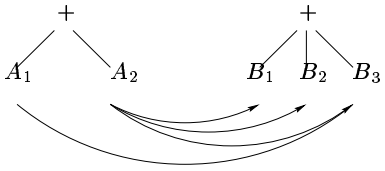

Figure 2: Graph of correspondence matching elements of the initial and final blobs

We assume that the animator provides us with a graph of correspondence that matches elements of the initial and the final shapes (figure 2). Elements that are not matched involve the creation of specific phantom elements. In practice, 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 Blob-Tree.

Since this graph is in general non-bijective, we split each component into sub-components with a view to creating a new graph bijectively matching those sub-components. An intermediate generic sub-component is associated to each graph link.

The whole transformation itself is characterized by a *generic* model whose instantiations interpolate the initial and the final shapes throughout time, computed as follows. Its time varying skeleton is defined as the linear interpolation based on Minkowski sum of the skeletons of the initial and final associated sub-components.

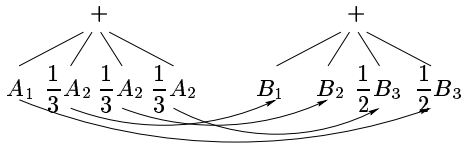

Figure 3: Creating a bijective graph through component splitting

As demonstrated in [6], the use of the Minkowski sum implicitly generates a trajectory path for each time varying skeletal element, which often results in a loss of shape coherence during the transformation. We have proposed to rely on local frame systems to control both the *dimension* of the intermediate skeletons and their *trajectory path*, which provides the animator with a tight control over the transformation.

Since directly interpolating distance and potential functions generates complex formulations, 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. The time varying implicit surface of intermediate shapes is defined as the points of space whose potential equals an intermediate threshold value that interpolates the thresholds of the initial and the final blob models.

Thus, we achieve a concise *generic representation*, which preserves the shape coherence during the transformation. Moreover, this generic representation enables us to combine interpolations so as to define a Bézier-like metamorphosis where control knots are replaced by control soft objects [5].

3.2 Blob-Tree metamorphosis

Unlike blobs that are characterized by a mere set of elements that blend in the same way, the Blob-Tree 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 Blob-Tree, 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 had 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-elements [5]. This process allows the creation of 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 Blob-Tree will be updated into equivalent structures.

3.2.1 Overview of the algorithm

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

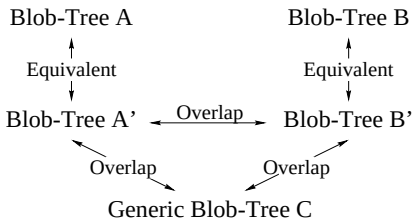

Figure 4: Generic Blob-Tree creation process

Starting from the roots of the Blob-Tree, we propose to iteratively descend down the Blob-Tree data-structures and create a graph of correspondence between the nodes at the same level. Whenever a node $\mathcal{N}_A$ of A holds several correspondence links, it is split into sub-nodes $\mathcal{N}_{A_i}$ and the Blob-Tree structure A is updated into an equivalent form A'. The same process is simultaneously performed on the nodes $\mathcal{N}_B$ of B.

The correspondence process ends when reaching the leaves of either Blob-Tree. Multiply matched leaves of the Blob-Tree are split as described in [5]. In practice, this correspondence process may be either controlled by the animator or achieved automatically through heuristics.

The *generic* Blob-Tree 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 $\{\mathcal{N}_{A'}, \mathcal{N}_{B'}, s(t)\}$ where $s(t)$ refers to the speed of transformation between the nodes $\mathcal{N}_{A'}$ and $\mathcal{N}_{B'}$. As we will see in section 3.2.4, those time varying nodes will be denoted as :

$$\mathcal{N}_{C(t)} = \mathcal{N}_{A'} \xrightarrow{s(t)} \mathcal{N}_{B'}$$

Notations The operators of the Blob-Tree fall into two categories : the commutative operator set and the difference and warping operators that are not commutative. 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.

3.2.2 Node decomposition background

In this section, we present the decomposition background that will enable us to create the equivalent Blob-Tree models during the correspondence process. As we will see in section 3.2.3, the decomposition algorithm will be only applied to the union, intersection and blending nodes.

In this section, we will refer to the following prefixed notation : $\mathcal{N}_A(\mathcal{N}_{A1}, \ldots, \mathcal{N}_{An_A})$ will denote a node $\mathcal{N}_A$ taken in the whole set $\{\cup, \cap, +, /, \omega\}$ and its n_A children $\mathcal{N}_{Ai}$. The following property will be involved in the decomposition algorithm.

Scaling nodes Let $\alpha > 0$ a real positive number, the scaled node $\alpha\mathcal{N}_A$ is equivalent to :

$$\alpha\mathcal{N}_A = \mathcal{N}_A(\alpha\mathcal{N}_{A1}, \ldots, \alpha\mathcal{N}_{An_A})$$

If $\mathcal{N}_A$ is a leaf characterized by a potential field function f, then $\alpha\mathcal{N}_A$ is the primitive characterized by $\alpha \times f$.

The interpretation in the case of operators relies on the function definition of the operators of the Blob-Tree. We restrict to binary nodes out of clarity. Let us define $\alpha\cup$. In the case of the union operator, 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 Blob-Trees. Let $\mathcal{N}_A(\mathcal{N}_{Ai})$ and $\mathcal{N}_B(\mathcal{N}_{Bi})$ two corresponding commutative nodes with n_A and n_B children respectively. Whenever a node $\mathcal{N}_{Ai}$ is multiply matched with $2 \leq n \leq n_B$ nodes $\mathcal{N}_{Bj}$, we split $\mathcal{N}_{Ai}$ so that the decomposition should be neutral with respect to the parent node $\mathcal{N}_A$:

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

  The heuristics for computing the coefficients α_i may be derived from [5] if the children of the nodes are skeletal elements, yet a simpler technique that performs well in the general case simply consists in setting $\alpha_i = 1/n$ for all sub-nodes.

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

3.2.3 The correspondence process

Let A and B two Blob-Trees 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 Blob-Tree data-structures A and B and simultaneously creates the equivalent Blob-Trees A' and B'.

The algorithm starts by matching the roots of the argument Blob-Trees. At each level, we apply the following two steps to each pair of matched nodes $(\mathcal{N}_A, \mathcal{N}_B)$. First we invoke a correspondence process that creates correspondence links between the children nodes of $\mathcal{N}_A$ and $\mathcal{N}_B$. 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 Blob-Tree, *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* Blob-Tree 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 $(\mathcal{N}_A, \mathcal{N}_B)$ of nodes paired by a link in the correspondence graph, the children of $\mathcal{N}_A$ and $\mathcal{N}_B$ may be paired according to the following rules.

Commutative nodes Whenever $\mathcal{N}_A$ and $\mathcal{N}_B$ are taken among the subset union, intersection, and blending (*i.e.* we address the transformation $\{\cup, \cap, +\} \rightarrow \{\cup, \cap, +\}$), any child of $\mathcal{N}_A$ may be paired with any child of $\mathcal{N}_B$. Automatic pairing may match all children of $\mathcal{N}_A$ with all children of $\mathcal{N}_B$, 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 5).

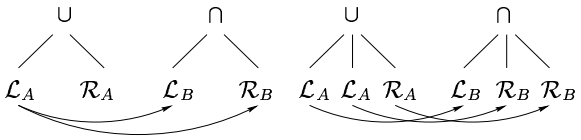

Figure 5: Decomposition of a left node $\mathcal{L}_A$, whose parent node is a union operator, paired with two target nodes $\mathcal{L}_B$ and $\mathcal{R}_B$. The right node $\mathcal{R}_B$ is also split accordingly

Multiply matched children of $\mathcal{N}_A$ and $\mathcal{N}_B$ are split into sub-nodes according to the previous decomposition rules.

Difference nodes This operator deserves a special case as it is only a 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.

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 Blob-Tree structures that will be transformed easily. Let $\mathcal{N}_B \in \{\cap, \cup, +\}$, let $\mathcal{N}_A$ the difference operator, and $\mathcal{L}_A$ and $\mathcal{R}_A$ the left and the right children of $\mathcal{N}_A$ respectively. We proceed as follows (figure 6) :

- We insert a difference operator above $\mathcal{N}_B$, whose left child is set to $\mathcal{N}_B$, and whose right child is the empty set, *i.e.* a null field function in practice.
- We replace the left child of the difference operator $\mathcal{L}_A$ by the same operator as $\mathcal{N}_B$ whose child will be set as $\mathcal{L}_A$, whereas $\mathcal{R}_A$ remains unchanged.

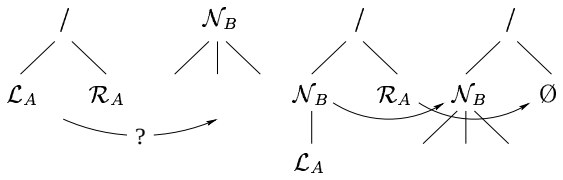

Figure 6: Matching a difference operator $\mathcal{N}_A = /$ with a commutative operator $\mathcal{N}_B$

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 $\mathcal{N}_A$ with the unique child of $\mathcal{N}_B$. As for the difference operator, warping nodes deserve a special algorithm when we address the correspondence problem $\{\omega\} \rightarrow \{\cap, \cup, +, /\}$.

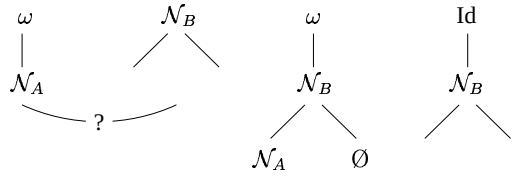

Figure 7: Matching a warping operator with a non warping operator

Let $\mathcal{N}_A$ the warping operator and $\mathcal{N}_B$ its paired counterpart. We perform the transformation shown figure 7, inserting an a phantom warping operator (the identity warp) at the top of the right tree, to match the warp of the left tree, and inserting operator $\mathcal{N}_B$ in the left tree, with a right child of empty to match the operator $\mathcal{N}_B$ in the right tree. This enables us to preserve the overall balance of the trees.

3.2.4 Evaluation of the generic Blob-Tree model

Let us recall that at this step of the metamorphosis algorithm, we have created two *overlapping* Blob-Tree models A' and B' whose nodes and leaves are bijectively matched. The intermediate *generic* Blob-Tree 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 3.1). Let $\mathcal{N}_A$ and $\mathcal{N}_B$ the nodes of two overlapping Blob-Trees A and B. The metamorphosis between the nodes $\mathcal{N}_A$ and $\mathcal{N}_B$ at speed $s(t)$ will be referred to as :

$$\mathcal{N}_{C(t)} = \mathcal{N}_A \xrightarrow{s(t)} \mathcal{N}_B$$

The generic Blob-Tree describing the metamorphosis is created by morphing all the nodes in correspondence. When the process

reaches the leaves of the Blob-Trees, we invoke the transformation of the skeletal primitives as for blobs (see section 3.1).

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 warpings $\{\omega\} \rightarrow \{\omega\}$.

Commutative operator metamorphosis Whenever the node $\mathcal{N}_A$ and $\mathcal{N}_B$ hold the same operator, we simply define $\mathcal{N}_{C(t)} = \mathcal{N}_A$. In the general case, we propose the following definition of time varying nodes, which holds for whatever union, intersection or blending operators :

$$\mathcal{N}_A \xrightarrow{s(t)} \mathcal{N}_B = (1 - s(t))\mathcal{N}_A + s(t)\mathcal{N}_B$$

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

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

As mentioned in [21], 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 $\oplus$ is defined as :

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

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 warpings 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.

As mentioned in section 1, warps are taken among Barr operators [1], namely twist, taper and bend. Let us recall that those deformations tools are parametrized 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. Otherwise, we need to split the global warp transformation into two simpler morphings (figure 8).

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

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

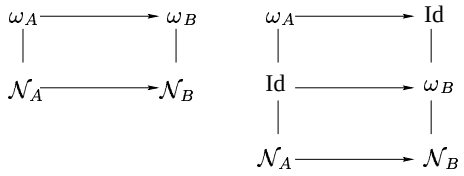

Figure 8: Turning the warping correspondence into simpler sub-problems

4 Metamorphosis sequences

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

Tower to queen This sequence (figure 9) shows the transformation of rook into a queen. This animation exhibits the transformation between two difference operators : the top of the rook 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 9: Metamorphosis between a rook and a queen

Cube to cut sphere This metamorphosis (figure 10) illustrates that our method can hanlde metamorphosis between shapes of completely different topologies.

Figure 10: Metamorphosis between a cut sphere and a rounded box

Sphere to Helix This sequence (figure 11) 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 Blob-Tree model of the sphere with a null angle parameter.

Figure 11: Metamorphosis between a propeller and a sphere

5 Conclusion and future work

The key feature of the Blob-Tree 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 Blob-Tree 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 Blob-Tree model, whose nodes are characterized as time varying operators. In practice, the user may rely on pre-defined interpolating operators, or chose its own morphing operators. A global control may be performed by only matching the upper nodes of the source and target Blob-Trees. A finer control can be achieved by specifying the trajectory and the transformation speed of the skeletal elements.

In this paper, we have put the emphasis on Barr operators [1] that form a restricted class of warpings. 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 Sederberg [19] or to Axial Deformations. 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)*, Vol. 18 : pages 21–30, July 1984.

[2] T. Beier and S. Neely. Feature Based Image Metamorphosis. *Computer Graphics (Siggraph'92 Proceedings)*, Vol. 26 : pages 35–42, July 1992.

[3] J. Bloomenthal, C. Bajaj, J. Blinn, M. P Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*, Morgan Kaufmann 1997.

[4] S. Coquillart and P. Jancene. Animated Free Form Deformation. *Computer Graphics (Siggraph'91 Proceedings)*, Vol. 24(4) : pages 23–26, July 1991.

[5] E. Galin and S. Akkouche. Soft Object Metamorphosis based on Minkowski sums. *Proceedings of Eurographics'96*, Vol. 15(3) : pages 143-153, August 1996.

[6] E. Galin and S. Akkouche. Shape Constrained Blob Metamorphosis. *Proceedings of Implicit Surfaces'96*, pages 9–23, October 1996.

[7] T. He, S. Wang, A. Kaufman. Wavelet-Based Volume Morphing. *Proceedings of Visualization'94*, pages 85–91, 1994.

[8] J. F. Hughes. Scheduled Fourier Volume Morphing. *Computer Graphics (Siggraph'92)*, Vol 26,2 : pages 43–46, 1992.

[9] A. Kaul and J. Rossignac. Solid interpolating deformations, construction and animation of pips. *Computer Graphics*, Vol 16(1) : pages 107–115, January 1992.

[10] J. R. Kent, R. E. Parent and W. E. Carlson. Shape Transformation for Polyhedral Objects. *Computer Graphics (Siggraph'92)*, Vol. 26(2) : pages 47–54, July 1992.

[11] T. Kanai, H. Suzuki and F. Kimura. Three dimensional Geometric Metamorphosis based on Harmonic Maps. *The Visual Computer* Vol. 14(4) : pages 166–176, 1998.

[12] T. Kanai, H. Suzuki and F. Kimura. Metamorphosis of Arbitrary Triangular Meshes with User-Specified Correspondence. *IEEE Computer Graphics and Applications* (to appear).

[13] J. R. Kent, R. E. Parent and W. E. Carlson. Shape Transformation for Polyhedral Objects. *Computer Graphics (Siggraph'92)*, Vol. 26(2) : pages 47–54, July 1992.

[14] F. Lazarus and A. Verroust. Metamorphosis of Cylinder-like Objects. *Journal of Vizualisation and Computer Animation*, Vol. 8 : pages 131–146, 1997.

[15] F. Lazarus and A. Verroust. Three-dimensional metamorphosis : a survey. *The Visual Computer*, Vol. 14(8/9) : pages 373–389, 1998.

[16] A. Lee, D. Dobkin, W. Sweldens and P. Shröder. Multiresolution Mesh Morphing. *Computer Graphics (Siggraph'99)*, August 1999.

[17] A. Lerios, C.D. Garfinkle and M. Levoy. Feature-Based Volume Metamorphosis. *Computer Graphics (Siggraph'95)*, pages 449-456, August 1995.

[18] A. Pasko and V. Savchenko. Constructing functionally defined surfaces. *Proceedings of Implicit Surfaces'95*, pages 97–106, Grenoble, France, 1995.

[19] T. Sederberg and S. Parry. Free Form Deformation of Solid Geometric Models. *Computer Graphics (Siggraph'86)*, Vol. 23(3) : pages 151–160, August 1986.

[20] B. Wyvill, C. Mc.Pheeters and G. Wyvill. Data Structure for Soft Objects. *The Visual Computer*, Vol. 2(4) : pages 227–234, 1986.

[21] B. Wyvill, A. Guy and E. Galin. Extending the CSG Tree (Warping, Blending and Boolean Operations in an Implicit Surface Modeling System). *Proceedings of Implicit Surfaces'98*, pages 113–121, June 1998.

Figure 12: Metamorphosis between a rook and a queen

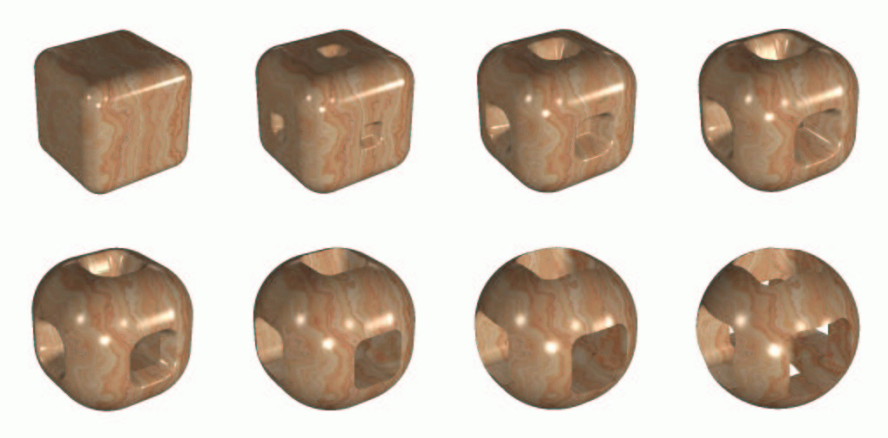

Figure 13: Metamorphosis between a pierced sphere and a rounded box

Figure 14: Metamorphosis between a propeller and a sphere