A Framework for Modeling, Animating and Morphing Textured Implicit Models

Aurélien Barbier, Eric Galin, Samir Akkouche

LIRIS, CNRS, Université Claude Bernard Lyon 1, France

Abstract

This paper presents a framework for modeling, animating and morphing textured implicit models. Our hierarchical skeletal implicit surface model incorporates key-frame animation and procedural solid texturing in a unified and coherent way. Our system enables the designer to create complex special effects by synchronizing shape, animation and texture transformations.

Keywords: *implicit surface modeling, animation, interpolation, shape metamorphosis, texture metamorphosis.*

1. Introduction

Metamorphosis, or morphing, can be defined as the process of smoothly transforming an initial shape into a final shape [7]. Metamorphosis has a vast variety of applications. It has been successfully applied in some modeling systems for generating a variety of models after a few control shapes, and has been extensively used for creating special effects in the entertainment industry.

Morphing techniques are often very complex. Given two shapes, there is an infinity of transformations that create a metamorphosis sequence. 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.

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. Unnecessary deformations or changes in the topology should be avoided. Amorphous or blobby intermediate shapes, which sometimes cannot be avoided in complex morphing sequences, should be limited in time. Therefore, a metamorphosis technique should provide some interactive user-control over the intermediate shapes. Although automating the metamorphosis process may be effective for restricted class of shapes or models, user control proves to be fundamental to produce convincing animations.

1.1. Related work

Metamorphosis techniques are strongly connected to the underlying geometric representation. A complete overview of shape morphing is beyond the scope of this paper. An interesting survey of existing metamorphosis techniques may be found in [8]. The proposed classification exhibits two major categories: triangle mesh morphing methods and volumetric techniques, which further separate into voxel based approaches and implicit surface techniques.

Triangle meshes, which are prominently used in major modeling and animation systems, are difficult to metamorphosize, especially when dealing with surfaces of different topology. A good overview of triangle mesh morphing techniques is presented in [1]. Most mesh metamorphosis techniques first create a graph of correspondences matching the vertices of the two models, possibly merging their topologies, and define the transformation by performing an interpolation between the geometries. The main challenge stems from the fact that methods that operate triangle meshes need to solve both the *vertex correspondence problem* to create an intermediate triangle model that merges the topologies of the argument shapes, and the *vertex path problem* to avoid self intersections and pieces of the surface folding onto themselves during the transformation.

Metamorphosis techniques for voxel models can handle the transformation of shapes with different topologies.Those methods 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.

Implicit surface techniques avoid the computation of a fixed size sampling grid and directly compute the metamorphosis by interpolating the parameters of the field functions representing the objects. Implicit surfaces have proved to be efficient as this representation implicitly deals with changes in the topology. Automatic techniques that have been proposed for general function representation [9] and variational implicit surfaces [14] do not provide a good control over the transformation and often create intermediate shapes that exhibit a loss of shape coherence during the interpolation.

1.2. Contributions

Although a wide range of metamorphosis techniques have been proposed for the past few years, existing methods that operate on three dimensional objects only address the transformation of the geometry of still objects and do not provide simple methods for transforming the texture simultaneously or synchronizing a metamorphosis with animation.

This paper presents a framework for controlling the metamorphosis of two animated and textured models. Our method relies on the BlobTree model [15] that has proved to be particularly efficient for creating and controlling complex transformations [6]. Our method extends and improves that approach in several ways.

In [5, 6], the shape transformation between the skeletal primitives was performed using a linear interpolation based on Minkowski sums. This approach used to limit primitives to polytopes (convex polygonal elements of arbitrary dimension) which both restricted modeling and could produce amorphous intermediate shapes during the transformation.

In this paper, we present a vast variety of shapes including curves, surfaces and volumes such as boxes or cone-spheres that may be used as skeletal elements. Complex skeletal elements extend the range of shapes that may be created. We also propose an original technique for computing the transformation between those skeletons. We also propose a coherent system for texturing the BlobTree by incorporating procedural solid texturing nodes in our model. We present a technique for interpolating texture nodes, which enables us to synchronise texture interpolation with shape transformation. Eventually, we show that the parameterization of the BlobTree by functions of time is a natural way to create complex animations and transformations. This approach encompasses the animation and metamorphosis in a unified and coherent model.

The remainder of this paper is structured as follows. Section 2 presents the fundamentals concepts of the BlobTree. In particular, we detail the different types of nodes involved in the hierarchy and present how textures and animation are incorporated in the BlobTree model. Section 3 presents some complex primitives with levels of detail management that enables us to model complex shapes with a few control parameters. Section 4 presents the TextureTree system which is used to incorporate procedural textures in the BlobTree model. Section 5 presents the metamorphosis of animated models: we focus on shape and animation coherence during the transformation and describe how textures are transformed simultaneously during a morphing sequence. Finally, we conclude this paper with a presentation of examples, followed by a discussion of our results and open problems for future research.

2. The BlobTree

An implicit surface is mathematically defined as the points in space that satisfy the equation $\mathcal{S} = \{\mathbf{p} \in \mathbb{R}^3, f(\mathbf{p}) - T = 0\}$ where $f(\mathbf{p})$ denotes a scalar field function in space and T a threshold value. The BlobTree model [15] is characterized by a hierarchical combination of primitives organized in a tree data-structure. The nodes of the tree include blending, Boolean and warping nodes, whereas the leaves are characterized as skeletal elements.

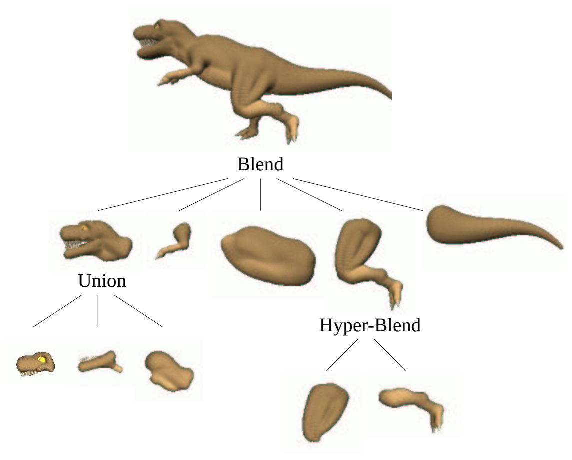

Figure 1: *A simplified representation of the tree structure of a Tyrannosaurus Rex model.*

The computation of the field function $f(\mathbf{p})$ at a given point in space is performed by recursively traversing the BlobTree structure. The skeletal primitives at the leaves of the tree return potential field values, which are combined by the operators at the nodes of the tree.

Skeletal elements The leaves of the BlobTree are skeletal elements that are characterized by a skeleton, a distance function and a potential field function with a compact support. The field functions $f(\mathbf{p})$ are decreasing functions of the distance to a skeleton $f = g \circ d$ where $g : \mathbb{R}_+ \to \mathbb{R}$ is the potential field function, and $d : \mathbb{R}^3 \to \mathbb{R}_+$ refers to the distance to the skeleton. The skeleton and the distance function d characterize the shape of the element, whereas the potential field function g defines the way elements blend together.

The potential field functions have a limited radius of influence, denoted as R. Therefore, every skeletal primitive has a bounded region of influence in space, denoted as Ω, which may be defined by sweeping a sphere of radius R over the skeletons. As the shape of Ω may be complex for some skeletons, every primitive incorporates a bounding box in its data-structure so as to rapidly discard useless field function evaluations when queries are performed in empty regions of space.

Blending nodes Our system implements different types of blending and includes a hyper-blending node. Global blending is defined by summing the field functions of the contributing elements:

$$f_{A+B}(\mathbf{p}) = f_A(\mathbf{p}) + f_B(\mathbf{p})$$

Our system includes a generalized hyper-blending node which is useful for controlling the way elements blend together:

$$f_{A+B}(\mathbf{p}) = (f_A(\mathbf{p})^n + f_B(\mathbf{p})^n)^{1/n} \quad n \in \mathbb{R}_+^*$$

We have adapted the local blending technique described in [11] by implementing a new local blending node. This operator has three children: the first two, denoted as A and B, represent the two BlobTree models that will be partially blended together, whereas the third, denoted as R, represent the region of space where blending will occur. We define the resulting field function as a weighted average between blending and union as follows:

$$f_{A+B\,|\,R} = f_R(\mathbf{p})\;(f_A(\mathbf{p}) + f_B(\mathbf{p})) + (1 - f_R(\mathbf{p}))\;f_{A\cup B}(\mathbf{p})$$

The field function f_R that maps $\mathbb{R}^3$ onto interval $[0,1]$ scales the amount of blending between the two sub-trees A and B.

Boolean nodes Boolean operators are implemented in different ways. The min and max functions prescribed in [15] for union and intersection are the most efficient:

$$f_{A\cup B}(\mathbf{p}) = \max(f_A(\mathbf{p}), f_B(\mathbf{p}))$$

$$f_{A\cap B}(\mathbf{p}) = \min(f_A(\mathbf{p}), f_B(\mathbf{p}))$$

The difference operator, denoted as \, is parameterised by the threshold value: $f_{A\setminus B}(\mathbf{p}) = \min(f_A(\mathbf{p}), 2T - f_B(\mathbf{p}))$.

Those operators produce gradient discontinuities in the potential function, which results in visible unwanted normal discontinuities on the surface. In contrast, R-Functions [9] define a field function with C^n continuity almost everywhere in space so as to avoid gradient discontinuities. We have adapted those functions to our model [3] as follows:

$$\begin{aligned} f_{A\cup B}(\mathbf{p}) = T + \frac{1}{2-\sqrt{2}} \Big[& (f_A(\mathbf{p}) - T) + (f_B(\mathbf{p}) - T) \\ & + \sqrt{(f_A(\mathbf{p}) - T)^2 + (f_B(\mathbf{p}) - T)^2} \Big] \\ f_{A\cap B}(\mathbf{p}) = T + \frac{1}{2+\sqrt{2}} \Big[& (f_A(\mathbf{p}) - T) + (f_B(\mathbf{p}) - T) \\ & - \sqrt{(f_A(\mathbf{p}) - T)^2 + (f_B(\mathbf{p}) - T)^2} \Big] \end{aligned}$$

Both representations produce the same implicit surface if the Boolean nodes are located at the top of the tree structure. In this case, the computation of the min and max is computationally inexpensive compared to R-Functions. In contrast, we use the modified R-Function equations to create a continuously differentiable potential field if blending nodes are located above Boolean operators in the BlobTree. Our system automatically adapts the function used to evaluate Boolean operators depending on the context during the evaluation.

Warping nodes Warping operators implement deformations nodes that distort the shape of the implicit surface by warping space in its neighbourhood. Generally speaking, a warp is a continuous function $\omega(\mathbf{p})$ that maps $\mathbb{R}^3$ onto $\mathbb{R}^3$. The evaluation of the field function is performed as follows:

$$f_{\omega(A)}(\mathbf{p}) = f_A \circ \omega^{-1}(\mathbf{p})$$

Translation and rotation nodes, which will be referred to as frame nodes, deserve special attention as they are involved in the definition of the animation skeleton. As we will see in Section 5, frame nodes may be inserted in the structure of an otherwise still BlobTree model for attaching an animation skeleton to a shape and animating it.

3. Complex skeletal primitives

Our modeling system incorporates a vast variety of skeletal primitives such as line segment, circle, circular arc, triangles, discs, boxes, cylinders or polyhedral shapes (Figure 2). In our framework, every primitive provides an algorithm that computes the Euclidean distance to its skeleton, denoted as $d(\mathbf{p})$. Although the computation of the distance $d(\mathbf{p})$ between a point in space and a simple skeleton such as a point or a line segment is straightforward and computationally efficient, algorithms become the more sophisticated as the complexity of skeletons increase. A complete description of distance computation is beyond the scope of this paper. A general overview on point to primitive distance computation may be found in [12].

In this section, we present some complex primitives, namely generalized cylinders, surface patches with variable thickness and surfaces of revolution with variable thickness. Those complex primitives extend the range of shapes that may be created, are easy to control in our animation and

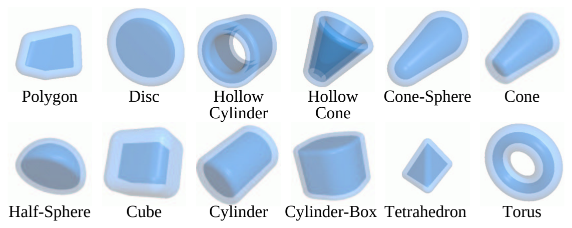

Figure 2: *Some skeletal primitives implemented in our system.*

metamorphosis system as they are defined by a small set of parameters (Figure 3).

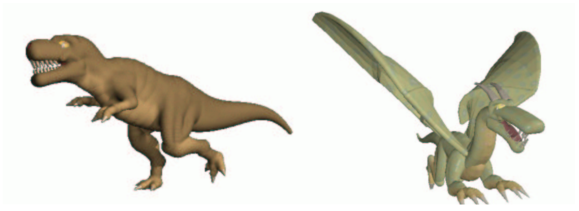

Figure 3: *A Tyrannosaurus-Rex and a Dragon defined with generalized cylinders and surface patches with varying thickness.*

Since computing the distance to those complex skeletal elements is very difficult, we rely on a decomposition and instantiation process that approximates the skeleton by a union of simpler skeletal elements such as spheres, cylinders, triangles or tetrahedral elements (Figure 2) organized in a tree which may be processed efficiently. Using the union operator guarantees that no bulging effect appears. The BlobTree generation process adapts the number of generated primitives to the required level of detail.

3.1. Generalized cylinders

A generalized cylinder primitive is parameterized by a sweeping curve, denoted as $\mathbf{c}(t)$, and a varying radius function, denoted as $r(t)$, along the support curve (Figure 4). The BlobTree generation process converts the curve into a set of n line segments whose vertices are denoted as $\mathbf{v}_i$, $i \in [0,n]$, depending on the required level of detail. For every vertex $\mathbf{v}_i$, we compute the corresponding radius parameter r_i and create a cone-sphere primitive for every line segment $[\mathbf{v}_i\mathbf{v}_{i+1}]$.

The generation process can be optimized as follows. If the end radii r_i and r_{i+1} are the same, the system instantiates a line-swept-sphere primitive which is faster to process. An in depth overview of the evaluation of $d(\mathbf{p})$ for cylinders, cones, line swept spheres and cone-spheres is presented in [2]. If the end radii are null, a line segment primitive is generated. Eventually, the generation process organizes primitives into an optimized hierarchy of union nodes

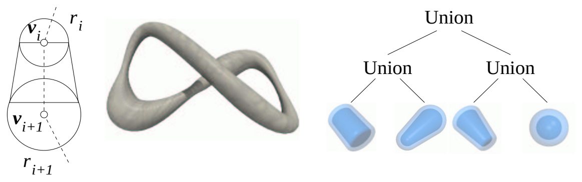

Figure 4: *A generalized cylinder and its corresponding BlobTree representation.*

(Figure 4) so as to take advantage of spatial coherence and reduce computations when performing potential field computation queries.

3.2. Surface with variable thickness

Our system implements different types of surface primitives, such as subdivision surfaces and parametric surface patches. Subdivision surfaces as well as parametric surfaces generate a set of triangles that approximate the surface at a given level of detail. Those triangles are converted into triangle primitives in the BlobTree generation process.

More complex shapes may be created by controlling the thickness of the surface by a parameterized thickness function, denoted as $t(u,v)$, where the tuple (u,v) refers to the parameterization of the surface. The wings of the dragon model (Figure 3) were created with Bézier surface patches with a variable thickness: the end parts of the wing are relatively thin compared to the section near the junction with the body which is thicker.

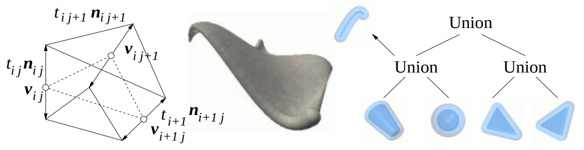
$t_{i\,j+1}\,\mathbf{n}_{i\,j+1}$
$\mathbf{v}_{i\,j+1}$
$t_{ij}\,\mathbf{n}_{ij}$
$\mathbf{v}_{ij}$
$t_{i+1\,j}\,\mathbf{n}_{i+1\,j}$
$\mathbf{v}_{i+1\,j}$
Union
Union
Union

Figure 5: *A surface patch with varying thickness and its corresponding BlobTree representation.*

The BlobTree generation process proceeds as follows (Figure 5). First, we create a triangle mesh of the surface. Then, for every vertex $\mathbf{v}_{ij}$, we compute the thickness parameter t_{ij} and the vertex normal $\mathbf{n}_{ij}$ by averaging the normals of the neighbouring triangles. Finally, for every triangle, we create a polyhedral primitive. The vertices of this primitive are computed by offsetting the three vertices of the triangle in the direction $\pm t_{ij}\,\mathbf{n}_{ij}$. The borders of the surface generate generalized cylinders as detailed earlier.

As for generalized cylinders, the generation process is optimized to create simpler primitives whose distance functions are faster to evaluate whenever possible. If the thickness at the vertices of the base triangle are the same, then the offset polyhedron has parallel faces and the general polyhedron skeletal element is replaced by another specific prism skeletal element. Triangles are generated if the thickness function is null. Finally, two neighbouring prisms with coplanar generating triangles and the same thickness parameters may be merged into a simple box primitive, which further simplifies to a rectangle if the thickness is null.

3.3. Surface of revolution with variable thickness

Surfaces of revolution are generated by sweeping a two dimensional curve surface around an axis. In our implementation, we can add some thickness to the surface of revolution by offsetting the generating curve by a variable thickness parameter. The models in Figure 6 were created using this technique.

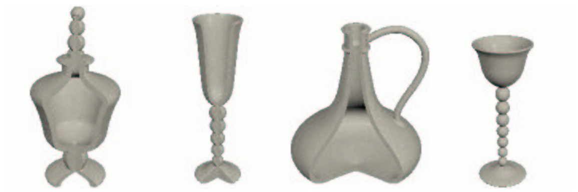
Figure 6: *Models created with surface of revolution primitives with varying thickness.*

The BlobTree generation process is similar to generalized cylinders. First, the system converts the profile curve into a set of n line segments whose vertices are denoted as $\mathbf{v}_i$, $i \in [0,n]$. For every vertex $\mathbf{v}_i$, the system computes the curve normal denoted as $\mathbf{n}_i$ and evaluates the corresponding thickness parameter t_i (Figure 7).

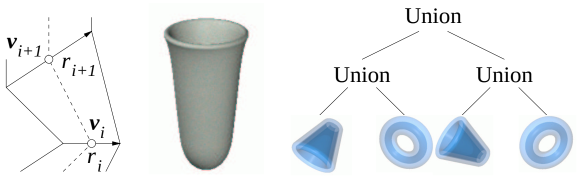

Figure 7: *A surface of revolution and its corresponding BlobTree representation.*

Then, for every line segment, the vertices of the four sided polygon $\mathbf{v}_i \pm t_i\,\mathbf{n}_i$ and $\mathbf{v}_{i+1} \pm t_{i+1}\,\mathbf{n}_{i+1}$ are computed. We rely on single primitive of revolution which is a hollow cone of varying radii. The system also generates torus primitives with center $\mathbf{v}_i$, major radius R_i that refers to the distance between $\mathbf{v}_i$ and the axis, and minor radius t_i to create smooth junctions between hollow cones. The final shape is defined as the union of those primitives.

As before, several optimization steps have been incorporated in this process. If the thickness parameter is null, we instantiate simpler hollow cone primitives. If the end radii are the same, we create the corresponding cylinder primitives.

4. Texturing the BlobTree

Texturing implicit surfaces is a challenging problem in computer graphics. Several techniques have been proposed for mapping a two dimensional texture image onto an implicit surface by following the stream lines of the gradient ∇f in space [13]. Those methods are slow, stretch and deform textures and do not provide a good control. Function representations may incorporate textures in a more natural way [**?**], but are not suitable for our animation and morphing purposes.

Our implementation of the BlobTree model incorporates texture nodes that assign a material, implemented as a procedural solid texture, to a skeletal primitive or to a whole subtree. Solid textures are defined by the TextureTree which is characterized by a hierarchical combination of textured regions organized in a tree data-structure. In our system, the textured regions are defined by the same skeletal primitives as for the BlobTree. The very difference is that those primitives include texture and material parameters. Boolean and blending operators combine textures together, producing different texturing effects. Blending nodes smoothly blend materials, whereas Boolean operators create sharp transitions between two materials. Warping nodes deform both the regions of influence and their corresponding solid texture.

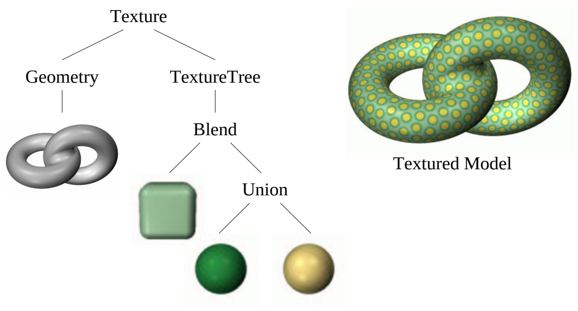

Figure 8: *Controlling the texture of a simple shape with the TextureTree*

Every node in the TextureTree returns two different kinds of information: a potential field value $f(\mathbf{p})$ and the material parameters $\Phi(\mathbf{p})$. The potential field values $f(\mathbf{p})$ are computed the same way as for the BlobTree model and are used to weight the characteristics of the different materials to create blending effects as presented in Section 4.2. A textured object is created by embedding its geometrical representation in the texture space defined by the TextureTree (Figure 4). The evaluation of the texture function at a given point in space is further detailed in the next paragraphs.

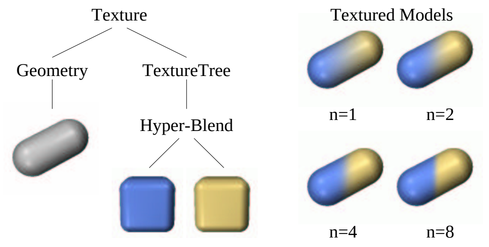

Figure 9: *Blending textures with different parameter: the transition between colors sharpens as n increases.*

4.1. Materials

Materials are implemented as a set of classes of solid textures [4] that are characterized by a parametrized procedural function, denoted as $\Phi(\mathbf{p})$, $\mathbf{p} \in \mathbf{R}^3$. The function $\Phi(\mathbf{p})$ defines the characteristics of a material for any point $\mathbf{p}$ in space. Material parameters include the color, the reflection and transmittance coefficients denoted as ρ and τ respectively, and the index of refraction of the material σ. Other parameters may be incorporated in the definition of the material, depending on the requirements of the illumination model.

Although the texturing function Φ may be of any type, we rely on a few classes of parameterized solid textures such as marble, wood, or granite [4]. As we will see in Section **??**, those classes will enable us to create complex special effects such as texture animation and metamorphosis easily.

4.2. Overview of the texturing algorithm

The evaluation of the texturing function in space is achieved by recursively traversing the TextureTree, either evaluating the solid texture functions at its leaves, or combining the characteristics of the materials returned by the children nodes with Boolean, blending or warping nodes. The algorithm may be outlined as follows:

1. If the node is a skeletal primitive, return the evaluation of the solid texture function $\Phi(\mathbf{p})$ and the potential field value $f(\mathbf{p})$ at point $\mathbf{p}$.
2. If the node is a Boolean or blending node, evaluate the texturing functions for its children nodes $\Phi_i(\mathbf{p})$ and combine those parameters with the field function values of its children $f_i(\mathbf{p})$, depending on the type of the node.
3. If the node is a warping node, evaluate the texturing functions of its child node with the point transformed in warped space $\omega^{-1}(\mathbf{p})$.

Blending nodes The blending operator weights the parameters of the materials of its children A and B by computing their weigted sum:

$$\Phi_{A+B}(\mathbf{p}) = \frac{f_A(\mathbf{p})\Phi_A(\mathbf{p}) + f_B(\mathbf{p})\Phi_B(\mathbf{p})}{f_A(\mathbf{p}) + f_B(\mathbf{p})}$$

Blending produces a smooth transition between several neigbhooring materials according to the regions of influence and to the field function values returned by the child nodes. Blending may be generalized to hyper-blending as follows:

$$\Phi_{A+B}(\mathbf{p}) = \frac{\left(f_A(\mathbf{p})^n\Phi_A(\mathbf{p})^n + f_B(\mathbf{p})^n\Phi_B(\mathbf{p})^n\right)^{1/n}}{\left(f_A(\mathbf{p})^n + f_B(\mathbf{p})^n\right)^{1/n}} \quad n \in \mathbf{R}_+^*$$

When n varies within the interval $[1, +\infty[$, argument textures are smoothly mixed from smooth blending to sharp union (Figure 9). Our system incorporates a local blending node in the TextureTree as well:

$$\Phi_{A+B\,|\,R} = f_R(\mathbf{p})\,\Phi_{A+B} + (1 - f_R(\mathbf{p}))\,\Phi_{A\cup B}$$

Boolean nodes Boolean operators create a sharp transition between two materials. The texturing functions for union, intersection and difference nodes are defined by comparing the field function values of its children:

$$\begin{aligned}
\Phi_{A\cup B}(\mathbf{p}) &= \begin{cases} \Phi_A(\mathbf{p}) & \text{if} \quad f_A(\mathbf{p}) > f_B(\mathbf{p}) \\ \Phi_B(\mathbf{p}) & \text{otherwise} \end{cases} \\
\Phi_{A\cap B}(\mathbf{p}) &= \begin{cases} \Phi_A(\mathbf{p}) & \text{if} \quad f_A(\mathbf{p}) < f_B(\mathbf{p}) \\ \Phi_B(\mathbf{p}) & \text{otherwise} \end{cases} \\
\Phi_{A\setminus B}(\mathbf{p}) &= \begin{cases} \Phi_A(\mathbf{p}) & \text{if} \quad f_A(\mathbf{p}) < 2T - f_B(\mathbf{p}) \\ \Phi_B(\mathbf{p}) & \text{otherwise} \end{cases}
\end{aligned}$$

Warping nodes Warping operators distort geometrical shapes and textures by warping space. The texturing function is defined as:

$$\Phi_{\omega(A)}(\mathbf{p}) = \Phi_A \circ \omega^{-1}(\mathbf{p})$$

By evaluating texture parameters in warped space, the designer can achieve various interesting special effects. Frame nodes also influence the evaluation of the TextureTree, which makes it possible to synchronize the animation of the textures with the animation of the geometry.

5. Metamorphosis

In this section, we present the animorphosis of the BlobTree that aims at creating a smooth animated metamorphosis between two freely animated 3D-objects. This problem is more complex than the metamorphosis between still shapes. In general, a metamorphosis between two still objects is visually appealing and convincing if shape coherence is preserved during the transformation. Both shape and animation coherence should be preserved. As illustrated in Figure16, the transformation between a walking Tyrannosaurus-Rex and a flying Dragon should not only create smooth interpolating shapes. The animation of the legs of the Tyrannosaurus-Rex should smoothly transform into the slow balancing movement of the legs of the Dragon. The wings should not only progressively grow from its back but should also start fluttering with an increasing amplitude as the Dragon takes-off.

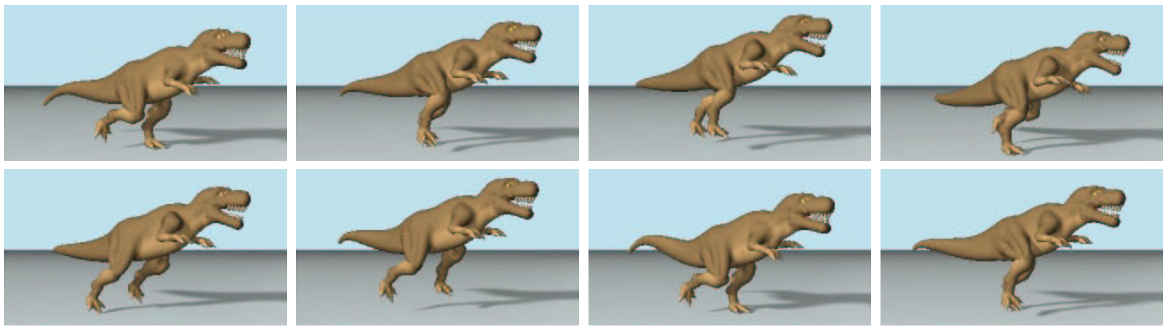

Figure 10: *A walking Tyrannosaurus-Rex.*

5.1. Animating the BlobTree

We animate the BlobTree model by defining parameters of its nodes as functions of time. Therefore, the animated BlobTree is characterized by a hierarchical generic tree data-structure, denoted as A, and a set of time varying parameters, denoted as $\mathcal{P}_A(t)$ (Figure 11). In this section, $A(t) = \{A, \mathcal{P}_A(t)\}$ will refer to animated BlobTree.

The set of parameters $\mathcal{P}_A(t)$ includes the global time varying threshold parameter, denoted as $T_A(t)$, as well as the set of time varying parameters of the nodes $A_i(t)$ of the BlobTree, denoted as $\mathcal{P}_{A_i}(t)$. Let $p_{A_{i,j}}(t)$ denote the j^{th} time varying parameter of the i^{th} node of the BlobTree. Every parameter $p_{A_{i,j}}(t)$ has its own evolution function.

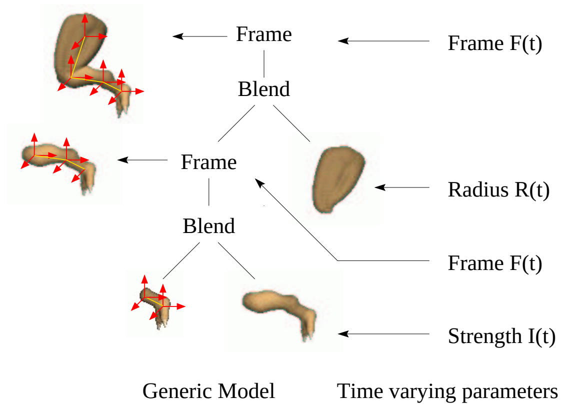

Figure 11: *The animated BlobTree model of the leg of the Tyrannosaurus-Rex character.*

Separating the animated BlobTree model into a generic structure and a set of time varying parameters provides us with a coherent framework for animating, deforming and metamorphosizing shapes. The animation, deformation and metamorphosis of our model will be performed by creating instances of the generic BlobTree structure A using the parameters $\mathcal{P}_A(t)$ at a given time step t. The creation of those instances is achieved by recursively traversing the nodes of the generic BlobTree structure and computing the values of the parameters of the nodes with the time varying parameters $p_{A_{i,j}}(t)$.

Key-frame animation In our system, key-frame animation is performed by attaching an animation skeleton to the BlobTree. This process can be performed easily by inserting frame nodes, *i.e.* rotation and translation nodes, in the BlobTree structure. Frame nodes are functions of time which define the movement of the animation skeleton. Figure 11 illustrates the frame nodes that were inserted in the definition of the leg of the Tyrannosaurus Rex model. The frames in Figure 12 outline the animation skeletons of the Tyrannosaurus-Rex and Dragon models which include 21 and 49 frame nodes respectively.

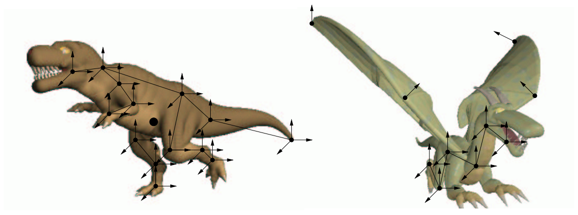

Figure 12: *Animation skeletons: dots outline sets of skeletal primitives deformed with time varying parameters (breathing and vertical tail movements) whereas frames outline the main articulations of the models.*

The time varying parameters $p_{A_{i,j}}(t)$ are characterized by piecewise Hermite spline curves. At a given time step t_k, a parameter $p_{A_{i,j}}(t_k)$ is characterized by a column vector of its successive time derivatives. The value of $p_{A_{i,j}}(t)$ over the interval $[t_k, t_{k+1}]$ is given by the interpolation of Hermite-Ferguson of degree n which is a polynomial of degree $2n-1$.

Deformations Shapes can be deformed either by modifying the parameters of their warping nodes, or by directly changing the parameters of some skeletal elements in the BlobTree.

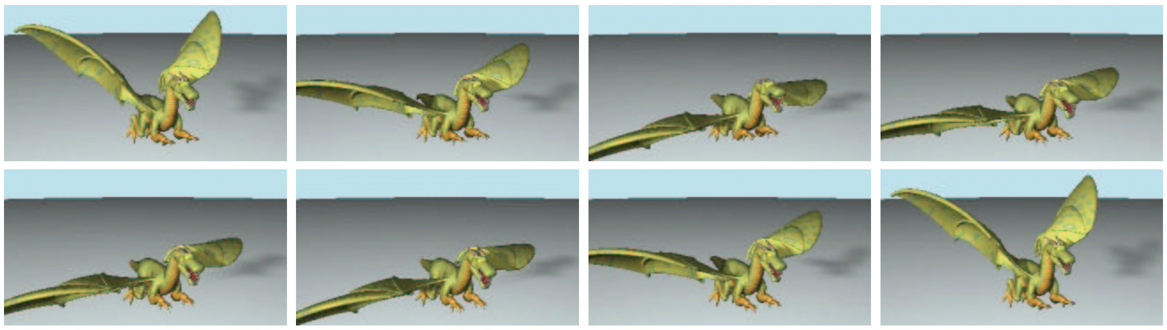

Figure 13: *A flying Dragon.*

Figure 11 shows that the intensity and the radius parameters of some skeletal elements of the leg were defined as functions of time to create a bulging effect during the animation. The pulsating belly was controlled by changing the radius of influence of the sphere primitives through time to fake breathing (Figure 12). The movement of the tail was directly controlled by parameterizing the end vertices of the generalized cylinders with periodic functions of time.

Application Figure 10 shows a walking Tyrannosaurus-Rex controlled by a key-framing animation system. This model has been created with 66 generalized cylinders combined by local blending operators, and incorporates 21 frame nodes for animation. The final instantiated models has 1835 skeletal elements (mostly cylinders and cone-spheres) and 201 Boolean and blending. The animation of the tail is controlled by modifying the control points of the generalized cylinders. It combines a rotation frame node that creates a left to right balancing movement synchronized with the hips.

Figure 13 shows a flying Dragon made with 43 generalized cylinders and 12 surface patches. The animation is controlled by 21 frame nodes and 30 control points for animating the tail and the wings.

5.2. General algorithm

Let $A(t) = \{A, \mathcal{P}_A(t)\}$ and $B(t) = \{B, \mathcal{P}_B(t)\}$ denote the source and target models. We characterize the transformation by a new *generic* animated BlobTree model $C(t) = \{C, \mathcal{P}_C(t)\}$. The generic structure C is computed by invoking the original BlobTree metamorphosis algorithm [6] between the corresponding generic models A and B. This generic structure will characterize the whole transformation. The time varying parameters $\mathcal{P}_C(t)$ interpolate the parameters $\mathcal{P}_A(0)$ and $\mathcal{P}_B(1)$. This last step is the most difficult, since a tight control is required to preserve both shape and animation coherence. The overall algorithm may be outlined as follows :

1. Define a graph of correspondences $\mathcal{G}_C(A \rightarrow B)$ matching two models A and B.
2. Create the generic structure C from the bijectively matching graph derived from $\mathcal{G}_C(A \rightarrow B)$.
3. Define the set of time varying parameter $\mathcal{P}_C(t)$ by interpolating the corresponding parent parameters $\mathcal{P}_A(t)$ and $\mathcal{P}_B(t)$.

Creation of the generic tree structure The first two steps of the algorithm are performed as presented in [6]. Given the generic BlobTree structures A and B, we aim at creating two new overlapping models, denoted as A' and B', whose nodes and leaves can be bijectively paired. This involves the creation of a graph of correspondences compatible with the tree structure of both the source and the target animated models (Figure 14).

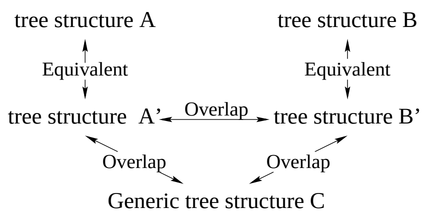

Figure 14: *Generic tree structure creation process*

Starting from the roots of the models, we iteratively descend down the tree structures and create a graph of correspondences between the nodes at the same level. Whenever a node of A holds several correspondence links, it is split into sub-nodes and the BlobTree A is updated into an equivalent form A'. The same process is simultaneously performed on the nodes of B. The correspondence process ends at the leaves of either structure. Multiply matched skeletal elements are processed as described in [5, 6]. The parameters at the blending, Boolean, warping and texturing nodes $\mathcal{P}_{A'}(t)$ are defined as copies of their parent parameters $\mathcal{P}_A(t)$. In contrast the parameters of the skeletal elements are derived from $\mathcal{P}_A(t)$ as described in [5]. This correspondence process

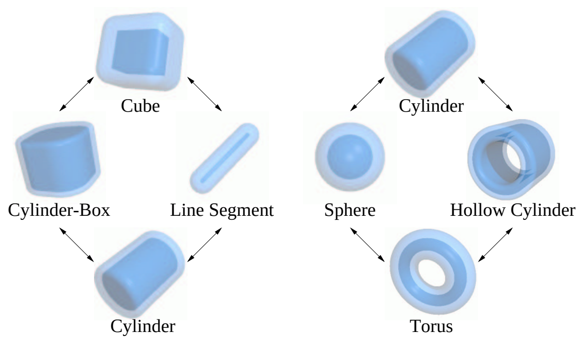

Figure 15: *Two interpolation graphs between some skeletal elements.*

may be either controlled by the animator or achieved automatically through heuristics.

Skeletal primitive transformation In [5, 6], the transformation between skeletal primitives was performed using a linear interpolation based on Minkowski sums. This approach limited primitives to polytopes (convex polygonal elements of any dimension). The Minkowski sum can no longer be used in the general case as the resulting shapes become extremely complex which makes distance evaluation computationally prohibitive or even impossible.

To overcome this problem and find a skeletal transformation for our set of primitives, we propose to rely on a graph of skeletal elements whose arcs define the possible transformations between the nodes. An arc exists between two skeletal shapes $\mathcal{S}_A$ and $\mathcal{S}_B$ if the geometrical parameters of $\mathcal{S}_A$ may be set so that its geometry degenerates into the geometry and topology of $\mathcal{S}_B$.

Figure 15 illustrates this concept by representing two interpolating graphs for a restricted set of skeletons. The graphs implement two different ways of transforming a cylinder into a box and a torus into a cylinder respectively. There exists an arc between a cylinder and a segment since the cylinder may degenerate into a segment by decreasing its radius parameter to 0. The transformation between a cylinder and a box can be achieved either using a deforming cylinder-box primitive, or by first transforming the cylinder into line segment and further transforming this line segment into a box, which produces simpler intermediate shapes.

In our system, given an initial and a final shape, we automatically identify the intermediate shapes needed to create a smooth transformation by finding a path in the interpolation graph connecting the two shapes. The arcs of the graph are valuated by weights that characterize the complexity of the interpolating shape. In general, the complexity of a shape directly relates to its dimension and the computational cost of its distance function $d(\mathbf{p})$. Therefore, we automatically search for the path with a minimal cost so as to avoid complex intermediate skeletons with a high dimension that tend to produce amorphous intermediate or blobby shapes as demonstrated in [6].

Controlling the generic parameters The last step of the algorithm controls the way parameters change throughout time. The animated model $C(t)$ should not only interpolate the initial and final shapes $A(0)$ and $B(1)$ but also produce a visually coherent interpolation between the animations of $A(t)$ and $B(t)$: the animation of $A(t)$ should smoothly disappear while the animation of $B(t)$ takes place. Therefore, we are confronted with both shape and animation coherence.

In the following paragraphs, the time varying parameters $p_{A_{i,j}}(t)$ and $p_{B_{i,j}}(t)$ of the nodes A_i and B_j will be denoted as $p_A(t)$ and $p_B(t)$ for the sake of clarity. Without loss of generality, we will assume that the transformation takes place over the interval of time $[0,1]$. In our approach, we interpolate the parameters $p_A(t)$ and $p_B(t)$ by using an evolution function denoted as $s(t)$ that weights the influence of the parent parameters in time:

$$p_C(t) = (1 - s(t))\, p_A(t) + s(t)\, p_B(t)$$

The evolution function $s(t)$ characterizes the speed of transformation between two parameters. In our implementation, we use piecewise cubic Hermite Splines that provide a simple intuitive control over the transformation. Figure 17 shows the control curve of the interpolation of the height parameter of the Tyrannosaurus-Rex and the Dragon. This curve constrains the model to the floor at the begining of the transformation and raises the creature in the air when the wings have grown enough.

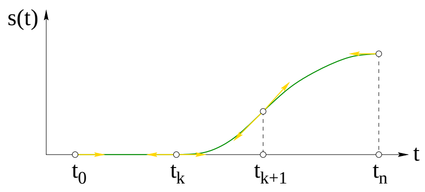

Figure 17: *Controlling the height of the trajectory of the transforming Tyrannosaurus Rex model with a piecewise interpolation function $s(t)$.*

Thanks to the hierarchical structure of the BlobTree, only a few parameters need to be interpolated to create complex transformations. Still, every parameter in the generic animated BlobTree model $C(t)$ may be tuned independently with its own evolution function if need be.

5.3. Key-frame metamorphosis

In our system, animations are controlled by key-framing. Therefore, we have developped an automatic technique that creates a new key-frame animation given two input key-frame animation models. Recall that we use Hermite spline

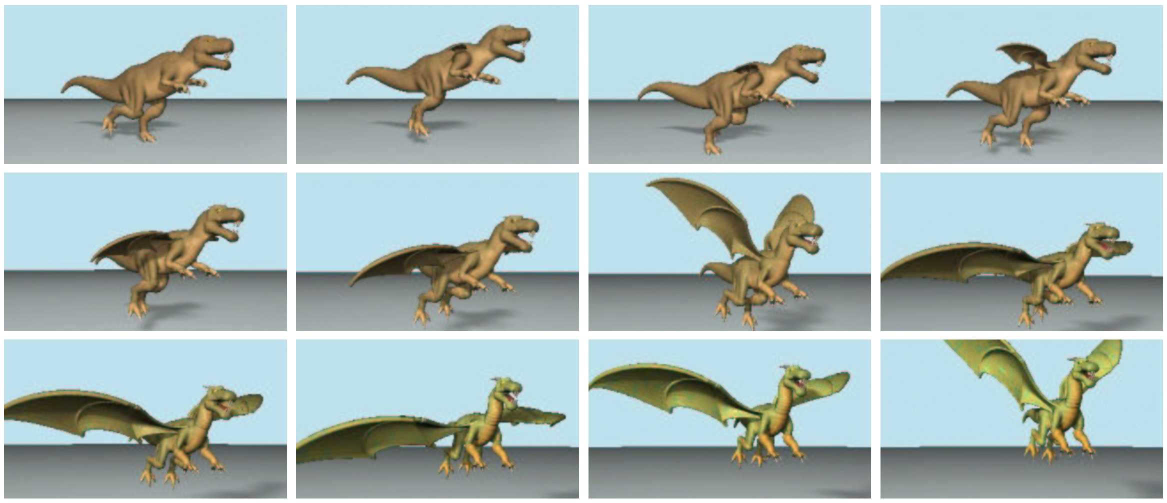

Figure 16: *Metamorphosis between a walking Tyrannosaurus-Rex and a flying Dragon. The back legs of the creature progressively stop moving while the wings grow and start fluttering as the creature takes off.*

curves to define the evolution functions $s(t)$. Therefore, since all time varying parameters are defined by Hermite Splines, we can constrain the computation of the parameters $p_C(t)$ so that animorphosis should produce a consistent animation model. Our goal is to define the intermediate shape as a simple key-frame animated BlobTree rather than an interpolation of two animated BlobTrees at each time step.

Given an interpolation function $s(t)$ and two key-frame parameter functions $p_A(t)$ and $p_B(t)$, the following algorithm automatically computes the interpolating key-frame parameter function $p_{C(t)}$ (Figure 18):

1. For every time interval $[t_k, t_{k+1}]$ of the interpolation function $s(t)$ defined by the animator, compute the set $\{t_i\}$ of the time steps of $p_A(t)$ and $p_B(t)$ in $[t_k, t_{k+1}]$.
2. For every time step t_i, define the key steps of $p_C(t_i)$ as follows:

$$p_C(t_i) = (1 - s(t_i))\,p_A(t_i) + s(t_i)\,p_B(t_i)$$

   The key steps of $p_C(t)$ at the end times of the control interval $[t_k, t_{k+1}]$ are obtained by plugging t_k and t_{k+1} into the previous equation, thus:

$$p_C(t_k) = (1 - s(t_k))\,p_A(t_k) + s(t_k)\,p_B(t_k)$$

$$p_C(t_{k+1}) = (1 - s(t_{k+1}))\,p_A(t_{k+1}) + s(t_{k+1})\,p_B(t_{k+1})$$

3. Evaluate the evolution function of $p_C(t)$ using the interpolation of Hermite-Ferguson of the new key-steps $\{p_C(t_i)\}$.

Let n_A, n_B and n_S denote the number of control knots in the definition of $p_A(t)$, $p_B(t)$ and $s(t)$ respectively. Our technique creates at most $n_A + n_B + n_s$ key-frames for the parameter $p_C(t)$. This approach preserves the original control

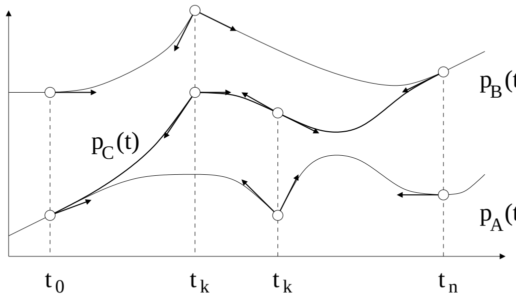

Figure 18: *Merging two time varying key-framed parameters $p_A(t)$ and $p_B(t)$ into an interpolating parameter $p_C(t)$.*

points of the argument animations. Therefore, our method defines the animation of the intermediate shape by a classical key-frame description and takes into account each time step of both the source and target models, which preserves both shape and animation coherence. Extra control points may be inserted to introduce specific steps in the transformation if necessary.

Application Figure 16 shows the transformation between a walking Tyrannosaurus-Rex and a flying Dragon. The overall hierachical graph of correspondence between the two models is relatively simple since the initial and final shapes have the same overall structure. The metamorphosis between the heads of the two models required a tighter control. The skulls and the jaws were finely tuned and controlled independently to avoid unwanted blending during their metamorphosis. It took us 2 hours to create the graph of correspondence between the two models. Much time was spent finely matching the jaws and the different parts of the skulls to ensure shape coherence.

During the transformation, the animated wings grow faster than other components so that the Tyrannosaurus-Rex should take-off only when the wings get large enough. Another 4 hours were necessary to tune the important interpolation parameters and tune the animation.

5.4. Texture metamorphosis

We compute the metamorphosis between two TextureTree models A and B by defining a new generic TextureTree model C whose nodes and leaves will be defined by time varying parameters. Since our TextureTree model inherits its structure from the BlobTree model, our approach is strongly based on the original BlobTree metamorphosis algorithm proposed in [6].

Let us recall the key steps of the method. The transformation between two TextureTree models is defined by a single generic model, whose nodes and leaves parameters change throughout time. The skeletal primitives are transformed by interpolating their skeleton and field function as prescribed in [6].

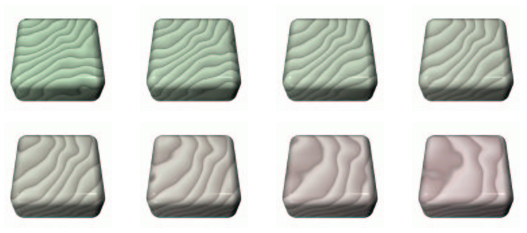
Figure 20: *Metamorphosis between two marble textures: interpolating the parameters preserving the overall texture coherence.*

The metamorphosis of solid textures deserves special attention. If both solid textures A and B are of the same type, *e.g.* both represent marble but with different veining and colors, they share the same parameters. Therefore, the generic texture $C(t)$ is defined as a time varying texture of the same type with interpolating parameters denoted as $\mathcal{P}_C(t)$. Let $s(t)$ denote the speed of the transformation between the parameters, we used:

$$\mathcal{P}_C(t) = (1 - s(t))\mathcal{P}_A + s(t)\mathcal{P}_B$$

As illustrated in Figure 5.4, this approach preserves texture coherence during the transformation.

In the general case however, argument textures may be of different type, *e.g.* a marble pattern may be transformed into a wood pattern. Therefore, they do not share the same parameters and the previous method can no longer apply.

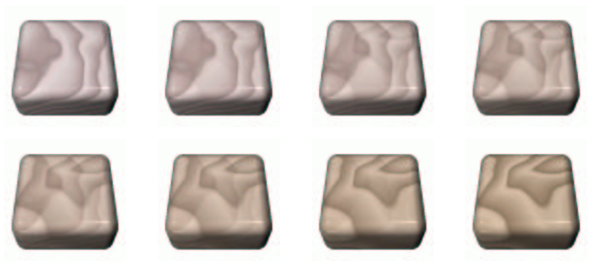
Figure 21: *The direct interpolation of the texture functions creates a non-natural cross-dissolving effect.*

In those cases, the metamorphosis between two solid textures is directly achieved by interpolating their corresponding procedural functions Φ_A and Φ_B. Let $s(t)$ denote the speed of the transformation between the two textures. The time varying function $\Phi_C(\mathbf{p},t)$ representing the transforming texture is directly defined as follows:

$$\Phi_C(\mathbf{p},t) = (1 - s(t))\Phi_A(\mathbf{p}) + s(t)\Phi_B(\mathbf{p})$$

Although general, this approach generally creates cross dissolving effects as illustrated in Figure 21.

Application Figure 19 represents the awakening of a petrified gargoyle into a living creature. Although the geometric model of the gargoyle is rather complex, the Texture-Tree model of the stone material is simply defined as a set of blended spheres that hold the stone pattern parameters. The de-petrifaction effect is achieved by slowly propagating flesh texturing spheres through the geometric model, while moving stone texturing spheres away.

6. Conclusion

In this paper, we have presented a general framework for creating and controlling metamorphoses between textured and animated implicit models. Our animated BlobTree model encompasses animation and metamorphosis in a unified and coherent fashion. Our metamorphosis system generates convincing and visually appealing transformations between complex objects while preserving both shape and animation coherence.

The creation of a generic model is fundamental: genericity enables us to use the same accelerated techniques to ray-trace or polygonize any instance of the animated BlobTree. Moreover, the computation time needed to create all the instances through time is very fast and negligible compared to the rendering time. More than 2000 complex objects may be created in less than 1 second on a Pentium 4 $2,4\,GHz$. In comparison, the polygonization of an instantiated model with a few thousands of primitives takes 10 to 20 seconds.

In this context, the use of level of details in the design of

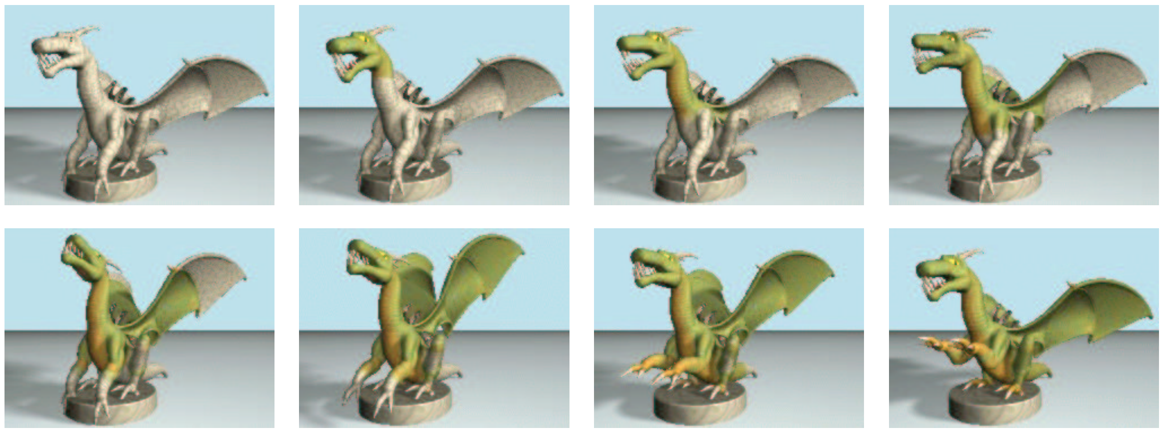

Figure 19: *The gargoyle progressively starts moving as stone slowly turns to flesh.*

complex BlobTrees could speed up rendering. Specific polygonization techniques that take advantage of temporal coherence will be needed in the creation of an interactive editor.

References

[1] M. Alexa. Recent Advances in Mesh Morphing. *Computer Graphics Forum*, **21**(2), 173–196, 2002.

[2] A. Barbier and E. Galin. Fast distance computation between a point and cylinders, cones, line-swept-spheres and cone-spheres. *Journal of Graphic Tools*, **9**(2), 31–39, 2004.

[3] A. Barbier, E. Galin and S. Akkouche. Complex Skeletal Implicit Surfaces with Levels of Detail. *Proceedings of WSCG*, **12**(4), 35–42, 2004.

[4] D. Ebert, F. Musgrave, D. Peachey, K. Perlin and S. Worley. Texturing and Modeling - A Procedural Approach. *AP Professional*, 1998.

[5] E. Galin and S. Akkouche. Soft Object Metamorphosis based on Minkowski sums. *Computer Graphics Forum (Eurographics'96 Proceedings)*, **15**(3), 143–153, 1996.

[6] E. Galin, A. Leclercq and S. Akkouche. Morphing the BlobTree. *Computer Graphic Forum*, **19**(4), 257–270, 2000.

[7] J. Gomes, L. Darsa, B. Costa and L. Velho. Warping and Morphing of Graphical Objects. *Morgan Kaufmann*, 1998.

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

[9] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, **11**(8), 429–446, 1995.

[10] A. Pasko,V. Adzhiev,B. Schmitt and C. Schlick. Constructive Hypervolume Modeling. *Graphical Models*, **63**(6), 413–442, 2001.

[11] G. Pasko, A. Pasko, M. Ikeda and T. Kunii. Bounded Blending Operations. *Proceedings of Shape Modeling International*, 95–104, 2002.

[12] P. Schneider and D. Eberly. Geometric Tools for Computer Graphics. *Morgan Kaufmann*, 2003.

[13] M. Tigges and B. Wyvill. Recent Advances in Mesh Morphing. A Field Interpolated Texture Mapping Algorithm for Skeletal Implicit Surfaces. *Computer Graphics International*, 1999.

[14] G. Turk and J. O'Brien. Shape Transformation Using Variational Implicit Functions. *Computer Graphics (Siggraph'99 Proceedings)*, **35**, 335–342, 1999.

[15] 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.