A Hierarchical Skeleton-based Implicit Model

MARCELO DE GOMENSORO MALHEIROS
WU, SHIN-TING

Grupo de Computação de Imagens (GCI-DCA-FEEC)
Universidade Estadual de Campinas (UNICAMP)
{malheiro,ting}@dca.fee.unicamp.br

Abstract. Implicit modeling is a recent trend in Computer Graphics and a variety of skeleton-based models has been proposed. In this paper we present a hierarchical framework that can encompass a wide range of these variants, providing a more flexible way for defining the shape of implicit objects. We also derive a new decay function, which is computationally cheap.

1 Introduction

Implicit modeling is a recent trend in the field of Computer Graphics that builds geometrical objects on the basis of implicit mathematical functions.

We are interested in a specific modeling technique, where the implicit objects are defined by *skeletons*. Informally, a skeleton is a set of primitives that are combined to define the shape of an object.

The skeleton-based approach provides a very flexible way to construct implicit objects, which can be altered by simply changing their primitives. It can also be used in complement with parametric methods to enrich the modeling domain. Particularly, skeletons are attractive to model a wide range of real objects, mainly organic-like shapes (for example, parts of the human body or animals) and liquids [7]. These models can also be directly animated [8].

The most widely used techniques based on the skeleton approach are *blobs* [1], *metaballs* [6], and *soft objects* [7, 8]. With these techniques, objects are specified in terms of distance measures to a set of points. These techniques are all similar, and differ on the distance function used. Several variants of these techniques were proposed in order to improve either the representativity or the controllability of the skeleton-based models [3, 5]. Since each of these models offer some distinctive features, it is desirable to have their functionalities integrated in order to facilitate the modeling of complex shapes.

In this paper we propose a framework that can encompass a wide range of skeleton-based models.

A skeleton is defined hierarchically using an n-ary tree structure, where the leaf nodes represent any primitive and the internal nodes denote intermediate results from a combination of their child nodes. Syntactically, there is no restriction on either the kinds of primitives or the types of combining operations. For our shape specification, we consider the definition of an implicit object as the composition of three geometrically meaningful classes of functions—*shape*, *decay*, and *blending* functions.

This paper is organized as follows. We begin with the basic concepts and notations used to describe skeleton-based implicit objects, then we discuss the related previous work. Next we present our proposed hierarchical model. Then we describe our implementation and some results. We conclude with a discussion of future work.

2 Skeleton-based Modeling

This section summarizes some basic concepts related to skeleton-based modeling. For further details, we suggest the reference [5].

In the following discussion, we denote points by boldface lower case letters, transformations by boldface capitals and scalars by italics. Subscripts denote members of a set of objects, while superscripts denote particular levels in a hierarchy, being 0 the topmost one. For example, $\mathbf{T}\mathbf{x}$ corresponds to the transformation $\mathbf{T}$ of the point $\mathbf{x}$, and $f_i^{(n)}$ is the i-th scalar function belonging to the n-th depth level.

Given a scalar function $F : \mathbf{R}^m \rightarrow \mathbf{R}$, which maps m-dimensional points to real values, and a real value c, the set of points $\mathbf{x} \in \mathbf{R}^m$ that satisfy

$$F(\mathbf{x}) = c$$

is said to be an implicit object. In fact, this point set corresponds to the level set of F associated to c.

Generally, skeleton-based models are built from a set of scalar functions $f_i(\mathbf{x})$, combined through well-defined operations known as *blends*. The resulting function F defines a *main scalar field* in $\mathbf{R}^m$. Once defined the value c, the set of scalar functions is called the *skeleton* of the implicit object.

For practical purposes, the functions used to define the main scalar field should satisfy some smoothness

conditions, like being continuously differentiable. Moreover, these functions typically have the following property: their values decrease as the distance to a given reference point $\mathbf{p}_i$ increases. These functions define fields with concentric level sets. Therefore, we may say that each of them contributes with "spherical-shaped fields" to the main scalar field. Because of the way the functions were defined, their influence on the resulting field is mostly concentrated in the neighborhood of $\mathbf{p}_i$.

The usual blending operation employed to combine those functions is a weighted sum. Note that if the functions are carefully chosen, then the resulting object is bounded.

3 Previous Work

Skeleton-based implicit modeling was introduced by Blinn [1], motivated by the need to display electron density maps of molecular structures. He used a point set as skeleton, summing the contributions due to a Gaussian distance function for each point.

A variation of the previous model was later developed in [6], using a piecewise quadratic polynomial instead of an exponential function. Differently from Blinn's model, the distance function was defined in such a way that the contribution of a given point dropped to zero after a certain distance. This allowed more efficient computation of the scalar field because only the contributions of the points within a predefined distance were needed.

Wyvill *et al.* [7] proposed a similar model, with a cubic polynomial over the Euclidean distance. There was an explicit parameter that controlled the radius of influence of each point.

These works were limited to point sets and linear blends. Later, Bloomenthal and Shoemake [3] proposed curve segments as references for the distance function (*generalized cylinders*), and a convolution blend to combine them. An extensive survey of skeleton primitives and blending techniques can be found in [5].

Bloomenthal and Wyvill [2] discussed techniques to improve the efficiency of display methods for implicit surfaces, which together with control flexibility are essential for an interactive design environment.

4 Hierarchical Skeletons

In this section our framework is presented. Although we restricted the discussion to implicit surfaces in $\mathbf{R}^3$, it can be easily generalized to any dimension.

4.1 Motivation

Careful analysis of previous skeleton-based models led us to conclude that we can get a large set of variants either by modifying the primitives or by altering the blending function. For the primitives we can also distinguish two terms:

- the Euclidean distance from $\mathbf{x} \in \mathbf{R}^3$ to a reference point, curve, or surface, which accounts for the level set shape (that is, a *shape function*); and
- the *decay function*, which forces the contribution of each reference entity to drop to an insignificant value at a certain distance from it.

From the discussion made in [2], we can say that there are basically three separate ways to interactively control an implicitly defined object:

- definition and manipulation of the shape function;
- definition and adjustment of the decay function; and
- definition and manipulation of the blending function.

These statements are the basis of our framework.

4.2 Structure

The scalar field defined by F is the result of a blending operation on a set of scalar fields. Each one of these fields may be, in its turn, composed of other fields, and so on. This naturally leads to a tree structure, whose leaf nodes correspond to *primitive fields*. Each internal node at the n-th depth level represents a scalar field resulting from the combination of the fields below it (Fig. 1). In the rest of this paper, we will refer to these leaf nodes as *primitive nodes* and to the internal nodes as *blend nodes*.

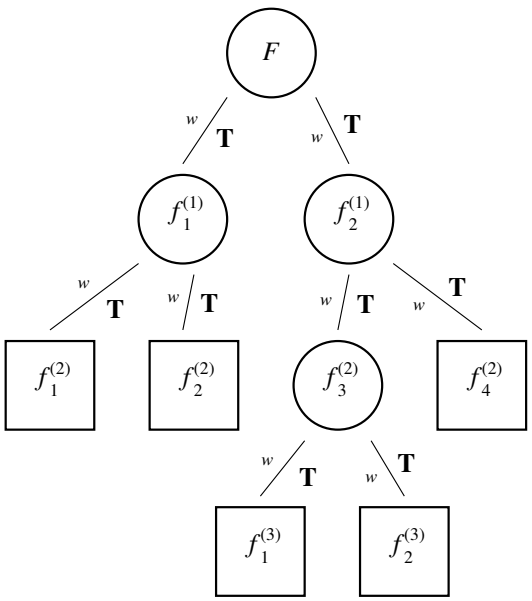

Figure 1: Skeleton structure.

A tree structure makes possible the selective combination of the primitives, and is not limited to just one type of blending operation as in the usual techniques. Furthermore, it simplifies the interactive manipulation of complex objects, because the user can separate the model into subtrees and work with only one at a time.

The main scalar field, which defines the implicit object, is given by

$$F(\mathbf{x}) = f_1^{(0)}(\mathbf{x}).$$

We may drop the subscript 1 because F is the unique field at the topmost level, which is the tree root node. The intermediate scalar fields associated to the other nodes are simply called *component fields*.

We define each field in its particular domain. This allows the choice of a convenient reference system for each field.

There is a transformation $\mathbf{T}_i^{(n)}$ that relates a field domain and its parent domain. When combining two or more scalar fields, we have to take this transformation into account in order to assure that the combination occurs under the same reference system [5]. Given a point $\mathbf{x}$ in the parent coordinate system, the associated field value is computed by the combination of its child field values. After the appropriate transformation $\mathbf{T}_i^{(n)}$ of $\mathbf{x}$ into each i-th child field domain, the child field values are obtained by applying $f_i^{(n)}$ on these transformed points.

Note that $\mathbf{T}_i^{(n)}$ is not limited to be a rigid motion: we also allow scaling and shearing transformations. With this, we can alter the resulting shape contribution of a particular scalar field. For example, two "ellipsoidal-shaped fields" can be obtained from two spherical ones by a scaling transformation (Fig. 2).

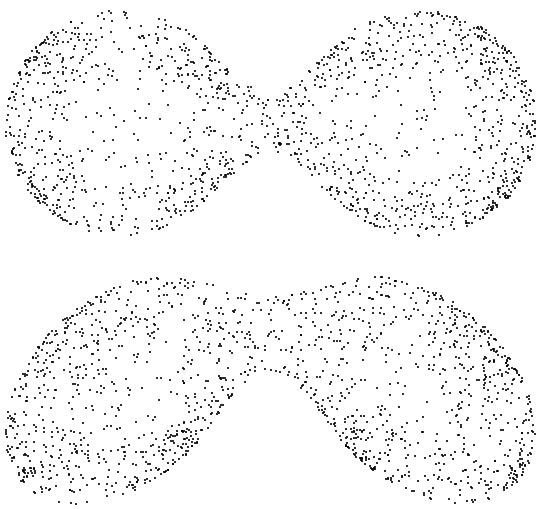

Figure 2: Operating on the domain of the primitives.

4.3 Blending Operations

The linear blend is the most commonly used blending operation for skeleton models, where the contributions of a set of functions are combined through a weighted sum.

We generalize this idea by associating a *weight* w_i to each component field f_i, independently of the blending function. That is, a blending operation combines the values $w_i f_i^{(n)}$ instead of $f_i^{(n)}$. The weight w_i can give a flexible control over the influence of each component field.

For example, a linear blending operation at the point $\mathbf{x}$ may be expressed by

$$f_i^{(n)}(\mathbf{x}) = \sum_j w_j f_j^{(n+1)}([\mathbf{T}_j^{(n+1)}]^{-1}\mathbf{x}), \qquad (1)$$

where j stands for its j-th component function. In this case, w_j may assume zero and even negative values, so that the corresponding field has respectively no and subtractive effects. In Fig. 3, the three primitives in the top row have weight 1. Below, the weight of the left primitive is set to 2, while the right one has a -2 value.

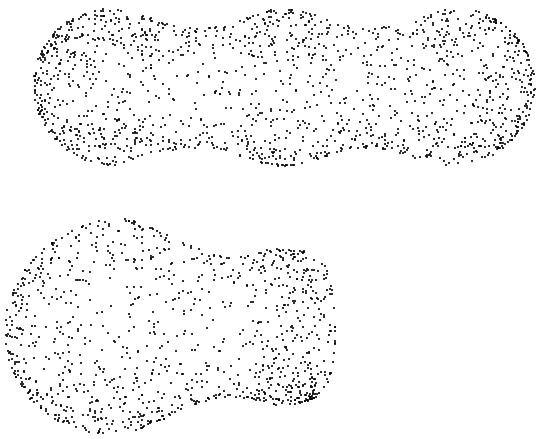

Figure 3: Influence of the weight.

Although not strictly a blend, another useful combining operation is the union. This operation takes the maximum of the field contributions in a given location, that is,

$$f_i^{(n)}(\mathbf{x}) = \max_j\{w_j f_j^{(n+1)}([\mathbf{T}_j^{(n+1)}]^{-1}\mathbf{x})\}. \qquad (2)$$

Note that this operation may introduce first order discontinuities on the surface, allowing us to create "sharp corners" on it.

4.4 Primitive Fields

Differing from blend nodes, a primitive node represents a field defined by

$$f_i^{(n)} = d_i \circ s_i,$$

where the functions s_i and d_i are called *shape* and *decay*, respectively.

4.4.1 Shape Function

As already explained, the shape function defines a scalar field in $\mathbf{R}^3$ and accounts for the form of the primitive. Because each field is defined in its own domain, the position, the size, and the orientation of any primitive can be arbitrary. Hence, we may always choose a convenient reference system to define it.

For example, a spherical primitive, centered at the origin and with *influence radius* given by r_i, can be expressed by

$$s_i(\mathbf{x}) = \frac{\|\mathbf{x}\|}{r_i}, \tag{3}$$

where $\mathbf{x} \in \mathbf{R}^3$. The scalar r_i is a user-defined parameter of this primitive[1]. Note that this field has value 1 at points belonging to the surface of a sphere with radius r_i and centered at the origin.

4.4.2 Decay Function

Usually, the field given by a shape function increases as we move away from the primitive. Thus we need to apply the decay function, to force that the field contribution goes toward zero with the augment of distance. In this way, local (or pseudo-local) control of the model is ensured.

Wyvill *et al.* [7] derived a decay function similarly shaped to the one proposed by Blinn [1], with the advantage of dropping to zero outside its range of influence. Their function required three additions and five multiplications. Based on the same principle applied by Wyvill *et al.*, we derived another similar function, also with limited influence. The additional advantage relies on the fact that our function is simpler, requiring fewer operations. We obtained this function by imposing that $\ddot{d}_i(0) = \ddot{d}_i(1) = 0$, instead of requiring that $d_i(0.5) = 0.5$.

Thus, our decay function is given by

$$d_i(t) = \begin{cases} 1, & \text{if } t \leq 0 \\ -(t-1)^3, & \text{if } 0 < t < 1 \\ 0, & \text{otherwise} \end{cases} . \tag{4}$$

Observe that the result of the composition $d_i \circ s_i$, given respectively by equations (4) and (3), is a piecewise monotonic and non-increasing function, whose value vanishes after a distance greater than or equal to r_i from the origin (see Fig. 4).

[1] In fact, this parameter is redundant, because the same effect could be obtained by applying a uniform scaling transformation on the domain of a spherical primitive with unitary radius. This is left in the prototype as a convenience.

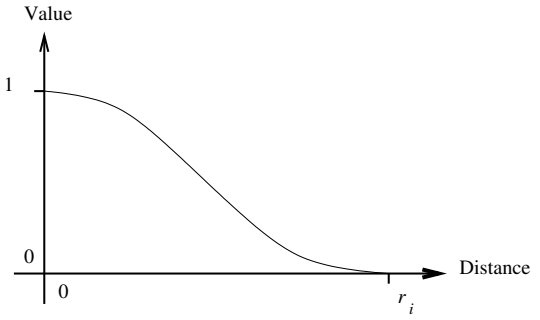

Figure 4: Spherical primitive field function.

5 Implementation

Since the implicit object is recursively defined from primitive fields, an adequate data structure for its representation is an n-ary tree. In this tree, the leaf nodes denote the primitive field functions and the internal nodes represent the blending functions, which are used to combine the function values of its child nodes.

Each node contains a homogeneous transformation matrix $\mathbf{T}$ that relates its domain coordinate system to the one from its parent. It also contains a weight w that affects the contribution of the node to the parent scalar field.

We developed a C++ class which implements this data structure. The methods provided by this class were designed aiming interactive manipulation on the implicit objects, as listed in the following:

- Methods to create and destroy primitive nodes. There are also methods to individually query and modify the parameters of their shape and decay functions.
- Methods to create and destroy blend nodes. Their parameters can be queried and modified (including the weight w), and the blending operations can also be changed.
- Methods to modify the transformation matrix $\mathbf{T}$ between a node and its parent.
- Methods to manipulate the tree structure (e.g., get root node, query parent, query child, remove subtree, etc).
- A method to query and modify the current level set value c of the main scalar field.
- A method to evaluate the main scalar field at a given point, returning both the scalar value and the associated gradient vector.

We used abstract classes for defining primitive fields and blending operations. Hence, new primitives and new

blends can be easily integrated by deriving specialized classes.

We have already implemented the primitive function shown in Fig. 4 and the operations given by the equations (1) and (2).

Several authors [2, 4] have observed that presenting an implicit object as a sample of points offers greatly improved speed relative to polygonization. Moreover, use of appropriate depth-cueing proves to be very effective for visualizing clouds of points in space.

Since we are interested on the rapid prototyping of the implicit object, we opted for visualizing the implicit object as a set of particles. We used the algorithm presented by Figueiredo and Gomes [4] to place these particles on the surface of the implicit object. Roughly speaking, their algorithm randomly scatters particles in R^3 and uses a modified gradient vector field to force them to migrate to the surface of the object.

For the rendering we used the OpenGL graphics library. The depth cue was obtained by using its *fog* feature.

6 Results

A model of a hand was built from a set of 17 primitives, whose outlines and relative positions are shown in Fig. 5. The resulting model is presented in Fig. 6, being visualized by 3200 particles lying at the object surface.

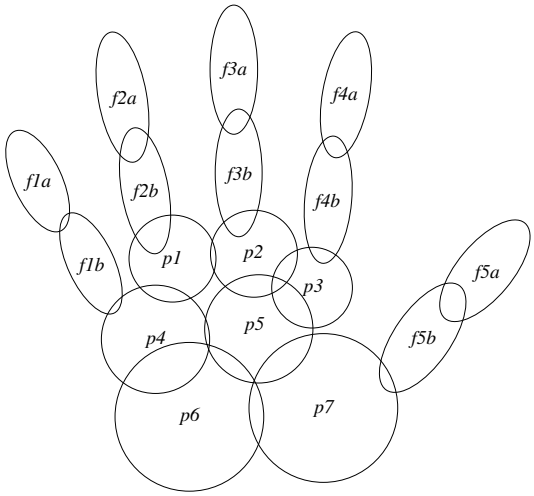

Figure 5: Outlines of the primitives.

First the palm was built, using 7 primitives, flattened from their original spherical shapes. Then each finger was modeled separately, as a subtree with two primitives. Next, the fingers were positioned relative to the palm, as if they were single primitives. Only linear blending operations were used (denoted by "+"), with unitary weight for each primitive (Fig. 7).

Fig. 8 exemplifies a modification of this hand, where the fingers are slightly moved apart from each other. Due to the hierarchical structure, to achieve this effect we just need to apply rotations to the corresponding subtree root nodes: *f1*, *f2*, *f3*, *f4*, and *f5*.

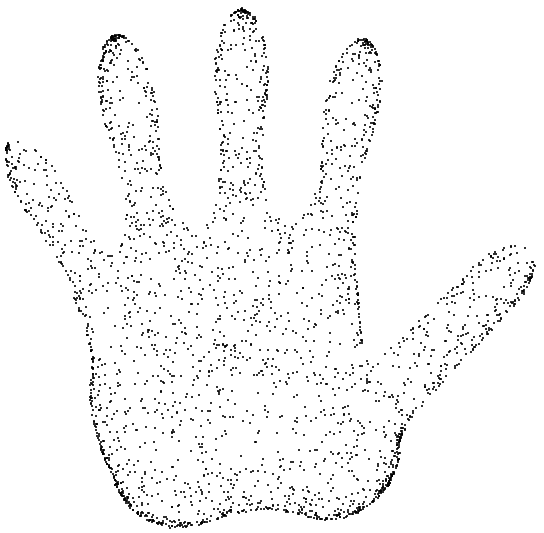

Figure 6: Implicit model of a hand.

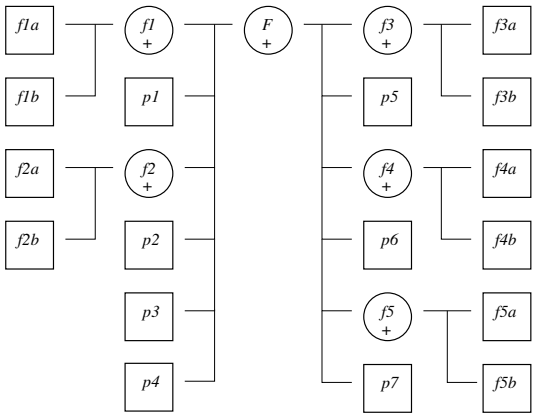

Figure 7: Structure of the hand.

The next example demonstrates the use of two combining operations—linear blend and union—in the same implicit object.

The object shown on the top of Fig. 9 is made of 3 primitives, with its corresponding tree structure sketched below it. The primitives labelled *g1a* and *g1b* are combined through a union operation (indicated by "U"), making the subtree labelled *g1*. This subtree is then combined with the horizontal primitive *g2* through a linear blend, thus yielding the final object. Observe the different connections between the primitives. The use of a union operation leads to a sharp corner (A), while the linear blend gives a smooth joint (B or C).

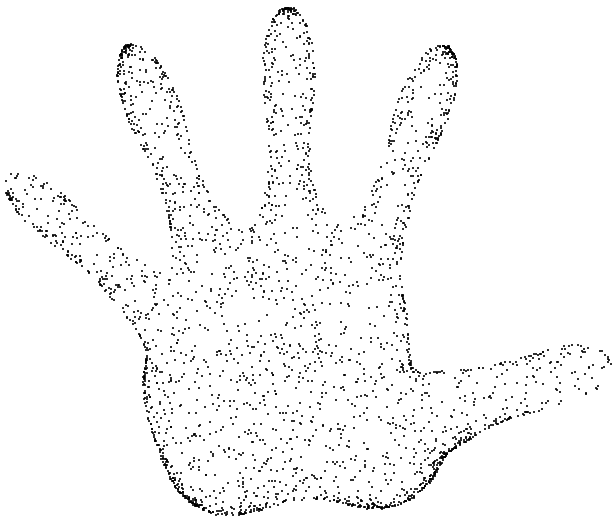

Figure 8: Modified hand.

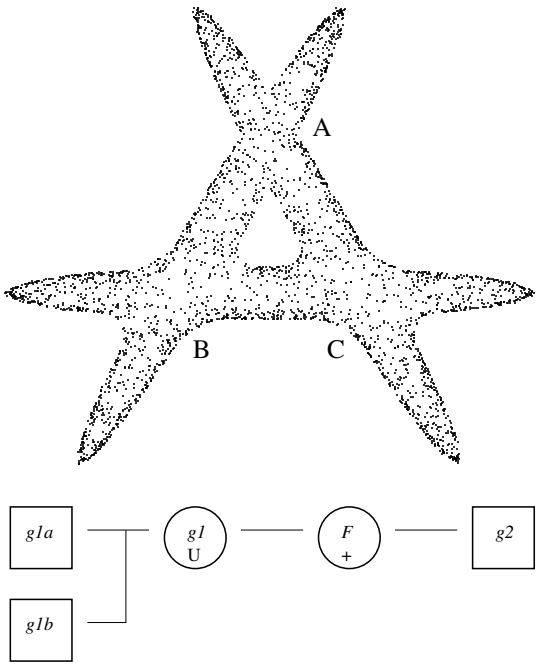

Figure 9: Example of multiple blends.

7 Conclusions

We presented a framework that not only encompasses several existing skeleton-based models but also allows their extension. The proposed hierarchical structure provides adequate definition and manipulation semantics for complex objects.

Our goal has been to support modeling of complex implicit objects in an interactive design environment. In this paper we addressed only the modeling problem, but we believe that it can be the underlying structure for interactive systems. Observe that the control flexibility is accomplished by the set of parameters involved in the definition of the model, which allow a vast range of intuitive geometric variations.

Regarding to the modeling problem, two contributions are worth to be mentioned. First, we derived a computationally cheap decay function. Then, we have used geometrical transformations on the function domain to improve the modeling flexibility.

As future work, we plan to explore the interactivity power of the proposed framework.

8 Acknowledgments

We wish to thank Marcos K. Aguilera and the referees for their helpful suggestions on the improvement of this paper. We also thank CNPq for the financial support.

References

[1] Blinn, J. F., *A Generalization of Algebraic Surface Drawing*, ACM Transactions on Graphics, 1(3), 1982.

[2] Bloomenthal, J. and Wyvill, B., *Interactive Techniques for Implicit Modeling*, Computer Graphics (1990 Symp. on Interactive 3D Graphics), 24(2), 1990.

[3] Bloomenthal, J. and Shoemake, K., *Convolution Surfaces*, Computer Graphics, 25(4), 1991.

[4] de Figueiredo, L. H. and Gomes, J., *Sampling Implicit Objects with Physically-based Particle Systems*, Computer & Graphics, 20(3), 1996.

[5] Gomes, J. and Velho, L., *Implicit Objects in Computer Graphics*, IMPA, 1992.

[6] Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I., and Omura, K., *Object Modeling by Distribution Function and a Method of Image Generation*, Japan Electronics Communication Conference 85, 1985.

[7] Wyvill, B., McPheeters, C., and Wyvill, G., *Data Structure for Soft Objects*, The Visual Computer, 2(4), 1986.

[8] Wyvill, B., McPheeters, C., and Wyvill, G., *Animating Soft Objects*, The Visual Computer, 2(4), 1986.