A Hybrid Shape Representation for Free-form Modeling

Rémi Allègre Aurélien Barbier Eric Galin Samir Akkouche

LIRIS CNRS, Université Claude Bernard Lyon 1, France

{Remi.Allegre,Aurelien.Barbier,Eric.Galin,Samir.Akkouche}@liris.cnrs.fr

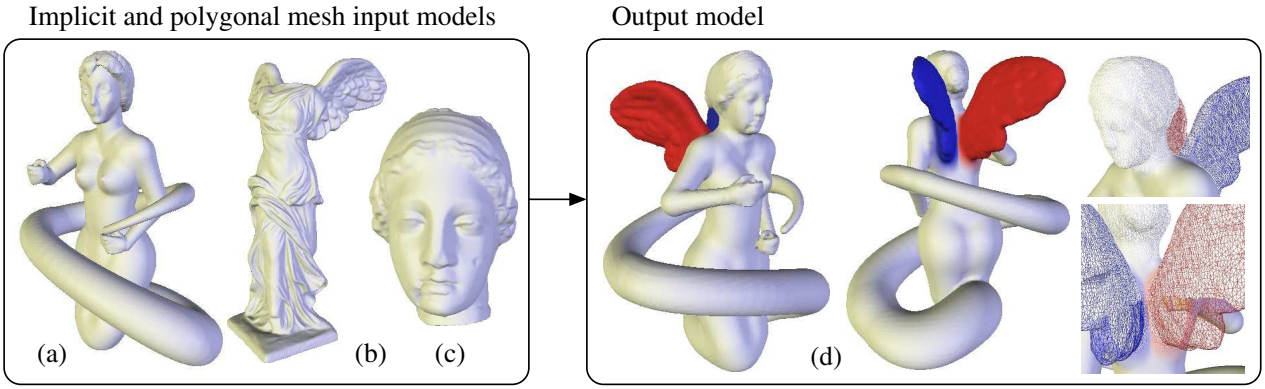

Figure 1. Modeling the winged snake-woman. Input models consist of an implicitly defined character (a), a mesh model of the Victory of Samothrace (b), and the Igea mesh model (c). These models are combined using intersection, difference and blending to obtain the final model (d). In the rightmost close-up views, implicit parts of the final model are rendered with smooth shading while mesh components are rendered in wireframe.

Abstract

In this paper, we introduce a hybrid modeling framework for creating complex objects. Our system relies on an extended CSG tree that assembles skeletal implicit surfaces and polygonal meshes in a coherent fashion: we call this structure the HybridTree. Editing operations are performed by exploiting the complementary abilities of both implicit and parametric surface representations. Implicit surfaces are powerful for combining shapes with Boolean and blending operations, while polygonal meshes are well-suited for local deformations such as FFD and fast visualization. The evaluation of the HybridTree is performed on the fly either through field function queries when the implicit form is required, or through a mesh creation process which is specific and optimized for every kind of node. Both types of queries are achieved in a complete transparent way for the user.

Keywords: shape modeling, implicit surfaces, polygonal meshes, blending, free-form deformations.

1. Introduction

In computer graphics, modeling realistic complex virtual environments has been an active research domain for several years. In this scope, efficient shape representations and modeling tools are among central concerns.

Implicit and *parametric* representations have specific and complementary advantages and drawbacks. Implicit surfaces [32] are powerful for representing objects of complex geometry and topology [16, 15, 22], and they naturally lend themselves for blending or Boolean operations. We have contributed to develop the *BlobTree* implicit modeling system [33], which is based on an extended CSG tree that combines skeletal implicit surfaces using Boolean, blending and warping operators. This model has proved to be a powerful tool for modeling and animating complex and realistic organic shapes [6]. However, local deformations are difficult to implement and are quite restrictive in most implicit modeling frameworks. Moreover, visualization of complex implicit surfaces is computationally demanding.

In contrast, parametric surfaces such as polygonal meshes can be efficiently visualized thanks to common graphic hardware. The surface may be edited interactively by a variety of powerful tools, such as free-form deformations [28], that provide a very intuitive local control over geometry. However, combining parametrically defined surfaces with Boolean operations is a complicated task, which is prone to topological inconsistencies. Moreover, parametric surfaces do not lend themselves for blending.

Mixing both surface models so as to benefit from their complementary advantages may be addressed through two kinds of approaches. For almost thirty years, conversion techniques have been developed to switch between these representations. Combining their abilities into a single hybrid model has recently received much attention in the computer graphics community for either geometry processing or shape modeling.

In this paper, we describe a new hybrid shape representation mixing both implicit and parametric representations for incremental modeling of complex shapes. Our goal is to provide the user with means of coherently combining objects from either representation in a complete transparent way. Our system addresses key issues such as fusion and local deformation of complex models, high reusability, and low storage and computational costs.

Our model is characterized by a tree data-structure that combines skeletal implicit surfaces and polygonal meshes by means of Boolean, blending and warping operators, including free-form deformations. We call our model the *HybridTree*, which may be seen as a generalization of the BlobTree [33]. The key features of the HybridTree are specific optimized methods for evaluating this structure in a coherent way, so as to efficiently perform high-level editing operation and visualization. The system inherits from both implicit and parametric surface representations so as to use the most suitable representation for every type of operation. The system dynamically switches from one representation to the other whenever needed during the evaluation process.

We are able to perform Boolean or blending operations on either implicit surfaces or polygonal meshes, including interactions between implicit surfaces and meshes. We also bring free-form deformations tools permitting local surface deformations to be applied to either implicit surfaces or polygonal meshes. We propose efficient conversion techniques, achieved in an incremental way, which are completely hidden for the user. We introduce a point to mesh distance for implicitizing a polygonal mesh with full control over the field function parameters. We finally bring a novel per-primitive meshing technique which efficiently integrates partial results and produces good aspect ratio meshes.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of related shape modeling frameworks. Section 3 describes the architecture of our system and present how implicit and parametric representations are combined together. In Section 4, we address some implementation details concerning the implementation of the HybridTree. Applications of the HybridTree to complex shape modeling are discussed in Section 5. Eventually, in section 6, we conclude and present future work.

2. Related work

Conversion techniques from implicit to parametric or from parametric to implicit representations have been extensively studied [20, 8, 31, 23]. These techniques make possible objects in either representation to coexist and interact in the same environment through a unified representation, based on the conventional implicit or parametric formulations. Some recent works focused on discrete implicit representations, such as Adaptive Distance Fields [15] or the level set framework proposed in [22]. These systems provide fast conversion algorithms from many other representations, and a wide range of robust editing tools. However, this representation is memory consuming and not well-suited for large deformations.

Hybrid models have been investigated by several authors for shape modeling in the last years. In the field of geometry processing, some specific problems may be efficiently solved using this approach. A good overview can be found in [19]. A hybrid system mixing volumetric and function implicit representations has been studied in [2]. The method relies on a conversion to voxel representation so as to make both representations to interact in a coherent way.

Depending on the surface representation, some operations cannot be performed easily in a direct way. In some cases, such an issue can be addressed with the help of an intermediate representation. Several methods for performing blending between polygonal meshes have been proposed. In [14, 30], an intermediate implicit representation built from star-shaped or locally star-shaped polyhedra is used, which is an important geometric restriction. Another method proposed in [18] operates in two steps. First, a registration process is applied between mesh boundaries that should be homeomorphic and then a B-Spline model is used to smooth the surface in the blending region. In this case, the restriction is of topological order.

Some hybrid techniques rely on implicit function to deform polyhedral objects [29, 12, 13]. A hybrid modeling framework using an implicit surface model to perform blending on point-sets has been proposed in [26]. The variational technique presented in [27] generates an implicit surface from point-sets via an interpolation scheme based on compactly supported radial basis functions. The resulting shapes can be locally controlled in an intuitive way by acting on the constraint points. The evaluation may be per-

formed at interactive rates using an octree data-structure. However, this approach remains computationally demanding when manipulating dense point sampled geometry, and sharp features are difficult to handle in this framework.

3. The HybridTree

The HybridTree model relies on a tree data-structure whose leaves hold either implicit primitives or polygonal meshes. These are combined by means of Boolean, blending and warping operators located at the nodes of the tree. Warping nodes include affine transformations, Barr deformations [7] and free-form deformations [28]. Constructive operators are binary whereas warping operators are unary operators.

Figure 2 shows the tree structure of the winged snake-woman model represented in Figure 1. The snake-woman model (a) has been entirely built from skeletal implicit surfaces. Using Boolean difference, only the body has been conserved, which has been then blended with the Igea mesh (b) so as to obtain the model (c). The wings of a mesh model of the Victory of Samothrace (d) have been extracted by intersecting the model with a box. The wings and the modified snake-woman model have been finally blended together in (e).

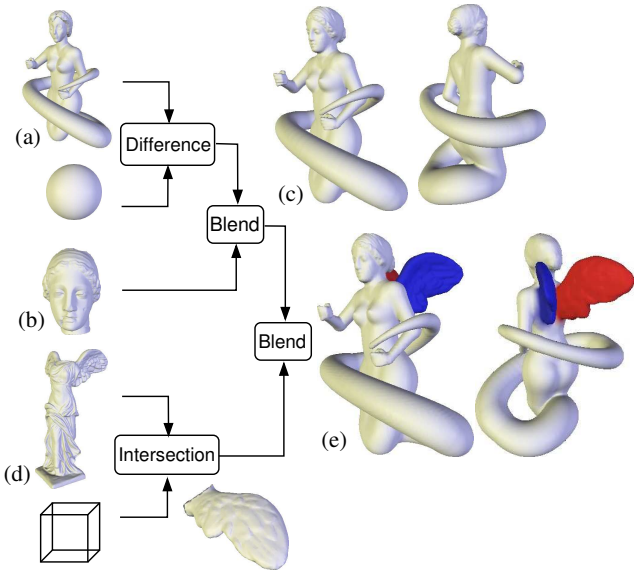

Figure 2. The HybridTree data-structure of the winged snake-woman model.

The evaluation of the HybridTree is achieved in an incremental way by recursively traversing the tree data-structure. The HybridTree may be evaluated through two kinds of queries. Field function queries at a given point in space are performed whenever the implicit formulation is required. An incremental polygonization process is invoked at a given node if the parametric formulation is needed. Partial results are then combined in a coherent fashion by the operators. In the following paragraphs, we detail both potential field function queries and meshing queries that are specific for every node in the HybridTree.

Notations Throughout the remainder of this section, the field function for a node A will be denoted as f_A, and $\mathcal{M}_A$ will refer to the mesh of the surface of A. The bounding box of the object A will be denoted as $\mathcal{B}_A$.

3.1. Queries on primitives

Our system handles implicit primitives built from complex skeletons as described in [33] and triangular meshes with manifold topology. Conversions between implicit and mesh representations occur frequently as the system dynamically adapts to the current representation while traversing the HybridTree. Therefore, efficient conversion methods are needed, ideally without any loss of geometrical and topological precision. Thus, every primitive implements specific functions for efficiently evaluating the field function and generating a triangle mesh.

3.1.1 Skeletal implicit primitives

For a given skeletal implicit primitive, the field function is evaluated using the following formulation:

$$f(\mathbf{p}) = g \circ d(\mathbf{p})$$

where $d : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ denotes the Euclidean distance to the skeleton, and $g : \mathbb{R}_+ \rightarrow \mathbb{R}$ refers to the potential field function. The latter is a compactly supported radial basis function that is parameterized by a maximum field value reached on the skeleton, and a radius of influence that will be denoted as R. The associated region of influence, characterized by non zero field values, will be denoted as Ω. The surface S of the object is defined as the points of space whose potential equals a threshold value denoted as $T \in \mathbb{R}$:

$$S = \{\mathbf{p} \in \mathbb{R}^3 | f(\mathbf{p}) = T\}$$

The HybridTree implements a wide range of complex skeletal primitives including line segment, curves and surfaces, surfaces of revolution and volumes such as boxes, cylinders and cone-spheres. The computation of $d(\mathbf{p})$ is optimized for each kind of skeleton.

Meshing The surface of a primitive is obtained by sweeping a sphere of constant radius $r_0 = g^{-1}(T)$ along the

skeleton. In our system, every implicit primitive can automatically generate an optimal mesh representation characterized by almost equilateral triangles everywhere. This mesh can be generated at different levels of resolution. Moreover, the mesh is produced by a specific and optimized procedure which makes mesh queries very efficient.

3.1.2 Polygonal meshes

Mesh leaves contain imported mesh models that are instantiated only one time for the whole evaluation process. For such nodes, the mesh creation process simply returns the original mesh, which is not duplicated in memory. Field function evaluations also directly apply to this information. In the following paragraph, we precisely describe our method for evaluating the field function from meshes.

Field function evaluation For a polygonal mesh, the field function is computed using the same formulation as for skeletal implicit primitives. The distance function $d_{\mathcal{M}}$ from a point $\mathbf{p} \in \mathbb{R}^3$ to a triangular mesh $\mathcal{M}$ is defined as the minimal Euclidean distance between $\mathbf{p}$ and any triangle $\mathcal{T}$ that of the boundary of $\mathcal{M}$:

$$d_{\mathcal{M}}(\mathbf{p}) = \min_{T \in \mathcal{M}} d(\mathbf{p}, \mathcal{T})$$

The distance $d(\mathbf{p}, \mathcal{T})$ is evaluated using the Voronoï diagram of $\mathcal{M}$ for point classification. The implicit surface generated by the skeletal mesh for a given threshold T is a rounded surface S which differs for the original mesh $\mathcal{M}$. This surface S may be defined by sweeping a sphere of radius $r_0 = g^{-1}(T)$ along the boundary of $\mathcal{M}$ (Figure 3(a)). To make the boundary of $\mathcal{M}$ and the isosurface to correspond independently from the field function parameters, we incorporate the threshold as an offset in a pseudo-distance function which is defined as follows:

$$d(\mathbf{p}) = \begin{cases} d_{\mathcal{M}}(\mathbf{p}) + r_0 & \text{if } \mathbf{p} \notin \mathcal{M} \\ r_0 - d_{\mathcal{M}}(\mathbf{p}) & \text{if } \mathbf{p} \in \mathcal{M} \text{ and } d_{\mathcal{M}}(\mathbf{p}) < r_0 \\ 0 & \text{otherwise} \end{cases}$$

Our distance function guarantees that the isosurface and the boundary of the mesh should be the same for any value of T, as shown in Figure 3(b). The user keeps control on every parameter of the field function, and may precisely control the range of the blend between two objects. The radius of influence of the mesh, which falls from R to $R - r_0$, is rescaled appropriately so that the distance offset is hidden for the user.

Since computing the minimum distance between a point $\mathbf{p}$ and all the triangles $\mathcal{T}$ of the mesh $\mathcal{M}$ is computationally expensive, we use Johnson and Cohen's algorithm [17]. This algorithm relies on a bounding box hierarchy built from a BSP data-structure, which is parsed using breadth-first traversal. For each node is computed a lower and an upper bound of the minimum point to mesh distance, which yields efficient space-pruning. We use the fact that the potential falls to 0 beyond the distance r from the mesh boundary so as to reject more useless point to mesh distance computations. Indeed, for a given point in space $\mathbf{p}$, $f(\mathbf{p})$ is non-zero if and only if $d(\mathbf{p})$ is less than r. The effectiveness of this optimization thus varies according to the geometry of $\mathcal{M}$ and the size of $R - r_0$. Lots of time is saved when r is small regarding the overall size of the mesh as only a very reduced set of bounding boxes have to be tested. The benefit of our acceleration decreases as r increases.

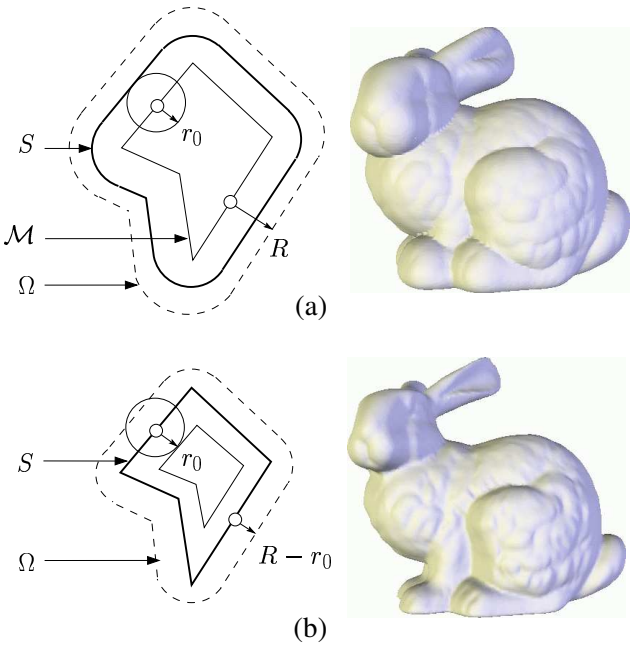

Figure 3. Applying our distance function to a mesh. The offset mechanism is schematized on the left, and we take the Stanford Bunny as an example. At the top (a), the basic distance formula is used. The model at the bottom (b) has been computed using our pseudo-distance.

3.2. Queries on blending and Boolean operators

Meshing an implicit surface is still a challenging issue. Existing implicit surface meshing techniques rely on many evaluations of the potential field function, which is computationally demanding in the general case. Our approach consists in optimizing the mesh generation of every primitive of the HybridTree and using specific meshing algorithm for every node. Wherever two primitives do not overlap very much, we observed that it is better to generate the meshes of the primitives before combining them rather than computing the overall mesh from scratch. The following

paragraphs detail specific methods for every operation in the HybridTree.

3.2.1 Blending operators

One of the key feature of implicit surfaces is their ability to blend together easily. We exploit this interesting property in our system so as to perform blending on our hybrid models, through both field function and polygonization queries.

Field function evaluation Let A and B denote two models that blend together. In our system, global blending between two objects is functionally defined as:

$$f_{A+B} = f_A + f_B$$

Meshing First, we will assume that A and B are associated with positive potential fields. If the two models A and B are only partially blended together, then the mesh of C is created after the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$. We use the incremental Marching Triangles algorithm as described in [4] to generate the mesh in the blending region. Otherwise, the whole mesh of C is generated by applying the Extended Marching Cubes algorithm as proposed in [20].

To determine the most favorable approach, we estimate how much the models A and B overlap, i.e. which proportion of volume is shared in their blending region. We introduce a ratio denoted as ρ such that $0 \leq \rho \leq 1$, which is computed as follows:

$$\rho = \frac{V_{A\cap B}}{V_{A\cup B}} = \frac{V_{A\cap B}}{V_A + V_B - V_{A\cap B}}$$

where V_A, V_B and $V_{A\cap B}$ denote the volume of the bounding boxes $\mathcal{B}_A$, $\mathcal{B}_B$ and $\mathcal{B}_A \cap \mathcal{B}_B$ respectively (Figure 4(b)).

We choose $\rho = 0.5$ as a default threshold, which appears as a good guess in most cases. If $\rho > 0.5$, the mesh of C is generated by the Marching Cubes technique. Otherwise, the algorithm proceeds as follows:

1. Create the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$ of A and B respectively.

2. Remove the triangles of $\mathcal{M}_A$ and $\mathcal{M}_B$ that lie in the inter-influence region of A and B, i.e. triangles that have at least one vertex $\mathbf{p}$ such that $f_B(\mathbf{p}) > 0$ for $\mathcal{M}_A$, and such that $f_A(\mathbf{p}) > 0$ for $\mathcal{M}_B$.

3. Invoke the Marching Triangles technique, starting from the boundary edges of the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$ to close the mesh.

For Step 3, we define a sampling rate that adapts to the length of the edges of the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$ so as to deal with objects of different scale.

The user can provide a value for ρ or directly specify which method should be used for each blending operator involved in a tree. If $\rho > 0.5$ for a given blending node, then ρ is also computed for the parent node if it is a blending node. For efficiency reasons, the Marching Cubes algorithm is applied to the first ancestor blending node encountered while traversing up the tree.

Figure 4 illustrates our algorithm for $\rho \leq 0.5$. The meshes of A and B are first generated. The triangles of $\mathcal{M}_A$ and $\mathcal{M}_B$ that are located in the inter-influence region of A and B are red. Then, these triangles are removed as shown in (b), and the final mesh is obtained by applying the Marching Triangles technique from the blue boundary edges. The new triangles produced during this step are the green ones in (c).

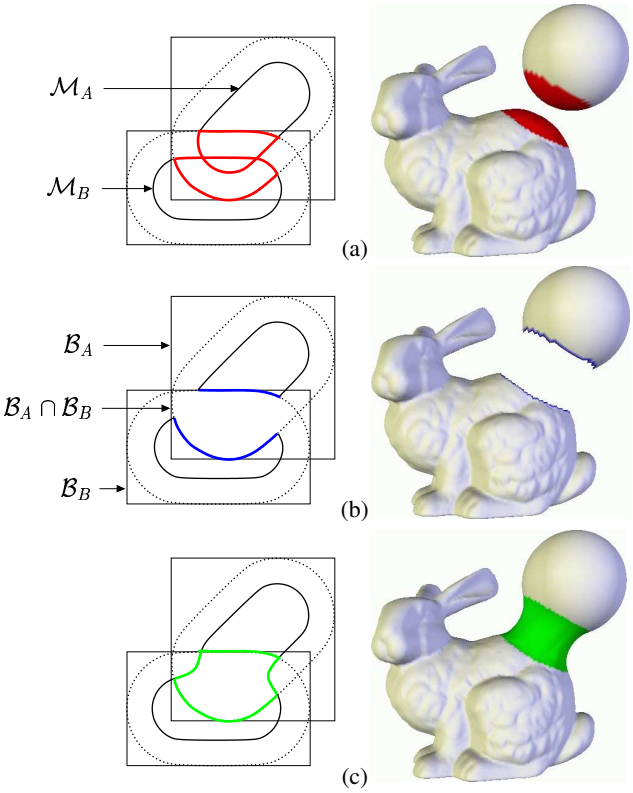

Figure 4. Meshing two blended objects A and B for $\rho < 0.5$. As an example, we show the Stanford Bunny that is blended with an implicit sphere.

Negative blending Negative blending smoothly removes material and creates holes in the original object. In this case, the previous algorithm is slightly modified as follows: In Step 1, we need only create the mesh $\mathcal{M}_A$ of A. In Step 3 we remove the triangles of $\mathcal{M}_A$ that lie in the inter-influence region of A and B, i.e. triangles that have at least one vertex

$\mathbf{p}$ such that $f_B(\mathbf{p}) < 0$. The Marching Triangles technique is invoked in Step 3 starting from the boundary edges of the meshes $\mathcal{M}_A$ to close the mesh.

3.2.2 Local blending

We have implemented a new local blending operator adapting the local blending technique described by [25]. This operator has three children, the first two, denoted as A and B, represent the two models that will be partially blended together, whereas the third, denoted as R, represent the region of space where blending will occur.

In our system, R is characterized by a potential field that is defined as a union of implicit primitives denoted as R_i (Figure 5). Such a combination makes it possible to build complex blending regions with predictible results. The field function f_R characterizes the blending region and is used to scale the amount of blending between the two sub-trees A and B. The values taken by the field functions f_{R_i} should range between 0 and 1, so that f_R is bounded in the same way. At a given point in space $\mathbf{p}$, if $f_R(\mathbf{p}) = 0$ then only union occurs, which is the case for any point outside the region of influence of R. In contrast, if $f_R(\mathbf{p}) = 1$, full blending takes place normally.

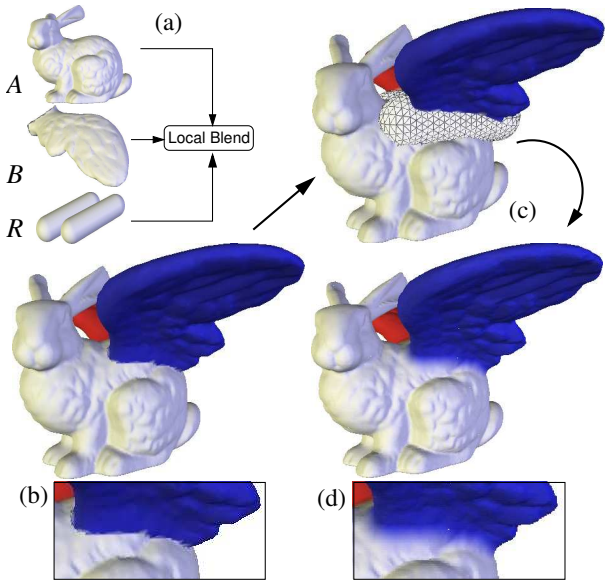

Figure 5. Illustration of our local blending operator. The HybridTree appears in (a). The blending region R is built from the union of two implicit segments. In (b) is shown the union of objects A and B. In (c), the boundary of the blending region is depicted in wireframe. The final result is presented in (d).

Field function evaluation The evaluation of the local blending operator is performed as follows. We first compute the potential field value resulting from the blending of the children nodes $f_{A+B}(\mathbf{p}) = f_A(\mathbf{p}) + f_B(\mathbf{p})$, and the field function value $f_{A\cup B}(\mathbf{p})$. We define the resulting field function as a weighted average:

$$f(\mathbf{p}) = f_R(\mathbf{p})\, f_{A+B}(\mathbf{p}) + (1 - f_R(\mathbf{p}))\; f_{A\cup B}(\mathbf{p})$$

Primitives build from a volume skeleton are very useful to define regions in space where full blending occurs.

Meshing Here we suppose that A and B generate positive potential fields, which is not required for R. The polygonization of the local blending node is the same as for global blending between positive potential fields with the only difference that Step 2 is modified so that triangles of $\mathcal{M}_A$ and $\mathcal{M}_B$ are removed if they intersect the blending region R.

3.2.3 Boolean operators

Union, intersection and difference naturally apply to implicit representations. The provided volume information offers fast point membership classification which contributes to efficiently generate mesh representations from such operations.

Field function evaluation The $\min$ and $\max$ functions prescribed in [33] for union and intersection produce gradient discontinuities in the potential function. This results in visible unwanted normal discontinuities on the surface.

Contrary to $\min$ and $\max$ functions, R-Functions define a field function with C^n continuity almost everywhere in space, except on the surface, which avoids unwanted discontinuities. The functions prescribed in [24] operate on field functions that have an infinite support, whereas our model operates on field functions that have a compact support. Moreover, R-Functions have been designed for implicit surfaces characterized by a null threshold value: $T = 0$.

We have adapted those functions to our model by incorporating the threshold value as an offset in the previous equations. A weighting coefficient appears so as to guarantee that the resulting field function still has a compact support. We have:

$$f_{A\cup B} = T + \frac{1}{2-\sqrt{2}}\Big[(f_A - T) + (f_B - T) + \sqrt{(f_A - T)^2 + (f_B - T)^2}\Big]$$

$$f_{A\cap B} = T + \frac{1}{2+\sqrt{2}}\Big[(f_A - T) + (f_B - T) - \sqrt{(f_A - T)^2 + (f_B - T)^2}\Big]$$

Although $\min$ and $\max$ functions on the one hand, and R-Functions on the other hand produce different potential fields in space, both reprensetations 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 HybridTree. Our system automatically adapts the function used to evaluate Boolean operators depending on the context during the evaluation.

Meshing Computing the mesh resulting from Boolean operations may be achieved as performed by standard B-Rep modelers. Our approach takes advantage of the dual representation of the Hybrid Tree. We rely on the implicit representation of the child nodes to efficiently perform point membership classification. For polygonal meshes, we perform an optimized in/out test. The algorithm for the intersection may be written as follows:

1. Create the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$ of A and B respectively.
2. Remove the triangles of $\mathcal{M}_A$ that lie inside B, i.e. remove triangles of $\mathcal{M}_A$ that have at least one vertex $\mathbf{p}$ such that $f_B(\mathbf{p}) > 0$, and process the triangles of B in the same way.
3. Invoke a crack fixing algorithm to close the mesh.

The crack fixing algorithm invoked in step 3 consists in bridging the gap between boundary triangles to close the polygonization. Our algorithm creates new triangles from the boundary edges. For one given pair of contours, we compute the closest neighbors from one side to the other and corresponding vertices are connected. This technique works well if the meshes $\mathcal{M}_A$ and $\mathcal{M}_B$ have triangles of almost the same size, but fails at producing good closing triangles when the length of the facing edges is too different.

3.3. Queries on warping operators

In our implementation, warping operators include affine transformations, Barr [7] deformations as global deformations. Our system also handles free-form deformations [28], denoted as FFD, so as to perform local deformations. Throughout the following paragraphs, w will denote a space transformation that maps $\mathbb{R}^3$ into $\mathbb{R}^3$, and w^{-1} the corresponding inverse transformation.

3.3.1 Affine transformations and Barr deformations

Affine transformations and twist, taper and bend deformations directly apply to polygonal meshes as well as to implicit surfaces, as we can compute the closed form expression of w^{-1}.

Field function evaluation Let A denote an object that is deformed. The resulting field function is defined using inverse space mapping as follows:

$$f_w(\mathbf{p}) = f_A \circ w^{-1}(\mathbf{p})$$

Meshing We first create the mesh $\mathcal{M}_A$ of A. Then the deformation is applied to the mesh $\mathcal{M}_A$ by simply changing the coordinates of any vertex $\mathbf{p}$ into $w(\mathbf{p})$ so as to obtain the deformed mesh.

Translation, rotation and uniform scaling preserve the aspect ratio of the triangles, whereas non uniform scaling or twisting, tapering and bending may stretch the triangles into flat triangles. In those cases, we can use a local remeshing process to get better shaped triangles.

3.3.2 Free-form deformations

Free-form deformations have been first introduce by Sederberg and Parry in [28], and then have been extended by several authors [11, 9, 21]. Applying local deformations to implicit surfaces is not straightforward as there is no easy way of computing an analytical formulation for w^{-1}. In contrast, FFD directly apply to triangle meshes.

Meshing The mesh $\mathcal{M}_A$ of A is first generated, and then the deformation is applied to the mesh $\mathcal{M}_A$ so as to obtain the deformed mesh.

Warping nodes hold a mesh representation of their own resulting surface using this method. Queries are performed directly on the stored mesh, preventing any recursive call. These data are useful when intensive evaluations of the field function are performed.

Field function evaluation In our framework, FFD operators are evaluated using an intermediate mesh representation. Let A denote the child object of the warping node. The algorithm proceeds as follows:

1. Generate the mesh $\mathcal{M}_A$ of A.
2. Apply the FFD to $\mathcal{M}_A$ by transforming the vertices of $\mathcal{M}_A$.
3. Compute the field function value using the distance to the deformed mesh $\mathcal{M}_A$.

4. Implementation details

We have implemented a prototype software of the HybridTree in C++ on a Linux system. The nodes of the HybridTree have been implemented as shown in the inheritance diagram provided in Figure 6, in which the name of abstract classes appears in italic. This diagram does not present all primitives for clarity reasons. The algorithms for the evaluation of the HybridTree have been implemented as the following member functions:

- `double Node::Intensity(Vector)` perform field function queries, which returns the value of the field function at the given point in space.
- `Mesh Node::Polygonize(double)` perform the incremental polygonization queries, which returns a mesh of the given precision.

These functions are specialized through the inheritance diagram for every kind of node. For instance the function `Node::Polygonize(double)` computes the mesh in the general case using the extended Marching Cubes algorithm proposed in [20]. The function `Sphere::Polygonize(double)` computes the mesh of a sphere using the efficient ad-hoc algorithm proposed in [3]. The function `Blend::Polygonize(double)` computes the mesh of a blending node using the proposed strategy, depending of the value of the parameter ρ.

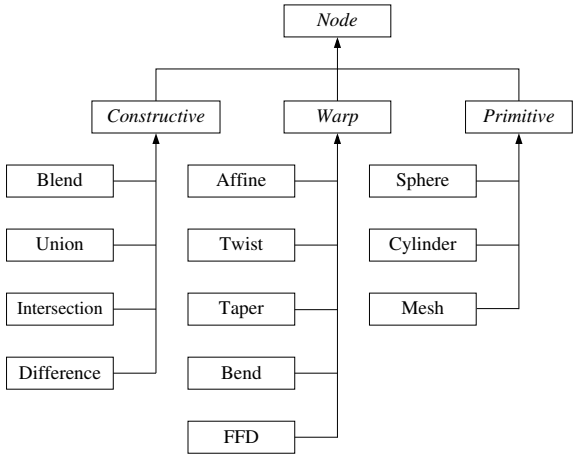

Figure 6. Inheritance diagram for the HybridTree.

The root of a HybridTree is encapsulated in the class `Tree` which provides the user with means of querying the structure. Below, we show the code of our declarative interface for the winged snake-woman of Figures 1 and 2.

```
Mesh WingedSnakeWoman(){

// Primitives:
//  Import the snake-woman model
Node snakewoman = CreateSnakeWoman();
//  Import the Victory mesh model
Node victory = new HMesh("victory.obj");
//  Bounding box
Volume box = Box(-1.0,-1.0,0.0,1.0,1.0,1.0);

// Operations:
//  Extract the wings of the Victory model
Node wings = victory.Crop(box);
//  Blend the snake-woman with the wings
Node blend = Blend(snakewoman,wings);

// Creation of the HybridTree
Tree hybridtree = Tree(blend);

// Computation of the mesh representation
return hybridtree.CreateMesh(0.05);
}
```

We implemented our meshes using a directed edge datastructure, as proposed in [10]. Although this structure is quite redundant, it enables a very fast localization of boundary edges, which is useful for crack fixing steps that occur frequently in our framework.

We build an axis aligned bounding box hierarchy so as to rapidly discard useless evaluations in empty regions of space. However the efficiency of our approach clearly depends on the structuration of the HybridTree. We try to minimize the impact of a non-optimal organization by applying an optimization process implemented in the class `Tree`, which is based on rewriting techniques. A HybridTree is completely reorganized with respect to a spatial criterion of minimizing the overlapping volume between objects. Warping nodes are also relocated at the lowest levels of the tree whenever possible.

5. Results and discussion

In this section, we present some complex models created by combining and deforming original implicit surfaces built from hundreds of primitives and meshes with thousands of triangles. Modeling was performed on a Pentium IV 2.4GHz - 1Go RAM workstation. Table 1 reports the timings corresponding to the evaluation of the final model as a triangle mesh (in seconds), as well as the overall number of triangles.

5.1. Free-form modeling

The winged snake-woman Figures 1 and 2 show blending and Boolean operations applied to implicit and mesh input models. The original snake-woman (Figure 1(a) and 2(a)) is an implicit model built from 250 spline implicit primitives blended together, which is stored in our own library of models. The body has been blended with the Igea mesh model (67,170 triangles) and the wings of the Victory

Figure	Polygonization time	Number of triangles
1(d), 2(e)	79	164,903
7(e)	70	91,961
8(d)	65	138,976
9(d)	78	110,713
10(c)	18	67,589

Table 1. Polygonization timings (seconds) and number of triangles.

of Samothrace (16,340 triangles). The mesh creation process first invokes the polygonization of the implicit snake-woman model. The Marching Cubes algorithm is used as all implicit primitives are overlapping much. The resulting mesh consists of 121,524 triangles, and took 9 seconds to generate. The head has been removed using Boolean difference with an implicit sphere primitive, and the body has been blended with the Igea model using our local meshing method. The wings have been extracted from the Victory of Samothrace mesh model by intersecting the original model with a box. The wings and the modified snake-woman model have been finally blended together using the local meshing method.

The winged Bunny Figure 7 shows a winged Stanford Bunny, which illustrates the abilities of our local blending operator on meshes. This model is defined as a blend between the mesh model of the Stanford Bunny (69,674 triangles) and the wings of the Victory of Samothrace (16,340 triangles).

The bowl The bowl in Figure 8 has been created using blending operators. The interior of the Igea mesh model has been carved using a negative potential field generated by a line segment implicit primitive. Handles built from two implicit circle primitives have been added using blending.

The spiky snake-woman Figure 9 illustrates our free-form deformation tool, applied to the snake-woman model with the Igea model as head. Before applying the deformation, this model has been first meshed as described previously. Additionally, a Boolean difference operation with an implicit cylinder primitive has been applied.

5.2. Virtual restoration of artwork

Figure 10 shows a virtual restoration process on the Igea mesh, which is a Greek artifact, using the HybridTree. We were interested in filling in the deep ridge on the right of the chin and restoring the nose, exactly as a specialist could do. We have used our blending tools to simulate cementing in a very intuitive and realistic way. First we have manually located implicit point primitives along the ridge and one at the tip of the nose. The parameters of the field functions have also been set by hand for each primitive. The primitives have been then blended with the Igea mesh model so as to produce the final mesh representation. We have built an independent subtree for the set of primitives of the chin and the another for the nose. The former has been polygonized using the Marching Cubes algorithm, as the primitives overlap much. Then, the resulting mesh has been blended with the Igea model using the local method. The same approach has been used for the nose.

5.3. Discussion

Performance Our system can handle simple implicit primitives and polygonal meshes of up to 20,000 triangles at interactive rates. Free-form deformations as well as local blending may be performed interactively in the general case. Boolean operations combining small implicit primitives or meshes compared to the overall size of the final object may also be performed at interactive rates.

The conversion step between triangles meshes and implicit surfaces is the critical limiting factor regarding computational performance. There is a need for fast polygonization algorithms as well as fast implicitization of complex meshes. Our experiments demonstrate that our optimized distance function used to implicitize a mesh speed-up computations the more significantly as the radius of influence of the triangle mesh primitive is small. For instance, the time needed to polygonize the final implicit representation of the winged Stanford Bunny model (Figure 7) drops from 210 to 70 seconds using our technique.

Nevertheless, the computation of the potential field function generated by a mesh at a given point in space remains computationally expensive. Experiments demonstrate that a field function query performed on a complex mesh can have a cost in time that is up to several hundreds times the cost of the same evaluation performed on a point primitive. Sampling the distance field on a grid as a preprocessing step would yield faster point to mesh distance queries. However the cost of storage would dramatically increase, which we wish to avoid.

Storage The HybridTree data-structure significantly reduces the amount of memory needed for storing complex models. Contrary to Level Set [22] or Adaptive Distance Fields models [15], we do not store any voxel grid or octree, which saves memory. The use of complex implicit skeletal primitives enables us to design complex shapes with a very compact representation. The snake-woman model represented in Figures 1 and 2 was created by blending a few hundred spline skeletal primitives together. The corre-

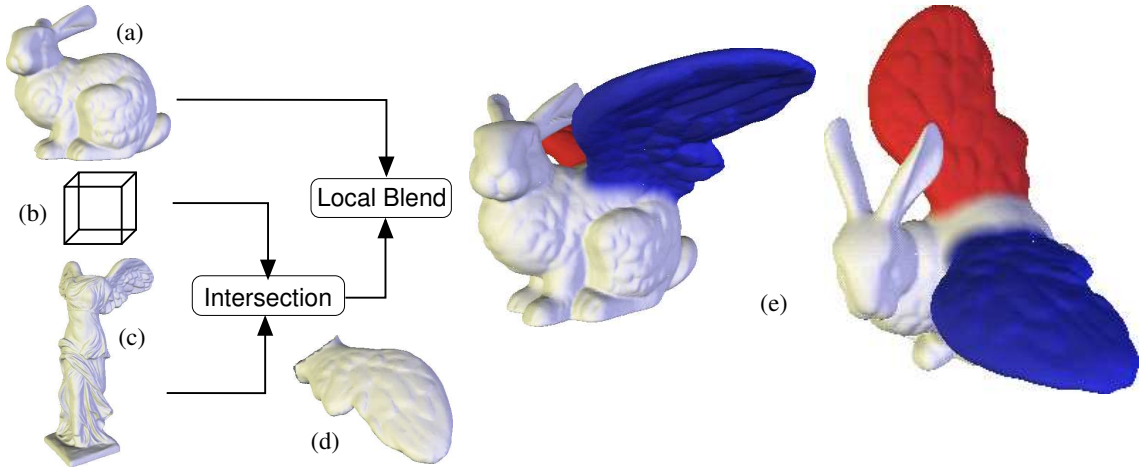

Figure 7. Modeling a winged Stanford Bunny: (a) the original Stanford Bunny mesh model, (b) the bounding volume which is used to extract the wings of the Victory of Samothrace mesh model in (c); the wings in (d) are then blended with the Bunny so as to obtain the final model in (e).

Figure 8. Creating a bowl from the Igea mesh: (a) the original Igea mesh, carved using a negative potential field generated by a line segment implicit primitive (b); (c) the handles created from two implicit circle primitives which are locally blended with the head using an implicit box so as to obtain the final model in (d).

sponding HybridTree representation takes less than 64 kilobytes in memory.

Shape control The ability to combine mesh models and skeletal implicit surfaces in a coherent framework not only extends the range of models that can be created but also permits us to have a tight control when editing our models.

The implicit surface representation enables blending of meshes of arbitrary topology and geometry. This compares favorably with other specific mesh fusion methods such as [30] or [18] that impose some geometric or topological restrictions. Moreover, our local blending technique provides fine control on the way shapes blend together. The designer may simply tune the radius of influence for mesh or implicit primitives so as to control the geometry of the blend with other objects. The implicit representation also provides means of creating negative blending between shapes, which is useful for simulating carvings. Eventually, our FFD tool enables very intuitive local deformations on hybrid models.

Conversion between implicit and mesh representations The evaluation of the HybridTree relies on conversion procedures between the implicit and the mesh representations. For example, the blending of two meshes involves their global implicitization, while free-form deformations applied to an implicit surface require its conversion into a mesh. Therefore, conversion steps occur very frequently in our system.

Regarding performance, the global conversion strategy is computationally demanding and introduces approximation errors. In the near future, we plan to investigate a bet-

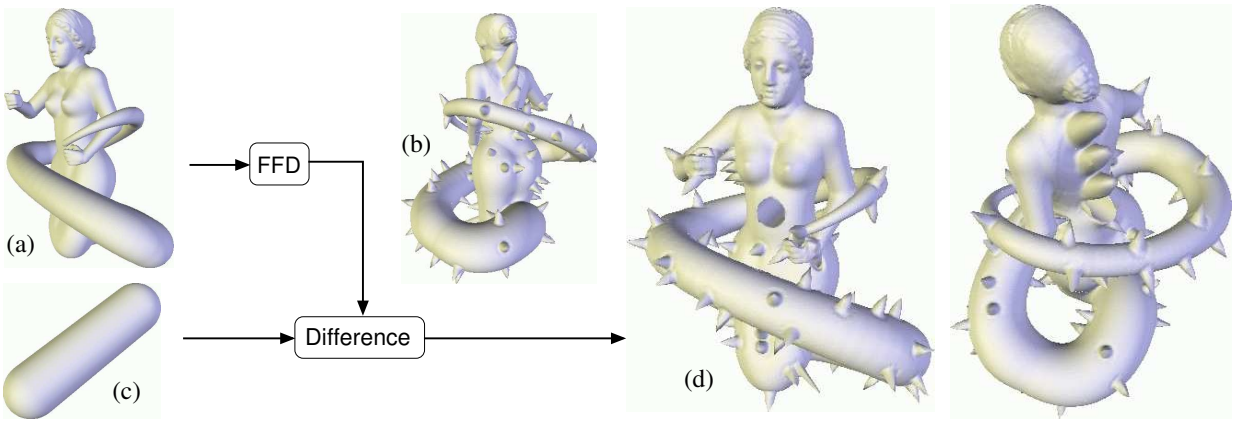

Figure 9. Free-form deformation and Boolean difference on the snake-woman (a) the snake-woman with the Igea mesh model as head; (b) result of the deformation; (c) the implicit line segment primitive subtracted from the the deformed model; (d) the final surface.

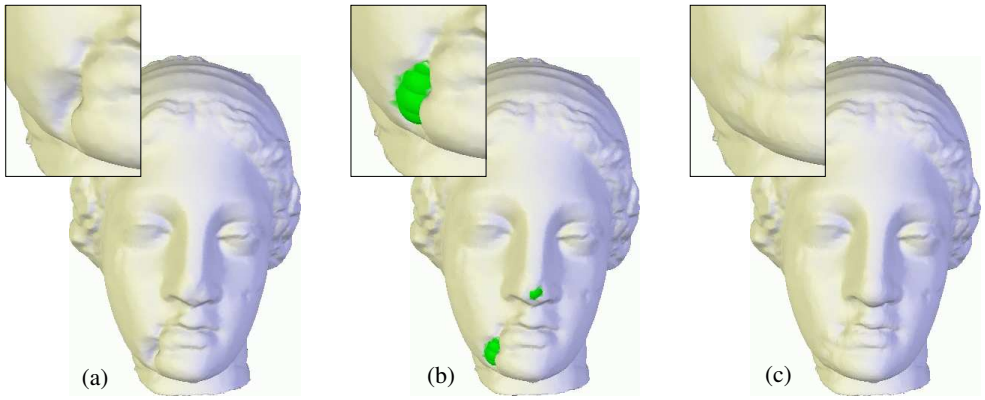

Figure 10. Virtual restoration of the Igea Greek artifact: (a) the original Igea mesh; (b) adding material; (c) partially restorized model.

ter approach that would consist in developing specific local conversion techniques. Thus, blending between two meshes would only necessitate the implicitization of the meshes in a restricted region of space. The mesh representation required by free-form deformations would also be computed only in the actually deformed regions of space. This approach would speed up the overall evaluation process and avoid some approximation errors.

6. Conclusion and future work

In this paper, we have proposed a new hybrid shape representation. Our model combines skeletal implicit and polygonal meshes into a coherent framework. The HybridTree exploits the complementary advantages of the mesh representation, useful for fast visualization and free-form deformations, and implicit surfaces, that lend themselves for Boolean, and local and global blending. These operations are performed in the most suitable representation in a totally transparent way for the user. Meshing and potential field function evaluations queries are optimized for every node of the HybridTree.

In the near future, we plan to investigate local conversion techniques so as to achieve interactive editing of large meshes and more complex skeletal implicit surfaces. We also wish to extend our HybridTree model by incorporating point cloud representations [26, 1] that have been presented recently. Dealing with point clouds would enable us to directly handle scanned objects, and make them interact with implicit surfaces and meshes. We will also investigate the automatic management of levels of detail in the HybridTree. We think that it should be possible to combine skeletal implicit primitives with levels of detail as presented in [5] with

multiresolution meshes and subdivision surfaces. Another interesting research field would be to bring unicity in the representation of the information and achieve reversibility in the evaluation process.

Acknowledgements

The authors would like to thank Raphaëlle Chaine for helpful discussions, and Tiphaine Accary for modeling the snake-woman implicit model. The model of the Victory of Samothrace was provided courtesy of Pascal Lefebvre-Albaret from Technodigit. The Bunny model was provided by the Stanford Scanning Repository and the Igea model by Cyberware Inc.

References

[1] B. Adams and P. Dutré. Interactive Boolean Operations on Surfel-Bounded Solids. *ACM Transactions on Graphics*, 22(3):652–656, 2003.

[2] V. Adzhiev, M. Kazakov, A. Pasko, and V. Savchenko. Hybrid system architecture for volume modeling. *Computers & Graphics*, 24(1):194–203, 2000.

[3] J. Ahn. Fast Generation of Ellipsoids. *Graphics Gems V*, pages 179–190, 1995.

[4] S. Akkouche and E. Galin. Adaptive Implicit Surface Polygonization using Marching Triangles. *Computer Graphics Forum*, 20(2):67–80, 2001.

[5] A. Angelidis and M.-P. Cani. Adaptive Implicit Modeling using Subdivision Curves and Surfaces as Skeletons. In *Proc. of Solid Modeling and Applications*, 2002.

[6] A. Barbier, E. Galin, and S. Akkouche. Controlled Metamorphosis of Animated Objects. In *Proc. of Shape Modeling International*, pages 184–196, 2003.

[7] A. H. Barr. Global and Local Deformations of Solid Primitives. In *Proc. of SIGGRAPH*, volume 18, pages 21–30, 1984.

[8] J.-D. Boissonnat and S. Oudot. Provably Good Surface Sampling and Approximation. In *Proc. of Symposium on Geometry Processing*, 2003.

[9] P. Borrel and D. Bechmann. Deformation of n-dimensional objects. In *Proc. of Solid Modeling and Applications*, pages 351–369, 1991.

[10] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed Edges - A Scalable Representation for Triangle Meshes. *Journal of Graphics Tools*, 3(4):1–12, 1998.

[11] S. Coquillart. Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling. In *Proc. of SIGGRAPH*, pages 187–196, 1990.

[12] B. Crespin. Implicit Free-Form Deformations. In *Proc. of Implicit Surfaces*, pages 17–23, 1999.

[13] P. Decaudin. Geometric Deformation by Merging a 3D-Object with a Simple Shape. In *Proc. of Graphics Interface*, pages 55–60, 1996.

[14] P. Decaudin and A. Gagalowicz. Fusion of 3D Shapes. In *Proc. of Computer Animation and Simulation*, pages 1–14, 1994.

[15] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively Sampled Distance Dields: A General Representation of Shape for Computer Graphics. In *Proc. of SIGGRAPH*, pages 249–254, 2000.

[16] C. Galbraith, P. Prusinkiewicz, and B. Wyvill. Modeling Murex cabritii Sea Shell with a Structured Implicit Surface Modeler. In *Proc. of Computer Graphics International*, pages 55–64, 2000.

[17] D. Johnson and E. Cohen. A Framework for Efficient Minimum Distance Computation. In *Proc. of Conf. Robotics and Automation*, pages 3678–3683, 1998.

[18] T. Kanai, H. Suzuki, J. Mintani, and F. Kimura. Interactive Mesh Fusion Based on Local 3D Metamorphosis. In *Proc. of Graphics Interface '99*, pages 148–156, 1999.

[19] L. P. Kobbelt and M. Botsch. Freeform Shape Representations for Efficient Geometry Processing. In *Proc. of Shape Modeling International*, pages 111–115, 2003.

[20] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature Sensitive Surface Extraction from Volume Data. *Proc. of SIGGRAPH*, pages 57–66, August 2001.

[21] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proc. of SIGGRAPH*, pages 181–188, 1996.

[22] K. Museth, D. E. Breen, R. T. Whitacker, and A. H. Barr. Level Set Surface Editing Operators. *ACM Transactions on Graphics*, 21(3):330–338, 2002.

[23] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level Partition of Unity Implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003.

[24] A. Pasko, V. Adzhizev, A. Sourin, and V. Savchenko. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer*, 11(8):429–446, 1995.

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

[26] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape Modeling with Point-Sampled Geometry. *ACM Transactions on Graphics*, 22(3):641–650, 2003.

[27] P. Reuter, I. Tobor, C. Schlick, and S. Dedieu. Point-based Modelling and Rendering using Radial Basis Functions. In *Proc. of ACM Graphite 2003*, pages 111–118, 2003.

[28] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Solid Geometric Models. In *Proc. of SIGGRAPH*, pages 151–160, 1986.

[29] K. Singh and R. Parent. Implicit Surface Based Deformations of Polyhedral Objects. In *Proc. of Implicit Surfaces*, 1995.

[30] K. Singh and R. Parent. Joining Polyhedral Objects using Implicitly Defined Surfaces. *The Visual Computer*, 17(7):415–428, 2001.

[31] G. Turk and J. F. O'Brien. Modelling with Implicit Surfaces that Interpolate. *ACM Transactions on Graphics*, 21(4):855–873, 2002.

[32] L. Velho, J. Gomes, and L. H. Figueiredo. *Implicit Objects in Computer Graphics*. Springer Verlag, New York, 2002.

[33] B. Wyvill, E. Galin, and A. Guy. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum*, 18(2):149–158, June 1999.