The BlobTree

Warping, Blending and Boolean Operations in an Implicit Surface Modeling System

Brian Wyvill & Andrew Guy
University of Calgary
Department of Computer Science
2500, University Dr. NW Calgary, Canada
blob@cpsc.ucalgary.ca

Eric Galin
Laboratoire L.I.G.I.M
Ecole Centrale de Lyon
Lyon, France
galin@cc.ec-lyon.fr

Abstract

Automatic blending has characterized the major advantage of implicit surface modeling systems. Recently, the introduction of deformations based on space warping and boolean operations between primitives has increased the usefulness of such systems. We propose a further enhancement which will greatly enhance the range of models that can be easily and intuitively defined with a skeletal implicit surface system. We decribe a hierarchical method which allows arbitrary compositions of models that make use of blending, warping and boolean operations. We call this structure the BlobTree. *Blending and space warping are treated in the same way as union, difference and intersection, i.e. as nodes in the* BlobTree. *The traversal of the* BlobTree *is described along with two rendering algorithms; a polygonizer and a ray tracer. We present some examples of interesting models which can be made easily using our approach that would be very difficult to represent with conventional systems.*

Key words : blending, implicit surfaces, polygonizing, raytracing, warping.

1. Introduction

The major advantage of implicit surface modeling systems has been the use of automatic blending between skeletal elements. Recent developments in such systems include the addition of boolean operations reminiscent of CSG systems [15] and space warping which provides a method of implementing deformations [7], [20].

Existing skeletal implicit surface systems do not include all these important characteristics in a single unified structure which would allow aribitrary blends between models that include boolean operations as well as warping at a local and global level.

The problem addressed by this work is how to organize blending, warping and boolean operations in a manner that will enable global and local operations to be exploited in a general and intuitive fashion.

CSG systems typically use a tree structure to describe the relationship of boolean set operations such as union, intersection and difference between half space primitives. Implicit surface systems have not evolved such an elegant way of representing the relationship between blending, warping and boolean operations on skeletal element primitives.

If implicit surface primitives were treated in a homogeneous manner, then blending would occur globally, *i.e.* between all primitives, whether this was desired or not. In this work we structure our implicit surface models using a tree which includes blending, warping and boolean operations. We refer to the structure as the *BlobTree*.

In this paper we present the following contributions:

- Complex models can be built with a small number of primitives using arbitrary combinations of blending, warping and boolean operations (section 3).
- Prototyping by polygonization (section 4).
- Efficient direct ray tracing for high quality visualizations (section 5).
- Joins between primitives can vary continuously between blend and union (section 3.3).
- Warping either globally or locally controlled. Warps can be arbitrary functions, examples given include the Barr deformations and affine transformations implemented as space warps (section 3.4).

We also give examples of some complex shapes built from a few primitives using the operations (*blend, warp, intersection, union, difference*) organized in the *BlobTree*. These models would be very difficult to build using conventional modeling techniques.

1.1. Previous Work

Jim Blinn introduced the idea of modeling with skeletal implicit surfaces as a side effect of a visualization of electron density fields [2]. Such models have various desirable properties including the ability to blend with their close neighbors. These models have been given a variety of names in particular: *Blobby Molecules* (Blinn), *Soft Objects* (Wyvill) [21] and *MetaBalls* (Nishimura) [14]. Jules Bloomenthal pointed out that these models could be grouped under the more general heading of *implicit surfaces*, defined as the point set: $F(P) = 0$ [3].

A system which includes blending, warping and boolean operations was introduced by Pasko & *al.* [15], who based the operations on functionally defined primitives. This functional approach, called the theory of *R–Functions*, also encompasses an analytical description of blending and boolean operations. Our system is similar except that we use skeleton based primitives [4] instead of functional primitives, structured using the *BlobTree*. As yet we have not done a comparison of these two approaches, however working with skeletons has enabled us to build an interactive modeler providing intuitive control to the operator.

Our warping uses the deformations introduced by Barr [6], *i.e.* the operations of *twist*, *taper* and *bend*.

Our polygonizer is a modification on the software described in [21]. We used a uniform space subdivison algorithm rather than an adaptive algorithm since it was readily available. Adative polygonizers have been described for both CSG systems [19] and implicit surface systems [18].

2. Implicit Surface Models

Skeletal implicit surface models are constructed from combinations of geometric skeletal elements. An implicit model A is generated by summing the influences of N_A skeletal elements, whose potential field will be denoted F_{A_i}, which together define a scalar field F_A. The global potential field $F_A(x, y, z)$ of an object, we call the implicit function, may be defined as:

$$F_A(x, y, z) = \sum_{i=1}^{i=N_A} F_{A_i}(x, y, z)$$

The surface of the object may be derived from the implicit function $F_A(x, y, z)$ as the points of space whose value equals a threshold denoted by T_A.

$$\Sigma_A = \{M(x, y, z) \in \mathbb{R}^3, F_A(x, y, z) = T_A\}$$

Each component of the implicit function $F_A(x, y, z)$ may be split into a distance function $d_{A_i}(x, y, z)$ and a field function $f_{A_i}(r)$, where r stands for the distance to the skeleton [1]. We will refer to the following notation:

$$F_{A_i}(x, y, z) = f_{A_i} \circ d_{A_i}(x, y, z)$$

Note that traditionally in implicit surface systems, skeletal elements do not have any structure, which means that all elements blend in the same way. In this work we introduce a new approach using a tree structure to store the relationships between the skeletal elements.

Visualizing the surfaces can be done either by direct ray tracing using an algorithm similar to that described in [13] (a more recent technique is described in [12]) or by first converting to polygons [21].

3. The Blob Tree

In our system models are defined by expressions which combine implicit primitives and the operators $\cup$ (union), $\cap$ (intersection), $-$ (difference), $+$ (blend), $\diamond$ (super–elliptic blend), and w (warp). The *BlobTree* is not only the data structure, built from these expressions but also a way of visualizing the structure of the models. The operators listed above are binary with the exception of warp which is a unary operator. In fact it is more efficient to use n–ary rather than binary operators.

3.1. Nodes of the BlobTree

Throughout this paper, $\mathcal{T}$ will refer to the *BlobTree*. $\mathcal{N}$ will be a node in the tree, and its left and right leaves will be referred to as $\mathcal{L}(\mathcal{N})$ and $\mathcal{R}(\mathcal{N})$ respectively. Thus, the field created by a node in the tree will be denoted as $F(\mathcal{N})$. In the following sections we describe the nodes of the tree.

3.2. Boolean operators

We recall that the union and intersection of primitives may be respectively defined as:

$$\begin{cases} F_{A\cup B} = max(F_A, F_B) \\ F_{A\cap B} = min(F_A, F_B) \end{cases}$$

The difference operator may be expressed in terms of a negation and an intersection: $F_{A-B} = F_{A\cap(-B)}$.

However, those functions show discontinuities, and several other C^n functions have been proposed [15].

$$\begin{cases} F_{A\cup B} = \left(F_A + F_B + \sqrt{{F_A}^2 + {F_B}^2}\right)\left({F_A}^2 + {F_B}^2\right)^{\frac{n}{2}} \\ F_{A\cap B} = \left(F_A + F_B - \sqrt{{F_A}^2 + {F_B}^2}\right)\left({F_A}^2 + {F_B}^2\right)^{\frac{n}{2}} \end{cases}$$

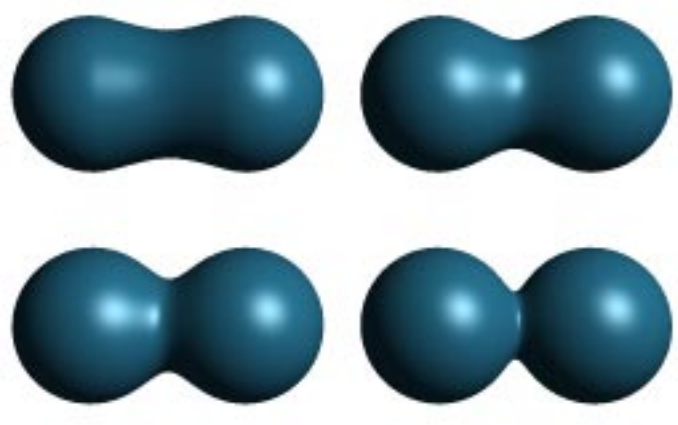

Figure 1. *Super–elliptic blending*

3.3. Blending operators

Although blending two primitives is generally performed by adding their potential fields, super–elliptic blending has been proposed [16] so as to achieve a large range of blends. Throughout the remainder of this paper standard blending will be referred to as $A + B$ (*i.e.* sum the functions F_A and F_B). So called generalized blending will be denoted as $A \diamond B$ and may be defined as:

$$F_{A \diamond B} = \left({F_A}^n + {F_B}^n \right)^{\frac{1}{n}}$$

It is interesting to point out the following properties:

$$\begin{cases} \lim\limits_{n \to +\infty} \left({F_A}^n + {F_B}^n \right)^{\frac{1}{n}} = max(F_A, F_B) \\ \lim\limits_{n \to -\infty} \left({F_A}^n + {F_B}^n \right)^{\frac{1}{n}} = min(F_A, F_B) \end{cases}$$

Moreover, this generalized blending is associative, *i.e.* $F_{(A \diamond B) \diamond C} = F_{A \diamond (B \diamond C)}$. The standard blending operator $+$ proves to be a special case of the super–elliptic blend with $n = 1$. When n varies from 1 to infinity, it creates a set of blends interpolating between blending $A + B$ and union $A \cup B$ (figure 1). Figure 3 shows the nodes to be binary or unary, in fact the binary nodes can easily be extended using the above formulation to n–ary nodes.

Following Pasko's functional representation [15], another generalized blending function may be defined as:

$$F_{A \diamond B} = \left(F_A + F_B + \alpha \sqrt{{F_A}^2 + {F_B}^2} \right) \left({F_A}^2 + {F_B}^2 \right)^{\frac{n}{2}}$$

When $\alpha \in [-1, 1]$ varies from -1 to 1, it creates a set of blends interpolating the union and the intersection operators. However, this operator is no longer associative which is incompatible with the definition of n–ary operators.

3.4. Warp operators

A useful tool in our system is the ability to distort the shape of a surface by warping the space in its neighborhood. A warp is a continuous function $w(x, y, z)$ that maps $\mathbb{R}^3$ into $\mathbb{R}^3$. Sederberg provides a good analogy for warping when describing free form deformations [17]. He suggests that the warped space can be likened to a clear, flexible plastic parallelpiped in which the objects to be warped are embedded. A warped element may be defined as:

$$F_{A_i}(x, y, z) = f_{A_i} \circ d_{A_i} \circ w_{A_i}(x, y, z)$$

Throughout this paper, a warp function will be denoted as $w(x, y, z)$. A warped element may be fully characterized by the distance to its skeleton $d_{A_i}(x, y, z)$, its potential function $f_{A_i}(r)$ and eventually its warp function $w_{A_i}(x, y, z)$. Such elements will be denoted as: $\{f_{A_i}(r), d_{A_i}(x, y, z), w_{A_i}(x, y, z)\}$.

In some cases we may want to compute the gradient of the potential field, *e.g.* for normal computation. We recall that the gradient of a C^1 bijective function $F(x, y, z)$ is:

$$\nabla F(P) = f' \circ d \circ w(P) \times J_w^{-1}(P) \times \nabla d \circ w(P)$$

Thus, for each component A_i, we need to be able to compute both $w_{A_i}(P)$ and the Jacobian $J_{w_{A_i}}(P)$ for each warp function $w(x, y, z)$.

3.4.1. Affine Transformations

The affine transformations can be applied as warp functions. Although the effect of this is no different from applying the same transformations to the skeletons in normal space, the advantage is that a skeletal element can be defined in its canonical position and orientation and a warp used to transform it into the world space, as is common practice in many ray tracers. The advantage in our system is that warping and transformations may be treated in a consistent fashion.

Affine transformations can also be composed with other warps. For efficiency, consecutive affine transformations can be concatenated.

3.4.2. Barr Deformations

The Barr deformation operators, *twist*, *taper* and *bend* have been implemented as warps. As indicated in [6] the deformations can be nested producing models such as shown in figure 5. In the figure three blended cylinders have been twisted and tapered, each sample point $P(x, y, z)$ is first transformed into warp space using $w_{Taper} \circ w_{Twist}(P)$.

Barr applies the warp function to wireframe models, and thus uses the warp function w_{A_i} to change the coordinates

Figure 2. *Implicit surface model including boolean operations and warping*

of the vertices. In our case, we wish to warp space, thus we use the inverse warp function $(w_{A_i})^{-1}$.

The inverse twisting operation is a twist with a negative angle, and the inverse tapering operation is a taper with the inverse shrinking coefficient. The inverse of bend cannot be produced by modifying the bend parameters (see [6] for details of the inverse of the bend operation).

3.4.3. Generic Warps

In practice, any kind of warp may be used, but the Jacobian may be difficult or impossible to compute if the warping is not bijective. In that case, we rely on a discrete approximation of the gradient (see section 3.8).

3.5. The leaf nodes

The leaf nodes of the *BlobTree* contain skeletal implicit primitives. The following have been are defined:

- Points: ellipsoids and super-ellipsoids.
- Lines: cylinders capped with hemispherical ends.
- Circles: torii.
- Polygons: offset surfaces.
- Polyhedra: offset volumes.
- Plane: offset plane.

3.6. Example of a BlobTree

An example of a model built from these nodes is shown in figure 3. Starting at the left most leaf node, the two spheres are blended (+ node). A horizontal plane is intersected with

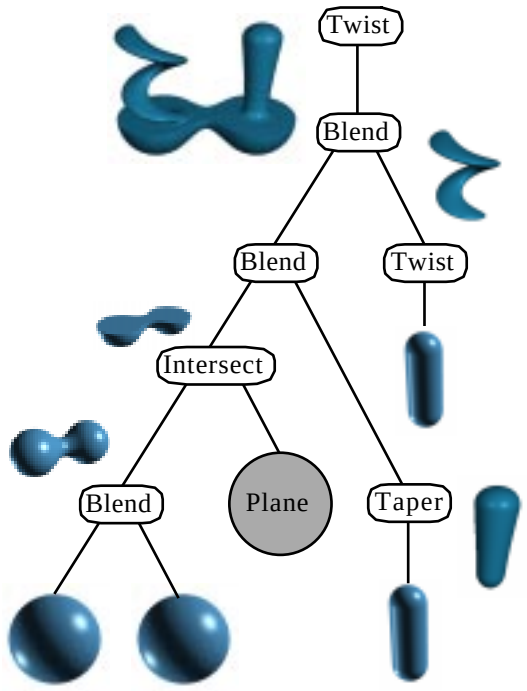

Figure 3. *The* blobtree *for figure 2*

the model leaving only the lower half of it (∩ node). This model is then blended with a *tapered* cylinder. A second cylinder has a *twist* applied to it and it is then blended to the result. Finally a global *twist* is applied to the entire tree to produce figure 2. As can be seen from this example nodes can be added to the *BlobTree* in any order.

3.7. Traversing the Blob Tree

Polygonization and raytracing algorithms need to evaluate the implicit field function at a large number of points in space. The function $F(\mathcal{N}, M)$ returns the field value for the node $\mathcal{N}$ at the point M, which depends of the type of the node:

function $F(\mathcal{N}, M)$:

1. Primitive : $F(M)$.
2. Warp : $F(\mathcal{L}(\mathcal{N}), w(M))$.
3. Blend : $F(\mathcal{L}(\mathcal{N}), M) + F(\mathcal{R}(\mathcal{N}), M))$
4. Union : $\max(F(\mathcal{L}(\mathcal{N}), M), F(\mathcal{R}(\mathcal{N}), M))$
5. Intersection : $\min(F(\mathcal{L}(\mathcal{N}), M), F(\mathcal{R}(\mathcal{N}), M))$
6. Difference : $\min(F(\mathcal{L}(\mathcal{N}), M), -F(\mathcal{R}(\mathcal{N}), M))$

3.8. Computing the Normal and Surface Properties

Exact normals may be computed given the normal of each primitive. The normal for a blend node is the weighted average (*i.e.* weighted by the value of the implicit function) of the normals computed from each of the child nodes. For a CSG node, normals are computed from the appropriate child. If the node is a warp, a method for computing the normal must be provided with the warp function. This will either use the Jacobian and the normal from the child node or a numerical approximation. In the case of the Barr warps, details of the Jacobians are given in [6]. A numerical approximation to the gradient is:

$$\nabla F(x,y,z) = \frac{1}{2\delta}\begin{pmatrix} F(x+\delta,y,z) - F(x-\delta,y,z) \\ F(x,y+\delta,z) - F(x,y-\delta,z) \\ F(x,y,z+\delta) - F(x,y,z-\delta) \end{pmatrix}$$

Both techniques have pros and cons. Discrete evaluation requires the computation of the potential field at six locations around the given points, whereas the exact computation involves the computation of the Jacobians of the warps which may be expensive (*e.g.* the Jacobian of the twist function requires a sine and cosine evaluation).

For attributes such as color, reflection, refraction coefficients *etc.* the resultant is computed as follows: for a blend node the resulting attribute values are calculated as the weighted average of the attributes of the children. For a CSG node the attributes are taken from the attribute of the appropriate child (*e.g.* for union take $max(\mathcal{L}(\mathcal{N}), \mathcal{R}(\mathcal{N}))$).

Currently the warp nodes do not alter the attribute values although this possibly opens up an avenue for some special effects with warped texture spaces in the future.

4. Polygonization using the BlobTree

Polygonization is used for prototyping models; after the polygon mesh has been produced it can be viewed on any graphics workstation. Adding boolean operations to implicit surface models introduces discontinuities or junctions between surface elements that are important to the resulting model. To reproduce these discontinuities in the polygon mesh a post–processing step is performed.

A uniform space subdivision algorithm [21] is applied to the *BlobTree* to produces a polygon mesh. For trees that contain CSG operations, a post–processing step is applied to the resulting mesh [20]. Other polygonization algorithms could also be used and the same CSG post–processing step performed.

The CSG post–processing algorithm finds edges in the polygon mesh that span discontinuities produced by the CSG operations. These edges occur when the field value is contributed by different nodes in the tree. Once these edges have been found they are subdivided and the newly created vertex is moved on to the surface at the junction using an iterative algorithm proposed in [20] (figure 4). This CSG post–processing algorithm needs the field value and normal from each of the contributing nodes at a number of points in space. This requires a path to the node that is the contributor for a given point in space. Finding this path also involves a tree traversal algorithm as follows:

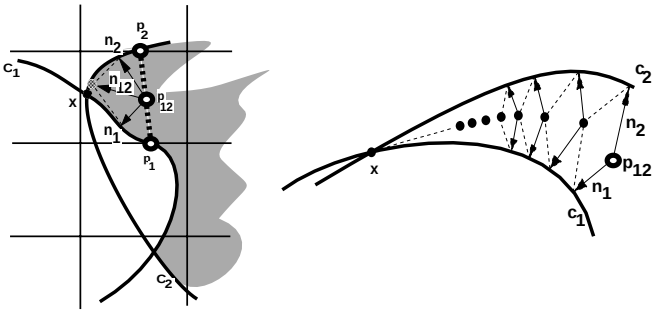

Figure 4. *Approximating the intersection of two implicit contours. Left:* $\mathbf{p}_{12}$ *is a first guess. Right: iterating to get a more accurate approximation of* $\mathbf{x}$*.*

The function $G(\mathcal{N}, M)$ returns a tuple of the path to the contributing node and the field value for the point M when applied to the node $\mathcal{N}$.

function $G(\mathcal{N}, M)$:

1. If $\mathcal{N}$ is a Primitive return $(\mathcal{N}, F(M))$
2. If $\mathcal{N}$ is a Warp, call the function on the child node, $(P_\mathcal{L}, F_\mathcal{L}) = G(\mathcal{L}(\mathcal{N}), w(M))$ and then return $(\text{prepend}(\mathcal{N}, P_\mathcal{L}), F_\mathcal{L})$, where prepend joins the node $\mathcal{N}$ on to the path $P_\mathcal{L}$
3. Otherwise the function is called on the child nodes, $(P_\mathcal{L}, F_\mathcal{L}) = G(\mathcal{L}(\mathcal{N}), M)$ & $(P_\mathcal{R}, F_\mathcal{R}) = G(\mathcal{R}(\mathcal{N}), M)$ and the return value depends on the type of the node.
4. If $\mathcal{N}$ is Blend:
   - if $F_\mathcal{L} = 0$ return $(\text{prepend}(\mathcal{N}, P_\mathcal{R}), F_\mathcal{R})$
   - if $F_\mathcal{R} = 0$ return $(\text{prepend}(\mathcal{N}, P_\mathcal{L}), F_\mathcal{L})$
   - else return $(\mathcal{N}, F_\mathcal{L} + F_\mathcal{R})$
5. If $\mathcal{N}$ is Union:
   - if $F_\mathcal{L} > F_\mathcal{R}$ return $(\text{prepend}(\mathcal{N}, P_\mathcal{L}), F_\mathcal{L})$
   - else return $(\text{prepend}(\mathcal{N}, P_\mathcal{R}), F_\mathcal{R})$
6. If $\mathcal{N}$ is Intersection:
   - if $F_\mathcal{L} < F_\mathcal{R}$ return $(\text{prepend}(\mathcal{N}, P_\mathcal{L}), F_\mathcal{L})$

Figure 5. *Blended cylinders warped using twist and taper.*

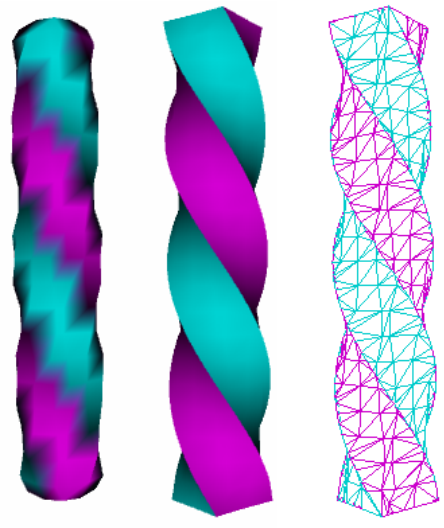

Figure 6. *Twisted rod of intersecting planes, right images include CSG post-processing*

- else return $(\text{prepend}(\mathcal{N}, P_{\mathcal{R}}), F_{\mathcal{R}})$

7. If $\mathcal{N}$ is Difference:
   - if $F_{\mathcal{L}} < -F_{\mathcal{R}}$ return $(\text{prepend}(\mathcal{N}, P_{\mathcal{L}}), F_{\mathcal{L}})$
   - else return $(\text{prepend}(\mathcal{N}, P_{\mathcal{R}}), -F_{\mathcal{R}})$

Note that the field value will be the same as the one returned by the standard tree traversal function (see section 3.7) and the function may produce a path to a non–leaf node.

Once the paths to the contributor nodes for an edge are found they can be compared. If they are the same no further action is required. If they differ the post–processing algorithm is applied to create a new vertex on the surface. The new vertex is then inserted into the mesh and new triangles and edges are created to ensure mesh coherence as in [20].

This post–processing requires the field value and the normal to be calculated from a given path. To find the field value at a point the path is traversed: any warp nodes are applied to the point and the field value is calculated from the last node. As in section 3.8 the normals can be calculated exactly or a numerical approximation can be used. If exact normals are required the normal calculated from the last node in the path must be transformed by the Jacobian for each of the warp nodes in the reverse order of entry in the path.

The object in figure 6 consists of 6 intersected planes that form a rod with a square cross section, this is then twisted along its axis. The left–hand image has not had the post–processing step applied. The improvement is significant and important for prototyping objects that use CSG operations.

5. Ray Tracing The BlobTree

Several methods exist for rendering implicit surfaces. Indirect methods such as polygonization generate a set of polygons that approximate the surface to a given tolerance. However, polygonization might not be guaranteed and may not detect disconnected or detailed sections of the surface. Moreover with complex surfaces requiring a very fine polygon mesh, the computation time for polygonization can be comparable to the time taken to ray trace the same scene. Polygonization is still a valid and useful prototyping method since a polygon mesh of arbitrary density can be produced for later viewing in real time.

Direct ray tracing requires the computation of the intersection of a ray and the surface. Let $F(x, y, z) = 0$ be the implicit equation and $r(t) = r_0 + \Delta t$ be the parameterized ray. Intersections occur at points $r(t)$ such that $F \circ r(t) = 0$.

Interval analysis finds solutions by defining the function $F \circ r(t)$ and its derivative over intervals, and uses interval analysis to bound those functions. [1]

In our current implementation, we rely on Lipschitz techniques [13]. A function is said to be Lipschitz if and only if the magnitude of its derivative remains bounded, any bound will be further denoted as L. The L–G equipotential surface

[1] In fact, we have also implemented a ray tracer whose ray intersection scheme is based on interval arithmetic; a comparison of its performance with the L–G ray tracer is left to future research

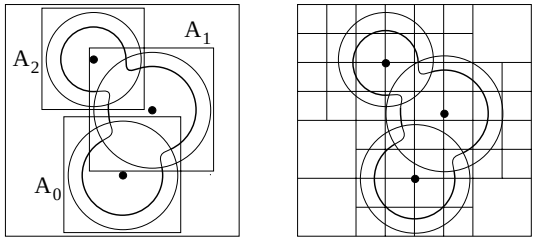

Figure 7. *Space subdivision* vs. *bounding volume subdivision*

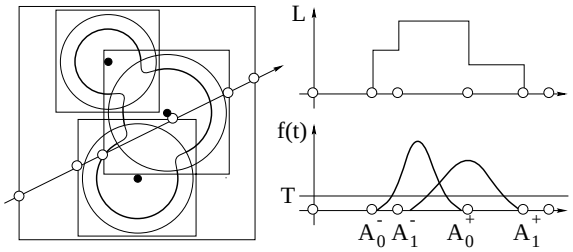

Figure 8. *Interval creation through bounding boxes ray tracing*

tracking is based on a recursive subdivision technique. L and G bounds provide criteria for determining the existence of none, one or possibly several roots in a given interval.

Tighter criteria using both first and second derivatives have been proposed [9]. However those were not implemented because it involves the computation of the Laplacian of warp functions which would only be possible for a certain class of these functions.

5.1. Evaluating the implicit function

The L–G surface techniques require two fundamental procedures in the core of the ray tracer which are function and derivative evaluation of the potential field at a point of space.

The evaluation of the intensity at a given point of space is implemented as a recursive tree traversal scheme, which is the same as the one used in the polygonization algorithm.

The computation of the derivative may be performed in two ways, either using an approximate discrete evaluation, or using exact formulation of the derivative.

5.2. Efficiency concerns

There are several techniques for speeding up the surface intersection algorithm:

- in general, finding accurate Lipschitz bounds speed up the surface intersection;
- checking which elements contribute to the global potential field at a given point, and disabling non contributing elements along a ray also speed up the *BlobTree* traversal.

Space subdivision techniques using cubic voxels or octrees may cull parts of space with no contributing elements or where no piece of the surface may be found [13], [9].

We use a nested bounding volume scheme, that yields a precise space occupancy description of the overall implicit surface model. We have not yet compared this technique with other spatial subdivision algorithms such as voxels or octrees.

A bounding box is associated with each primitive and each node. Each leaf bounding box keeps track of the Lipschitz bound of the primitive. The node bounding boxes combine the Lipschitz bounds of their leaves according to the node operator. The ray is first intersected with the bounding boxes to check contributing elements along its path. This technique fits the internal *BlobTree* representation of the objects.

5.2.1. Bounding box construction

The bounding boxes of the primitives are defined as the bounding box of their volume of influence in space. Therefore, the bounding box of a *union* or *blending* node is defined as the bounding box of the union of the bounding boxes of their children. The bounding box of an *intersection* is the intersection of the bounding boxes. Because of its definition, the bounding box of the *difference* may be computed with the previous rules.

Warp nodes deserve a special case. Let $\mathcal{B}$ be the bounding box of a node $\mathcal{N}$, and w the warp function applied to that node. Whenever warp functions are continuous, we have the following:

$$\mathcal{N} \subset \mathcal{B} \Rightarrow w(\mathcal{N}) \subset w(\mathcal{B})$$

Our warp functions conform to the above. The computation of the bounding box of $\mathcal{B}(w(\mathcal{B}))$ is different for each warp function.

5.2.2. Ray traversal

The ray surface intersection algorithm is modified as follows. First, we compute the intersection of the ray with the

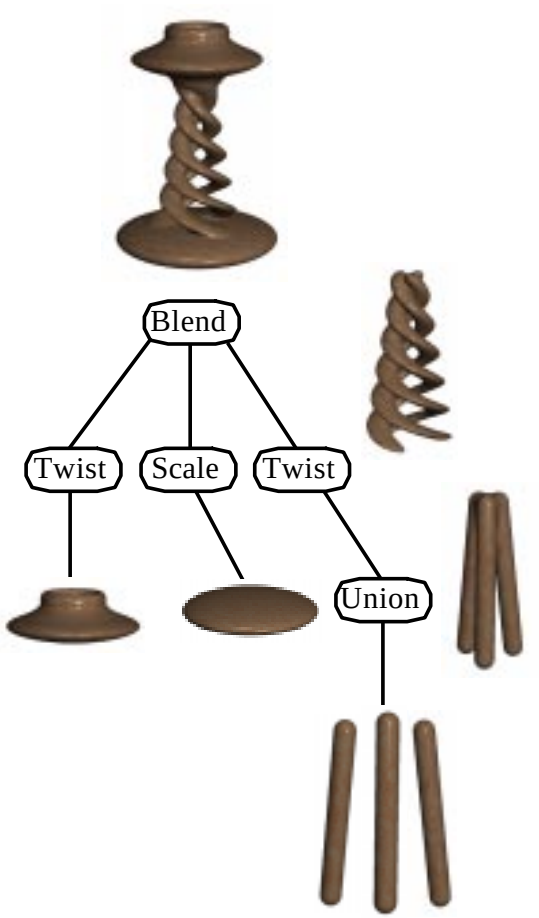

Figure 9. *The* BlobTree *which represents the candlestick in Figure 10*

bounding box hierarchy so as to generate a set of intervals. Each interval keeps track of its contributing element.

The *BlobTree* bounding box traversal is performed recursively. Given a node $\mathcal{N}$ in the tree, we check the intersection of the ray with its box. Several cases arise :

- if no intersection occurs, the recursion ends;
- if the node is a leaf, *i.e.* a primitive, the box intersection interval is stored with a reference to the contributing element;
- if the node is a binary node, *i.e.* a union, intersection, difference or blending node, the ray intersection is recursively performed on the children of that node;
- if the node is a warp node (but not a rotation, scaling or translation), then the box intersection interval is stored with a reference to the blob subtree.

Figure 10. *A scene consisting of implicit surface models using the* BlobTree

When a warp node is encountered, the bounding boxes of its descendants may no longer be used (except in the case of affine transformations). The reason why we store a reference to the blob subtree when the node is a warp operator is that the warp function may transform the ray into a curved path. In our current implementation, *affine transformations* are defined as a special subclass of warp functions (as mentioned in 3.4.1), and in that case, the tree traversal can be achieved. We are currently investigating the possibility of subdividing this class of bounding box.

6. Results

The scene in figure 6 has been entirely built with *BlobTrees*. The wooden part of the hour glass is defined as a super–elliptic blend between two polygon based primitives and three twisted super–ellipsoids. The glass bulb is defined as two blended spheres, with a slight tapering applied at the top and the bottom.

The candlestick involves seven primitives: we used the union of three cylinders to avoid blending between them,

Figure 11. *A Stamingo*

and applied a twist to this union. The top of the candlestick uses super–elliptic blending with $n = 4$, so as to avoid an amorphous shape, but keep the junctions smooth. The candle itself is a twisted super–elliptic cylinder. Figure 9 shows the *BlobTree* which represents the candlestick. The shelf and the table are also *BlobTrees*, with unions of elements based on polygonal skeletons. The image was rendered with the L–G ray tracer, and the normals where computed with Jacobian computation [6].

Figure 11 which shows a *Stamingo* (cross between a stork and a flamingo) was ray traced using interval arithmetic[5]. It is built from 15 cylinders, 5 ellipsoids, 3 tapers, 5 bends and a number of affine transformations. The tail is made from a tapered cylinder that has then been bent. The head is composed of an ellipsoid, the plumage and a tapered line for the beak; all blended together. An ellipsoid was tapered and bent twice to form the plumage. To finish off the head, two eyes where added using union. The neck was shaped using two bends, note that the head is also warped by these bends so it remains attached to the neck.

7. Conclusions and Future Work

In this paper we have described our work on generalizing an implicit surface system to treat blending, warping and CSG operations in an homogeneous manner. This approach enables complex models to be described which would be very difficult to design using other methods. Local and global space warps, particularly those based on the Barr deformations are very useful in modeling and work well with blends and CSG operations, simplifying the design of a wide variety of models.

Warp functions can change the shape of the space in which the models are embedded. The hierarchical structure of the *BlobTree* would allow the introduction of warp functions which affect texture and color spaces so that global and local texture coordinates could also be warped.

[10], [11], [8] have suggested a graph relationship between elements, where nodes represent implicit surfaces and edges represent the blending relationship, as a method of solving the unwanted blending problem. This could be combined with the *BlobTree* by labeling the graph edges so that they specify the type of join between primitives. However where the unary warping operator would fit into this scheme is unclear.

8. Acknowledgements

Acknowledgements have been omitted for the blind review process.

The authors would like to thank the following researchers; Kees van Overveld (Phillips Corps. and University of Eindhoven) especially for figure 4, Geoff Wyvill (University of Otago), and Jules Bloomenthal (Microsoft) for their very considerable contributions to this work over the years, and John Cleary (Unveristy of Waicato) for some helpful remarks. We would also like to thank Samir Akkouche for proof reading an earlier version of this paper. This work is partially supported by grants from the Natural Sciences and Engineering Research Council of Canada and also partially supported by grants from the *Centre Jacques Cartier*.

References

[1] C. Blanc and C. Schlick. Extended field functions for soft objects. In *Implicit Surfaces '95*, pages 21–32, Apr. 1995.

[2] J. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1:235, 1982.

[3] J. Bloomenthal. Polygonisation of Implicit Surfaces. *Computer Aided Geometric Design*, 4(5):341–355, 1988.

[4] J. Bloomenthal and B. Wyvill. Interactive Techniques for Implicit Modeling. *Computer Graphics*, 24(2):109–116, 1990.

[5] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[6] R. Cook. Global and Local Deformations of Solid Primitives . *Computer Graphics (Proc. SIGGRAPH 84)*, pages 21–30, July 1984.

[7] B. Crespin, C. Blanc, and C. Schlick. Implicit sweep objects. In *Eurographics '96*, volume 15, pages 165–174, Aug. 1996.

[8] E. Galin and S. Akkouche. Shape constrained blob metamorphosis. In *Second Eurographics Workshop on Implicit Surfaces, Eindhoven, October 1996)*, pages 9–23. Eurographics, 1996.

[9] J.-D. Gascuel. Implicit patches: An optimised and powerful ray intersection algorithm. In *Implicit Surfaces '95*, Apr. 1995.

[10] M.-P. Gascuel. An Implicit Formulation for Precise Contact Modeling Between Flexible Solids. *Computer Graphics (Proc. SIGGRAPH 93)*, pages 313–320, August 1993.

[11] A. Guy and B. Wyvil. Controlled blending for implicit surfaces. In *Implicit Surfaces '95*, Apr. 1995.

[12] J. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer (in press)*, December 1996.

[13] D. Kalra and A. Barr. Guaranteed Ray Intersections with Implicit Functions. *Computer Graphics (Proc. SIGGRAPH 89)*, 23(3):297–306, July 1989.

[14] H. Nishimura, A. Hirai, T. Kawai, T. Kawata, I. .Shirakawa, and K. Omura. Object Modelling by Distribution Function and a Method of Image Generation. *Journal of papers given at the Electronics Communication Conference '85*, J68-D(4), 1985. In Japanese.

[15] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 2(8):429–446, 1995.

[16] Ricci. Constructive Geometry for Computer Graphics. *computer journal*, 16(2):157–160, May 1973.

[17] T. Sederberg and S. Parry. Free Form Deformation of Solid Geometric Models. *Computer Graphics (Proc. SIGGRAPH 86)*, 23(3):151–160, August 1986.

[18] L. Velho. Simple and efficient polygonization of implicit surfaces. *Journal of Graphics Tools*, 1(2):5–24, 1996.

[19] J. Woodwark and A. Bowyer. Better and faster pictures from solid models. *Computer Aided Engineering Journal*, 3(1):17–24, February 1986.

[20] B. Wyvill and K. van Overveld. Polygonization of Implicit Surfaces with Constructive Solid Geometry. *Journal of Shape Modelling*, 2(4):257–273, 1996.

[21] G. Wyvill, C. McPheeters, and B. Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.