

Incremental Techniques for Implicit Surface Modeling

Eric Galin & Samir Akkouché
Laboratoire d'Informatique Graphique Image et Modélisation
Ecole Centrale de Lyon
B.P. 163, 69131 Ecully Cedex

e-mail : [galin-samir]@cc.ec-lyon.fr

Abstract

This paper describes an incremental polygonization technique for implicit surfaces built from skeletal elements. Our method lends itself for an interactive modeling system as the mesh is updated locally in regions of space where changes in the potential field occurred. We rely on an octree decomposition of space combined with Lipschitz conditions to recursively subdivide cells until a given level of precision is reached.

Timings show that our incremental algorithm dramatically speeds up the overall polygonization process for complex objects.

Keywords : implicit surfaces ; interactive modeling ; polygonization ; topology.

1. Introduction

Implicit surfaces have drawn a lot of attention for the past few years. They have been successfully used in the design of smooth objects of arbitrary topology and geometry, as well as in descriptive and physically based animation systems.

An implicit surface Σ is mathematically defined as a set of points in space $M(x, y, z)$ that satisfy the equation $f(x, y, z) = 0$. Thus, visualizing implicit surfaces typically consists in finding the zero-set of f , which may be performed either by polygonizing the surface or by direct ray-tracing.

Despite several accelerated schemes [21, 7], ray-tracing algorithms do not lend themselves for interactive visualization whereas polygonization provide a fast approximation of the surface.

Modeling with implicit surfaces proves to be somewhat difficult for the shape of the resulting surface may not be always anticipated. For example, unpredictable changes in the topology or phantom elements may appear [16] when modifying some parameters characterizing the potential field f .

Only the displayed approximation can alert the animator of unwanted artifacts.

In this paper, we address the polygonization of implicit surfaces built from elements whose regions of influence are bounded in space. The implicit surface is interactively designed by adding or removing elements, or changing some of their parameters. It is worth pointing out that most of the surface remains unchanged during a modification, which is also known as *shape coherence*.

We propose an incremental method based on a recursive decomposition of space that focuses on regions where changes in the potential field f occurred so as to interactively update the surface meshing where needed.

Even though the aim of this work is to provide a fast and accurate approximation of the implicit surface Σ , we will focus on the creation of cells straddling the surface and propose to use an octree decomposition of space. Eventually, a configuration type may be assigned to the straddling cells of the octree and we rely on existing techniques to create the list of polygons [20, 4] and handle topological ambiguities [18].

The paper is organized as follows. In section 2, we provide a short overview of existing polygonization schemes. In section 3, we recall the basic features of implicit surfaces built from skeletal elements. In section 4, we address the recursive creation of the octree structure and present our incremental algorithm. Eventually, we provide timings in section 5.

2. Related work

Several classifications have been proposed in the literature. Topological consistency requires that a polygonal approximation should be consistent with itself, *i.e.* without holes or disconnected vertices, edges or faces, whereas topological correctness implies that the polygonal approximation should be homeomorphic to the implicit surface.

2.1. Polygonization by surface sampling

A first class of polygonization algorithms interrogate implicit surfaces through spatial sampling. They divide space into cells and search for only those cells that intersect the implicit surface. Surface tracking techniques [20, 11] partition space into cells, and recursively find straddling cells among its neighbors until all neighbors have been checked.

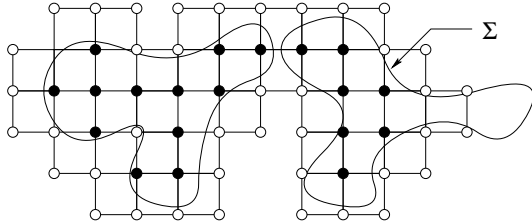


Figure 1. Surface tracking techniques may miss small geometrical features, or yield a polygonal approximation whose homotopy type differs from the implicit surface's

Unfortunately, those techniques do not guarantee topology and are slow since they need to start over every time a new element is added or deleted. Moreover, surface tracking techniques require a seeding for each surface component which is often difficult to provide. Even with a correct initialization, the algorithm may miss small features (figure 1).

Some cellular polygonization schemes can guarantee that the topology is contained in the union of a set of small cells. Both the Lipschitz condition [9] and interval analysis [15] can guarantee that the implicit surface does not pass through some cells. Ambiguous cells are recursively subdivided until a given level of precision is reached, and the surface is known to lie within the union of ambiguous cells. Those ambiguous cells may be further subdivided into globally parametrizable components [12], and the topology of the surface may be determined with a few point samples.

Such polygonization schemes yield a both geometrically precise and topologically correct representation of the surface, though at the expense of a deep recursion in the subdivision process.

2.2. Particle systems

An alternative technique to spatial partitioning consists in constraining a particle system to the implicit surface [17, 19]. The surface is sampled by scattering particles randomly throughout space that migrate to the surface following the field function's gradient. Those particles split and repel each other to achieve a uniform sampling distribution over the

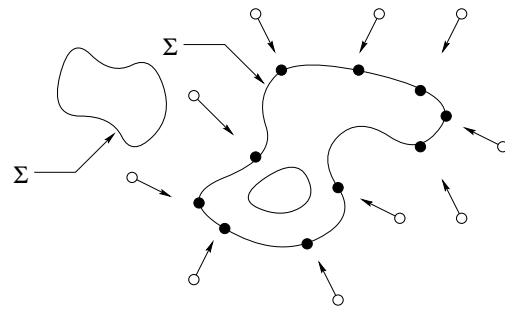


Figure 2. Scattering particles in space that migrate towards the surface may result in a wrong polygonization with missed hollow holes or disjoint surface components

surface. A Delaunay triangulation eventually creates the polygonization from the set of particles.

The polygonization can be updated when adding or removing elements [13]. Changes in the topology may be detected by comparing the implicit surface's gradient at the vertices of the polygonization with the normals of the approximating polygons.

However, disjoint pieces of the implicit surface as well as holes may be missed if the initial set of particles was wrongly scattered (figure 2). Moreover, particles that are often rendered as disks oriented tangent to the surface give only an approximate representation of the underlying surface (*e.g.* no hidden surface removal is provided). A Delaunay triangulation may create the corresponding polygonization, yet at the expense of extra computations that slow down the overall process.

2.3. Critical point tracking techniques

Morse theory shows how the topology of an implicit surface is affected by its function's critical points. Critical points are points of space where the gradient of the function drops to a null vector. Shrinkwrap polygonization techniques use the Lipschitz condition on the gradient to track the absence of critical points in a given neighborhood [10, 3].

Although the shrinkwrap algorithm updates the polygonization of vertices surrounding critical points, it may fail in finding hollow holes hidden within the implicit surface.

As shown in [16], extensive tracking of all critical points provides sufficient background for creating a polygonization whose topology is homeomorphic to the implicit surface's. Instead of shrinking the surface, an inflation process correctly detects and handles hollow holes in the surface.

However, this method requires the field function to be C^2

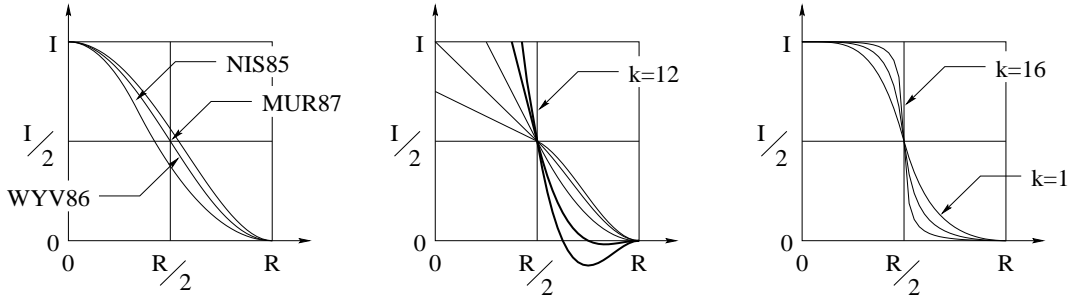


Figure 3. Blending functions parametrized by the radius of influence R_i , the maximum intensity I_i and the stiffness coefficient k_i ; Gascuel’s blending function reaches negatives values for high stiffness values

continuous and is limited to a restricted class of implicit surfaces built from elements based on point skeletons.

2.4. Hybrid techniques

An alternative technique consists in associating a fixed set of seed points to each element generating the implicit surface. Unlike other particle systems, seeds migrate towards the surface along a predefined vector [5]. When several elements intersect, some of those seeds become temporally invalid and deformations as well as topology changes may be handled.

However, the polygonization is often poor when several elements blend together.

3. Implicit surfaces

An implicit isopotential surface Σ is characterized as the points of space whose potential $f(x, y, z)$ equals a threshold value denoted T .

$$\Sigma = \{M(x, y, z) \in \mathbb{R}^3, f(x, y, z) = T\}$$

In this paper, we address the polygonization of implicit surfaces built from skeletons [20], also known as *soft objects*. A soft object is described by its scalar field $f(x, y, z)$, that is generated by summing the influences of scalar field elements $f_i(x, y, z)$ associated to their skeletons \mathcal{S}_i .

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

In general, the field contributions f_i are decreasing functions of the distance to a skeleton, thus we will assume that $f_i = g_i \circ d_i$ where $g_i(r)$ is the field function, and $d_i(x, y, z)$ refers to distance to the skeleton; thus an element will be

characterized by a triple $\{\mathcal{S}_i, d_i, g_i\}$. In our own implementation, the skeletons \mathcal{S}_i may be any geometric primitive admitting a well defined distance function, *i.e.* convex polyhedra of any dimension.

Through out this paper, the blending functions will have a finite support, *i.e.* $\forall r \in \mathbb{R}, r > R_i \Rightarrow f_i(r) = 0$ where R_i refers to the radius of influence of the element i . Therefore, the scalar field functions f_i will have no influence over f outside the region of influence, which will be denoted as Ω_i . As we will see later, the finite support property enables us to design an incremental polygonization scheme.

As suggested in [8] and [14], generalized anisotropic distance functions may also be used. Another wide class of parametrized translational and rotational primitives has been proposed in [2]. All those distance functions may be used in our implementation as long as the region of influence Ω_i remains bounded in space.

Several blending functions have been proposed in the literature [1]. Blending functions g_i are parametrized by the radius of influence R_i and the maximum intensity I_i (figure 3). The following functions are the fastest to evaluate because of their polynomial closed form in r^2 that saves a square root evaluation. [20] has proposed the following cubic function that provides C^1 continuity:

$$g(r) = -\frac{4}{9}r^6 + \frac{17}{9}r^4 - \frac{22}{9}r^2 + 1$$

Simpler quadric functions have been designed with a view to optimizing computations:

$$g(r) = (1 - r^2)^2$$

The curvature of the blend between two elements may be controlled with another class of blending functions that involve a stiffness coefficient k_i . The following one [1] proves to be both computationally efficient and provides the animator with extra control.

$$\begin{cases} g(r) = 1 - \frac{9r^4}{k + \left(\frac{9}{2} - 4k\right)r^2} & 0 \leq r^2 \leq \frac{1}{4} \\ g(r) = \frac{(1-r^2)^2}{\frac{3}{4} - k + \left(\frac{3}{2} + 4k\right)r^2} & \frac{1}{4} \leq r^2 \leq 1 \end{cases}$$

As we will see later, modifying the parameters characterizing an element i will change the potential field f_i and the changes will be propagated to f but limited to the region of influence Ω_i .

4. Incremental Algorithm

In this section, we address the recursive creation of the octree structure and focus on its incremental updating as elements are added or removed from the implicit surface characterization or as parameters previously described are modified.

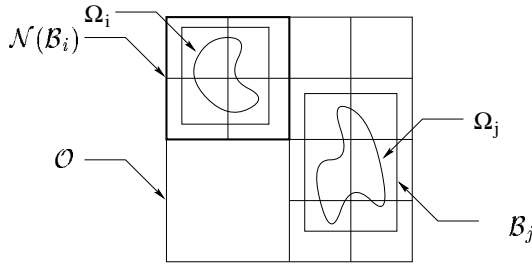


Figure 4. Octree structure \mathcal{O} , the node $\mathcal{N}(\mathcal{B}_i)$ is the smallest node in \mathcal{O} embedding the bounding box \mathcal{B}_i of the element i

In the remainder of this paper, an octree will be referred to as \mathcal{O} , \mathcal{N} will denote a node of \mathcal{O} and \mathcal{C} a cell of the octree (figure 4). Given the element i , the bounding box of its support Ω_i will be denoted as \mathcal{B}_i and $\mathcal{N}(\mathcal{B}_i)$ will refer to the smallest node in the octree embedding the bounding box \mathcal{B}_i .

4.1. Overview

Octree based polygonization techniques require the creation of a cell embedding the whole object. The recursive decomposition of this initial cell relies on subdivision criteria that identify regions of space that are to be further explored. Whenever the potential field is changed, the whole octree needs to be created from scratch.

In this paper, we address the polygonization of implicit surfaces built from elements i whose regions of influence Ω_i are bounded in space. Modifying the parameters characterizing an element i such as the field function g_i , the distance function d_i or the skeleton \mathcal{S}_i results in a modification of the potential field which is limited to the region of influence Ω_i . The potential field remains unchanged outside Ω_i . Adding or removing an element j also changes the potential field f locally, more precisely f is modified in the region of influence Ω_j .

Thus, we wish to avoid the polygonization wherever no changes occurred, *i.e.* outside Ω_i . We propose an incremental polygonization algorithm based on an octree decomposition of space. Implicit surfaces built from elements are created by adding or removing elements (moving elements can be thought of as a special sequence of remove and add).

Given an implicit surface built from n elements, we propose to inflate or deflate the octree whenever adding or removing an element. A modification of the definition of the potential field results in a local modification of the potential field and a local update of the implicit surface's polygonization.

The use of bounding volumes for each primitive enables us to speed-up the computation of the potential field, and preserve the octree structure and the polygonization wherever its cells do not intersect the region of influence of the modification.

Only the cells of the octree that intersect the region of modification need to be adaptively resampled and polygonized accordingly.

4.2. Creation of the octree structure

Spatial subdivision is a pre-processing step that prunes away empty regions of space. The creation of the octree structure has been addressed for ray-tracing acceleration purpose [9, 7].

4.2.1. Lipschitz properties

The Lipschitz properties provides us with criteria to detect the intersection between the surface Σ and a cell \mathcal{C} of the octree. Let us recall the definition of a Lipschitz function, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is said to be Lipschitz if :

$$\exists \lambda > 0, \forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n, |f(x) - f(y)| \leq \lambda \|x - y\|$$

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ a Lipschitz function and λ its Lipschitz constant. Let $\mathcal{B}(c_0, r_0)$ the ball of space centered at c_0 and of radius r_0 ; whatever the point $x \in \mathcal{B}(c_0, r_0)$, $|f(c_0) - f(x)| \leq \lambda r_0$. If $x \in \Sigma$, then $|f(c_0)| \leq \lambda r_0$, thus we derive the following corollary: if $|f(c_0)| \geq \lambda r_0$, then $\Sigma \cap \mathcal{B}(c_0, r_0)$ is empty.

As we will see later, implicit surfaces built from skeletal elements lend themselves to Lipschitz condition based algorithms for the Lipschitz constants λ and γ of f can be computed easily.

4.2.2. Octree refinement

Given a cell \mathcal{C} , centered at c_0 and of radius r_0 , the following criteria culls away cells in space that do not straddle the implicit surface [9].

- If $|f(c_0)| > \lambda r_0$, the cell \mathcal{C} does not straddle the implicit surface Σ (step ①).
- Otherwise, the cell \mathcal{C} is subdivided into eight sub-cells and the algorithm recursively applied.

Several improvements to this algorithm have been proposed in [7]. Step ① may be replaced by a more accurate root exclusion technique based on the Taylor expression of f :

- If $|f(c_0)| > \|\nabla f(c_0)\| r_0 + \gamma r_0^2$, then the cell \mathcal{C} cannot straddle the implicit surface Σ .

It is interesting to point out that this new criterion is inexpensive since we are to evaluate $\nabla f(c_0)$ to improve the accuracy of γ . In practice, it results in a slight acceleration in the root isolation algorithm.

Several cell condensation heuristics have also been proposed in [7]. A trivial one is that if all the children of a cell are detected empty, then they are deleted and the cell itself is marked empty.

However, the condensation of cells that hold similar Lipschitz constants may not be applied as it is the case in ray tracing. When polygonizing with octrees, space must be recursively subdivided as far as possible until cell exclusion criteria are met or until the maximum depth is reached.

4.2.3. Lipschitz constants computation

Eliminating straddling cells is the more efficient (which means that the surface converging is the faster) as the Lipschitz constants λ and γ are the smaller. Thus, local Lipschitz constants may be associated with each cell so as to speed-up spatial subdivision.

Most existing accelerating techniques try to find an accurate estimation of λ and γ to speed-up the computations. Thus, local Lipschitz constants are associated with each cell so as to speed-up intersection tests between the cells and the surface.

The computation of cell constants may be performed in two ways. The first method consists in computing more precise

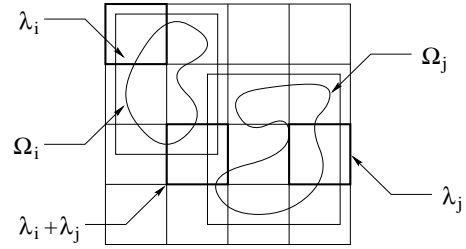


Figure 5. Fast local Lipschitz constants computation

local Lipschitz constant λ as follows : the norm of the gradient is computed at the center c_0 of the cell \mathcal{C} , and given the bound γ of the second derivative, therefore we have the following expression where r_0 stands for the half length of the diagonal of the cell :

$$\|\nabla f(x)\| < \|\nabla f(c_0)\| + \gamma r_0$$

Although this method requires more pre-processing computations because of the gradient's computation, it provides tighter derivative bounds. Moreover, it relies on a global γ constant which is far from optimal whenever but a few elements blend within the cell \mathcal{C} .

We propose the following technique that consists in intersecting the region of influence Ω_i of each element with the cell \mathcal{C} . If the intersection $\Omega_i \cap \mathcal{C}$ is not empty, then λ_i are added to the local value (figure 5). Let δ_i be set to 1 if $\Omega_i \cap \mathcal{C}$ is not empty and set to 0 otherwise, λ may be written as :

$$\lambda = \sum_{i=1}^{i=n} \delta_i \lambda_i$$

The same technique is applied to define γ_i in a given cell. An even more accurate Lipschitz constant λ may be derived by combining the previous expressions :

$$\lambda = \|\nabla f(c_0)\| + \left(\sum_{i=1}^{i=n} \delta_i \gamma_i \right) r_0$$

Adding and removing elements are the key steps in our incremental technique. The modification of the octree structure and its polygonization occurs only in those cells that intersect the region of space where the potential field has been modified. It is worth pointing out that adding, removing or modifying an element i changes both the potential field f_i and its corresponding Lipschitz constants λ_i and γ_i . Therefore, λ and γ are computed on the fly for each cell \mathcal{C} intersecting Ω_i .

4.3. Bounding boxes

In practice, checking whether the cells \mathcal{C} of the octree and the regions of influence Ω_i intersect proves to be computationally expensive.

Whatever the dimension of the skeleton, the region of influence may be defined as a rounded polytope that is the Minkowski sum $\Omega_i = \mathcal{S}_i \oplus \mathcal{B}(0, R_i)$ where $\mathcal{B}(0, R_i)$ refers to the ball centered at the origin and of radius R_i [6]. Thus, checking if $\Omega_i \cap \mathcal{C}$ is non-empty is the more complex as the dimension of the skeletons \mathcal{S}_i increases.

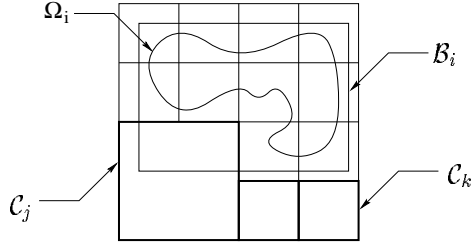


Figure 6. The bounding box \mathcal{B}_i intersects all high level subcells (including the bottom-left subcell \mathcal{C}_j even though $\mathcal{C}_j \cap \Omega_i$ is not empty), but rapidly culls away smaller subcells (e.g. \mathcal{C}_k)

It may become even impossible with elements built with anisotropic distances for the very reason that the region of influence Ω_i may not be explicitly defined (figure 6).

We propose to create a bounding box \mathcal{B}_i embedding the region of influence Ω_i for each soft object's element i . This bounding box is in general straight forward, and speeds-up the intersection tests.

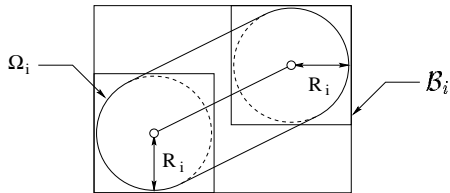


Figure 7. Bounding box of an element built from a segment skeleton

As shown in [6], the characterization of the bounding box \mathcal{B}_i is straight-forward. Whatever the dimension of the skeleton, the region of influence may be defined as a polytope that is the Minkowski sum $\mathcal{S}_i \oplus \mathcal{B}_\infty(0, R_i)$ where $\mathcal{B}_\infty(0, R_i)$

refers to the cube centered at the origin with an half-side length R_i . In our own implementation, the skeletons \mathcal{S}_i of the contributing elements are convex polyhedra of any dimension. Thus, the bounding box \mathcal{B}_i may be easily computed from the vertices of the skeleton and the radius of influence R_i (figure 7).

As we will see later, although bounding boxes may not tightly approximate the regions of influence Ω_i , they speed-up the intersection test between the elements and the cells \mathcal{C} of the octree.

4.4. Inflating and deflating the octree

Whenever adding a new element or increasing the size of an existing primitive, we may need to inflate the octree structure so that the octree \mathcal{O} should still satisfy $\Omega \subset \mathcal{O}$. Alternatively, the octree may be deflated whenever removing an element if only one of the sub-cells of the root cell is non-empty. We show how to update the octree structure accordingly in the following paragraphs.

4.4.1. Inflating the octree

Let \mathcal{O}_{n-1} be the octree built from $n - 1$ elements, it is defined by a cell \mathcal{C} centered at c_0 . How do we update the octree structure when a new element n is added ?

If the element n is inside \mathcal{C} (i.e. $\Omega_n \subset \mathcal{O}_{n-1}$), then \mathcal{O}_n need not be inflated and we set $\mathcal{O}_n = \mathcal{O}_{n-1}$. Otherwise, there are eight ways to create a new node \mathcal{N} in such a way that \mathcal{O}_{n-1} will be a child of \mathcal{N} (figure 9).

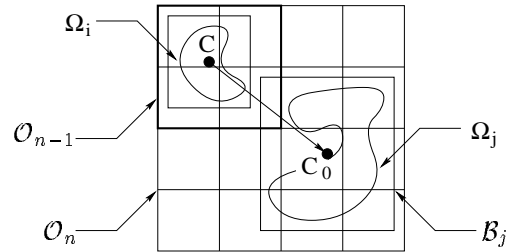


Figure 9. Inflating the octree

Let c the center of \mathcal{N} , the vector \vec{cc}_0 characterizes in which direction the octree \mathcal{O}_{n-1} should be inflated. Several iterations of the inflation sequence may be needed, thus this process is repeated until $\Omega \subset \mathcal{O}_n$.

As mentioned earlier, checking $\Omega_n \subset \mathcal{O}_{n-1}$ may prove to be computationally expensive or impossible, thus we rely on the very efficient bounding box test $\mathcal{B}_n \subset \mathcal{O}_{n-1}$.

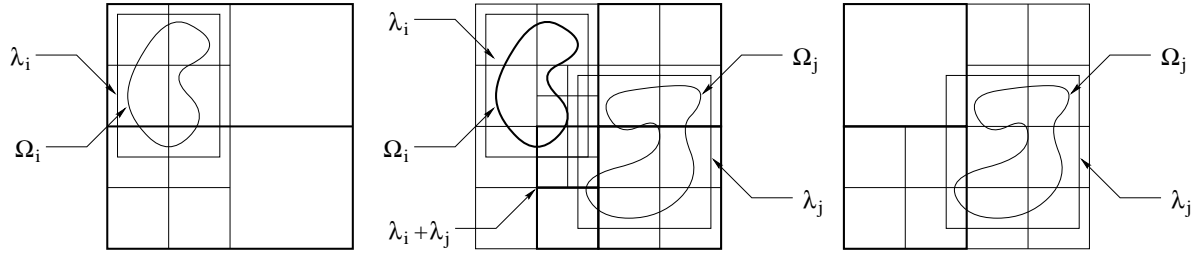


Figure 8. Incremental update of the octree structure, dark octree cells have been modified (they have been either further refined or their children were deleted) whereas others need to be updated

4.4.2. Deflating the octree

The octree may be deflated if the cells of seven children out of eight of the root node of \mathcal{O}_n do not intersect the bounding box of the remaining $n - 1$ elements. This may occur when an element is removed or when the region of influence of an element shrinks (for example by reducing the radius of influence).

The root node is deleted and the new root node of \mathcal{O}_{n-1} is set to the non-empty child of \mathcal{O}_n . This process is repeated until more than one of eight children of the root node is non-empty.

4.5. Modifying the octree's structure

In this paragraph, we assume that the element k , *i.e.* its field function f_k , has been modified.

Let $\mathcal{N}(\mathcal{B}_k)$ be the smallest node of \mathcal{O}_k embedding \mathcal{B}_k . The structure of $\mathcal{N}(\mathcal{B}_k)$ needs to be updated according to the new potential function. We propose the following incremental algorithm that recursively explores the existing octree and updates the potential field values at the corners of the cells whenever necessary (*i.e.* whenever the support of the modifying element Ω_k intersects the cell).

Since checking the intersection $\Omega_k \cap \mathcal{C}$ is computationally expensive, we rely on the bounding box approximation \mathcal{B}_k of Ω_k to speed-up the intersection tests.

Thus, if the cell \mathcal{C} of the node \mathcal{N} intersects \mathcal{B}_k , then potential field at the corners of the cell is updated and the Lipschitz condition is applied. Three cases may arise during this recursive octree traversal (figure 8) :

- the node \mathcal{N} was already refined, however the Lipschitz condition tells that it is now empty, then the children of \mathcal{N} are recursively deleted and the node is marked as empty ;
- the node \mathcal{N} was already refined and the Lipschitz con-

dition tells that it still may not be empty, then the children of \mathcal{N} are recursively explored ;

- the node \mathcal{N} was a terminal node (*i.e.* a leaf in the octree) and the Lipschitz condition asks for further refinement, then the standard subdivision algorithm may be applied to that node.

```

void Explore (Node  $\mathcal{N}$ , Element  $j$ )
if ( $\mathcal{N} \cap \mathcal{B}_j == 0$ ) return ; ①
Update Intensity At Vertices ( $\mathcal{N}$ ) ;
if (Check Subdivide ( $\mathcal{N}$ ) == FALSE)
| delete Children ( $\mathcal{N}$ ) ;
| return ;
if (All Children ( $\mathcal{N}$ ) == NULL)
| Subdivide ( $\mathcal{N}$ ) ;
| return ;
for ( $i = 0$ ;  $i < 8$ ;  $i++$ )
| if ( $\mathcal{N}_i \neq \mathbf{NULL}$ )
| | Explore ( $\mathcal{N}_i, j$ ) ;
| else
| | Create ( $\mathcal{N}_i$ ) ;
| | Subdivide ( $\mathcal{N}_i$ ) ;
if (All Children ( $\mathcal{N}$ ) == NULL)
| delete  $\mathcal{N}$  ;
|  $\mathcal{N} = \mathbf{NULL}$  ;
end.

```

It is worth noticing that updating the potential field at the corners of an interacting cell may be performed by only summing or subtracting the field's contribution f_n to the previous potential field.

The step ① of the algorithm takes advantage of the bounding box \mathcal{B}_n to rapidly step over cells that may not be modified by the new element. This enhancement speeds-up the overall performance significantly. Since we rely on the modified Lipschitz constants to analyse the cells \mathcal{C} of the octree,

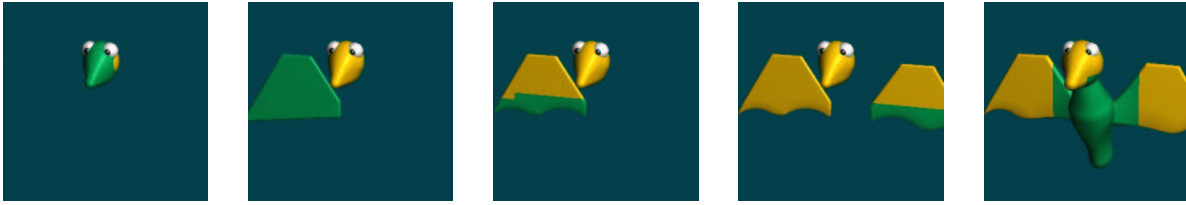


Figure 10. Modeling an implicit bird, the lower parts of the wings have been designed with negative potential fields

we evaluate λ and γ on the fly for each cell when the octree is traversed.

5. Results

The octree provides us with a set of cells that straddle the implicit surface Σ . As mentioned in the introduction, we will not focus on the creation of the polygons. For each cell \mathcal{C} straddling the implicit surface Σ , the potential field is computed at its vertices which yields the cell *configuration type*. The polygons approximating Σ within \mathcal{C} are created according to this configuration type [20, 11, 4].

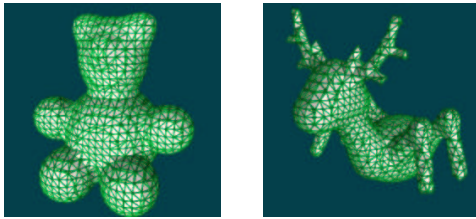


Figure 11. Polygonized bear and elk characters (7566 and 8926 triangles respectively)

Ambiguous configurations types may require additional work to maintain surface consistency and avoid holes or dangling faces. Several disambiguation strategies have been addressed [12].

As pointed out in [4], octree partitioning may introduce artifacts in the polygonization such as *cracks* that are created between adjacent cells of different size. As mentioned in section 4.2.2, we do not suffer from such cracks as space is recursively subdivided until exclusion criteria are met or until the maximum depth is reached, which yields terminal cells of equal size.

The algorithms have been implemented in our object-oriented ray tracer and tested on a Pentium-II processor workstation with a 266MHz Clock and 64 megabytes of main memory. The algorithms incorporated the same en-

hancements wherever applicable. Reported cpu time is in seconds.

To compare the different methods, we have incrementally built different *soft toys like* shapes and checked the elapsed time for both the incremental update and the force brute re-polygonization. The bear, the bird and the elk characters are built from 9, 11 and 32 elements respectively. The bear is built from point skeletal elements only, whereas the bird and the elk involve higher dimension skeletons (segments and polygons).

Depth	Triangles	Polygonization	
		Incremental	Global
5	3680	0.11	0.11
6	4254	0.12	0.19
6	5152	0.24	0.40
7	5250	0.08	0.51
7	5348	0.09	0.62
7	5370	0.06	0.76
7	5850	0.19	1.01
7	6330	0.20	1.22
7	6948	0.29	1.52
7	7566	0.36	1.88

Table 1. Timings (in seconds) for global and incremental octree polygonization of the bear, the overall number of generated triangles is also reported

The two color plates (figure 10, 12) represent the incremental creation of a bird and an elk-like character. The skeletons of the elements are of any dimension, and only a few primitives blend at the same time. At each step, pieces of the surface that were re-polygonized because of the propagation of the influence of the new elements have been colored in dark green, whereas unchanged patches have been set to bright yellow.

As expected, modifying the octree structure and the corresponding polygonization only where needed speeds-up the

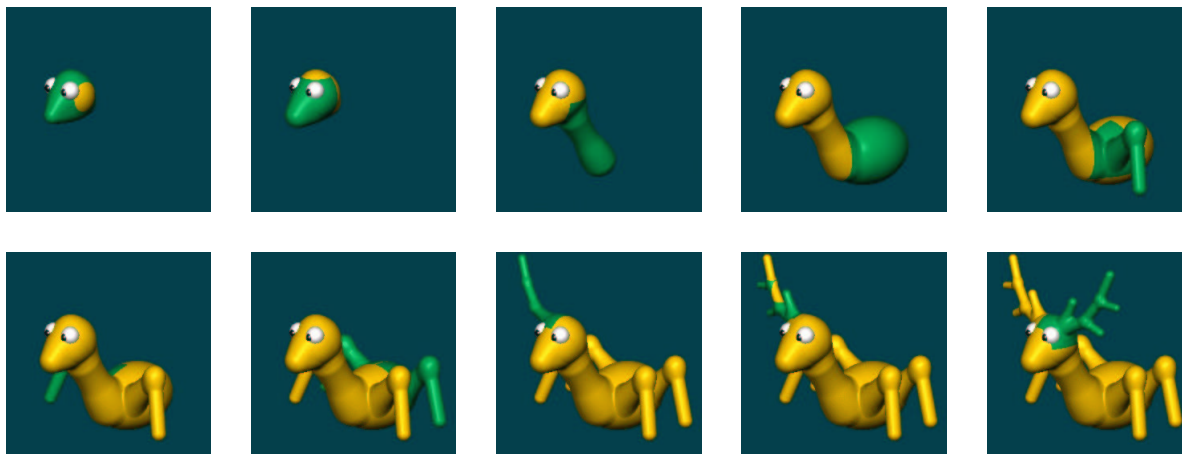


Figure 12. Modeling an implicit elk, the creation of the horns have almost no influence over the overall shape which results in a very fast incremental update of the polygonal approximation

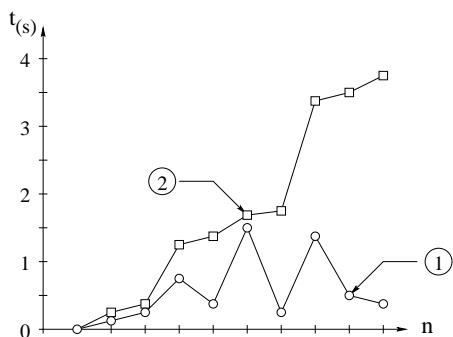


Figure 13. Timings corresponding to the incremental creation of a bird (11 primitives, 6720 triangles) ; our incremental method (circles) proves to be much faster than a brute force polygonization performed at each step (squares)

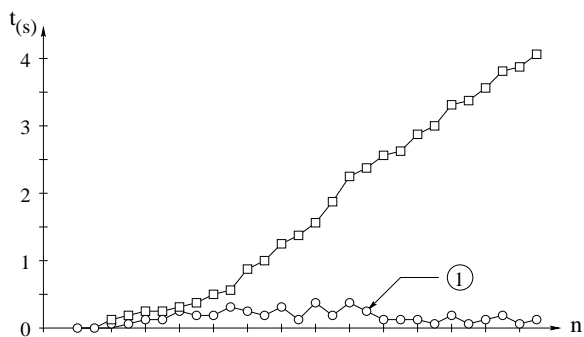


Figure 14. Incremental creation of a elk (32 primitives, 2202 triangles) ; modeling with small primitives results in very fast updates

algorithm significantly. As shown in table 1, incremental updates are performed interactively.

Timings show that the accelerations are the greater as the object becomes more and more complex and involves more and more primitives. As show in figure 13, updating the octree structure and incrementally modifying the polygonization is the faster as the region of influence Ω_i of the primitive i is smaller. Small elements have very little influence over the overall polygonization which results in fastest updatings (figure 13, case ①), whereas elements with a large region of influence (figure 13, case ②) may affect almost all the polygonization.

Figure 14 demonstrates that the incremental polygonization method outperforms the brute-force repolygonization by an order of magnitude whenever the primitives are small compared to the overall implicit model.

In both cases, the maximum depth of the octree has been set to 9, which provided a very accurate approximation of the surface. In practice, this maximum depth may be reduced so as to get even faster polygonization.

6. Conclusion

We have proposed an incremental polygonization method for implicit surfaces built from elements whose regions of influence are bounded in space. Such an algorithm lends it-

self for interactive modeling as it benefits from the shape coherence and updates both the octree structure and the corresponding meshing only where needed.

Whenever creating or updating the octree, we recursively subdivide space to locate straddling cells until the maximum depth is reached. Although the surface approximation becomes the more precise as the octree depth increases, numerous triangles are generated, even in relatively smooth parts of the surface. A more sophisticated algorithm would adaptively sample space, and deal with cracks as mentioned in [4].

We are currently working on a more complex octree data structure that would allow us to define the approximation of the surface as a topologically consistent mesh based on a adjacency graph.

References

- [1] C. Blanc and C. Schlick. Extended Field Functions for Soft Objects. *Proceedings of Implicit Surfaces'95*, pages 21–32, Grenoble, April 1995.
- [2] C. Blanc and C. Schlick. Implicit Sweep Objects. *Proceedings of Eurographics'96*, Vol. 15(3) : pages 165–174, August 1996.
- [3] A. Bottino, W. Nuij and K. VanOverveld. How to Shrinkwrap through a Critical Point : an algorithm for the daptive triangulation of iso-surfaces with arbitrary topology. *Proceedings of Implicit Surfaces'96*, pages 53–72, October 1996.
- [4] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, Vol. 5(4) : pages 341–356, 1988.
- [5] M. Desbrun, N. Tsingos and M.P. Gascuel. Adaptive Sampling of Implicit Surfaces for Interactive Modeling and Animation. *Computer Graphics Forum*, Vol. 15(5) : pages 319–325, December 1996.
- [6] E. Galin. Métamorphose et Visualisation de Blobs à Squelettes. *Ph.D. Thesis*, July 1997.
- [7] J.D. Gascuel. Implicit Patches : An Optimized and Powerfull Ray Intersection Algorithm for Implicit Surfaces. *Proceedings of Implicit Surfaces'95*, pages 143–160, Grenoble, France, April 1995.
- [8] Z. Kacic-Alesic and B. Wyvill. Controlled Blending of Procedural Implicit Surfaces. *Proceedings of Graphics Interface'91*, pages 236–245, 1991.
- [9] D. Kalra and A.H. Barr. Guaranteed Ray Intersections with Implicit Surfaces. *Computer Graphics*, Vol. 23(3) : pages 297–306, July 1989.
- [10] K. van Overveld and B. Wyvill. Potentials, Polygons and Penguins. An efficient adaptive algorithm for triangulating an equi-potential surface. *Proceedings of 5th Annual Western Computer Graphics Symposium*, Vol. 23(3) : pages 31–62, 1993.
- [11] W.E. Lorensen and H.E. Cline. Marching Cubes : a High Resolution 3–D Surface Construction Algorithm. *Computer Graphics*, Vol. 21(4) : pages 163–170, 1987.
- [12] P. Ning and J. Bloomenthal. An Evaluation of Implicit Surface Tilers. *IEEE Computer Graphics and Applications*, Vol. 13(6) : pages 33–41, 1993.
- [13] H.C. Rodrian and H. Moock. Dynamic Triangulation of Animated Skeleton-based Implicit Surfaces. *Proceedings of Implicit Surfaces'96*, pages 37–52, October 1996.
- [14] S. Sclaroff and A. Pentland. Generalized Implicit Functions for Computer Graphics. *Computer Graphics (Siggraph'91 Proceedings)*, Vol. 25 : pages 247–250, July 1991.
- [15] J.M. Snyder. Interval Analysis for Computer Graphics. *Computer Graphics (Siggraph'92 Proceedings)*, Vol. 26 : pages 121–130, July 1992.
- [16] B.T. Stander and J.C. Hart. Guaranteeing the Topology of Implicit Surface Polygonization for Interactive Modeling. *Computer Graphics (Siggraph'97 Proceedings)*, Vol. 32 : pages 279–286, August 1997.
- [17] R. Szeliski and D. Tonnensen. Surface Modeling with Oriented Particle Systems. *Computer Graphics (Siggraph'92 Proceedings)*, Vol. 26 : pages 185–194, July 1992.
- [18] J. Wilhelms and A. VanGelder. Topological considerations in isosurface generation (extended abstract). *Computer Graphics*, Vol. 24(5) : pages 79–86, November 1990.
- [19] A.P. Witkin and P.S. Heckbert. Using Particles to Sample and Control Implicit Surfaces. *Computer Graphics (Siggraph'94 Proceedings)*, Vol. 28 : pages 269–278, July 1994.
- [20] B. Wyvill, C. McPheeters and G. Wyvill. Data Structure for Soft Objects. *The Visual Computer*, Vol. 2(4) : pages 227–234, 1986.
- [21] G. Wyvill and A. Trotman. Ray-Tracing Soft Objects. *Computer Graphics International'90*, pages 467–476, 1990.