A Fast Particle System Framework for Interactive Implicit Modeling

Eric Galin
LIRIS - CNRS
Université Claude Bernard Lyon 1
eric.galin@liris.cnrs.fr

Rémi Allègre
LIRIS - CNRS
Université Claude Bernard Lyon 1
remi.allegre@liris.cnrs.fr

Samir Akkouche
LIRIS - CNRS
Université Claude Bernard Lyon 1
samir.akkouche@liris.cnrs.fr

Abstract

In this paper, we present a controllable, fast and efficient particle system framework for skeletal implicit surfaces defined by the BlobTree model. We propose efficient algorithms for generating sets of particles for every type of skeletal element and for combining those sets for warping, blending and Boolean nodes. Given a target level of detail, our system generates particles which are distributed uniformly over the implicit surface. Our incremental approach lends itself to interactive editing of complex models.

Keywords: implicit surface modeling, interactive visualization, particle systems.

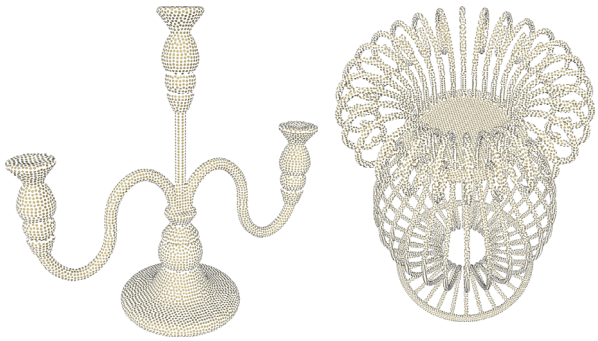

Figure 1. Some complex BlobTree models visualized by uniformly distributed particles

1. Introduction

Implicit surfaces have been successfully used for modeling complex surfaces, reconstructing shapes from point sets, animating viscous fluids and morphing objects of different topology. Still, implicit surface modeling has been hampered by the lack of fast interactive visualization methods. Interactive modeling is an iterative, essentially trial and error process which makes interactive visual feedback necessary. Therefore, there is a still a need for fast implicit surface visualization techniques.

An implicit surface is mathematically defined as the set of points $\mathbf{p}$ in space such that $f(\mathbf{p}) = 0$ where f denotes the function that characterizes the surface:

$$S = \{\mathbf{p} \in \mathbf{R}, f(\mathbf{p}) = 0\}$$

Implicit surfaces can be displayed either directly by ray-tracing techniques, or indirectly by polygonization and particle systems. Although several techniques have been proposed for accelerating the computation of the intersection between a ray and an implicit surface [19, 10, 11, 18, 25, 14], ray tracing remains too slow for interactive applications. Several efficient incremental polygonization techniques have been proposed for interactive editing and visualization of implicit surfaces [9, 2, 12]. Still, polygonization techniques need to generate and often keep track of the connectivity information between the vertices, which is computationally demanding.

With recent research on point based representation of shapes [3], several particles systems have been developed for generating points on a implicit surface [23, 13, 24]. Those approaches have been developed for general implicit surfaces however, and do not take advantage of the specific properties of a given model to speed up computations. Moreover, they are computationally demanding and remain impractical for generating a dense sampling of the surface.

In this paper, we present a system for creating tens of thousands of particles over an implicit surface defined by a BlobTree model at interactive rates. Our algorithm recursively traverses the tree structure of the BlobTree: the leaves procedurally generate sets of particles that are efficiently combined at the nodes of the tree. Because we need not keep track of the connectivity information between vertices as for polygonization techniques, our system support extreme geometric deformations and Boolean and blending operations during interactive editing.

Our approach is closely related in spirit to shape modeling with point sampled geometry [16] and to the interactive Boolean operation techniques proposed for point set sur-

faces [1]. The very difference between our system and those approaches is that we need not approximate the field function locally from the point samples as we can use the BlobTree model directly to evaluate $f(\mathbf{p})$ everywhere in space instead of using a local moving least square reconstruction [15]. The particles are distributed almost uniformly over the surface, and this distribution may be optimized by performing but a few iterations of a Witkin Heckbert particle system algorithm [23].

The remainder of this paper is organized as follows. Section 2 presents an overview of some related work. Section 3 recalls the fundamentals of the BlobTree model and defines notations. In Section 4, we detail our algorithms for generating the particles on the implicit surface defined by a BlobTree model. Eventually, Section 5 presents some examples and the corresponding timings.

2. Related work

In 1994, Witkin and Heckbert [23] introduced a novel, physically based technique in which the constrained a set of interacting particles to sample an implicit surface. In this system, each particle repels neighboring particles to minimize a Gaussian energy function. At equilibrium the system produces an homogeneous sampling of the implicit surface.

Several approaches [17, 6] have been proposed to enhance the original Witkin-Heckbert technique by adapting the particle distribution to the local curvature of the surface. The main difficulty with those techniques is that they introduce several critical parameters which are difficult to specify and control the particle system. Those parameters include the characteristic length in the inter-particle distance σ which is the standard deviation of the Gaussian, and free parameters involved in the discretized gradient descent algorithms that are used for projecting particles on the implicit surface. Recently, [13] presented a new class of energy functions and a inverse Hessian minimization scheme that provide more stable, scalable and controllable framework. The proposed method generates particles which are adaptively distributed over the implicit surface, at the expense of a more computationally demanding algorithm.

An alternative approach to particle systems consists in using a stochastic method for generating sample points on an implicit surface as proposed in [21]. The method relies on a Monte-Carlo simulation confined on an implicit surface and generates a uniform distribution of particles, which may be mathematically proved. The stochastic feature of the algorithm guarantees the realization of equilibrium state when the number of samples is large.

3. The BlobTree

The BlobTree model [26] is characterized by a hierarchical combination of primitives organized in a tree datastructure (Figure 2). The nodes of the tree include global and local blending, Boolean operators and warping nodes, whereas the leaves are skeletal primitives. The implicit surface is defined as the set of points whose potential field equals a threshold value, denoted as T:

$$S = \{\mathbf{p} \in \mathbf{R}, f(\mathbf{p}) = T\}$$

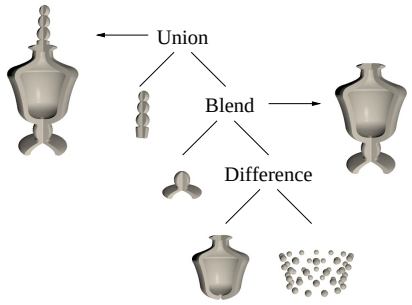

Figure 2. A simplified representation of the tree structure of a bottle

Skeletal primitives are defined by a skeleton, a distance function d and a potential function g with a compact support. The field function for every skeletal primitive is defined as $f(\mathbf{p}) = g \circ d(\mathbf{p})$. In our system, we use the Euclidean distance to the skeleton and the following potential function parameterized by a radius of influence denoted as r_0 and the intensity (or strength) denoted as s:

$$g(r) = s\left(1 - \left(\frac{r}{r_0}\right)^2\right)^2$$

Since the potential function g has a compact support, every skeletal primitive has a bounded region of influence in space, which will be denoted as Ω. The region of influence of a primitive is defined by sweeping a sphere of radius r_0 over the skeleton.

Every node incorporates a bounding box, denoted as $\mathcal{B}$, so as to rapidly discard useless field function evaluations when queries are performed in empty regions of space. The computation of the field function $f(\mathbf{p})$ at a given point in space is performed by recursively traversing the BlobTree structure. The skeletal primitives at the leaves of the tree return potential field values, which are combined by operators at the internal tree nodes.

4. Particle generation algorithm

The creation of the set of particles covering the implicit surface S is performed in an incremental way by recursively traversing the tree data-structure. Given a target average distance between particles, the primitives at the leaves of the tree generate sets of particles which are combined together at the internal nodes of the tree.

Notations Throughout this section, $\mathcal{P}$ will denote a set of particles, and $\mathcal{P}_i$ will refer to the i^{th} particle of this set. A particle will be characterized by its location $\mathbf{p}$ and its normal $\mathbf{n}$. In our algorithms, δ will refer to the target average distance between particles.

4.1. Generating particles for skeletal primitives

In our system, every implicit primitive automatically generates a set of particles for a target average distance between particles. We have developed a specific and optimized procedure for every kind of skeletal primitive. The particle generation procedure generates a set of particles on the surface with an almost uniform sampling distribution.

The implicit surface generated by a single skeletal primitive can be defined by sweeping a sphere with a radius $g^{-1}(T)$ along the skeleton. The inverse potential field function g^{-1} is defined as:

$$g^{-1}(T) = r_0 \left(1 - \left(\frac{T}{s}\right)^{1/2}\right)^{1/2}$$

Thus, the implicit surface generated by a line segment is a rounded cylinder, and the surface generated by a circle is a torus. For most primitives, this surface can be defined as a patchwork of simple surface patches such as portions of spheres, cylinders, planes, discs or tori. We have identified the different surface patches for every skeletal primitive in our system, and implemented efficient sampling schemes for those surface patches.

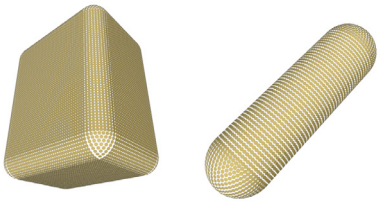

Figure 3. Procedurally generated particles for some skeletal primitives

For every type of surface patch we invoke a specific optimized procedural particle generation scheme that relies on a hexagonal sampling pattern and aims at creating an almost uniform particle distribution over the surface. Table 1 presents a short overview of some surface patches for a reduced set of primitives.

Skeleton	Torus	Cylinder	Square	Sphere	Disc
Point				×	
Segment		×		×	
Circle	×				
Box		×	×	×	
Cylinder	×	×			×

Table 1. Some primitives and their corresponding surface patches

During the sampling process, we take advantage of the symmetries whenever possible to speed up the particle generation process. The set of particles is then replicated with respect to the symmetry centers, axes or planes. Thus, we need to sample an octant of a sphere, a quadrant of a cylinder and a rectangle for a box primitive (Figure 3).

4.2. Generating particles in complex regions

Complex regions are defined as regions in space where particles on the implicit surface can't be directly obtained from the sets of particles generated by skeletal primitives. Thus, complex regions are essentially produced by local and global blending operators, and can be defined as the intersection of the compact supports of the argument models.

Particles are generated in complex regions by decomposing space into small cubic cells, and by creating a particle for every cell straddling the implicit surface. We proceed as follows (Figure 4). Let $\mathcal{C}$ denote a cubic cell in space, and $\mathbf{c}$ the center of the cell. We first evaluate the gradient at the center of the cell $\nabla f(\mathbf{c})$ and define the line Δ as the line passing at point $\mathbf{c}$ and with direction $\nabla f(\mathbf{c})$.

Let $\mathbf{a}$ and $\mathbf{b}$ denotes the intersection points between the line Δ and the cell $\mathcal{C}$. We compute the intersection between the line segment $(\mathbf{a}, \mathbf{b})$ and the implicit surface S by applying simple interval analysis [19] or Lipschitz [11] ray implicit surface intersection techniques. If an intersection occurs, we create a new particle located at the intersection point $\mathbf{p}$ with a normal $\mathbf{n} = \nabla f(p)/\|\nabla f(p)\|$. Otherwise, no particle is created for the cell.

The particles generated by our voxel decomposition of space may not be uniformly distributed over the implicit

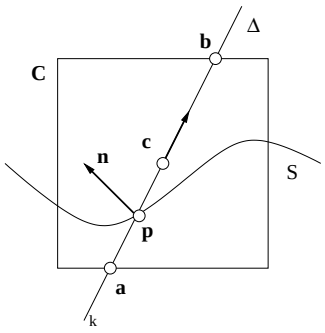

Figure 4. Particle generation scheme in a cubic cell

surface. Thus, in our system, it is possible to apply a local Witkin-Heckbert algorithm [23] to stabilize the particles to a more uniform distribution.

Implementation details Computing the intersection between a line segment $(\mathbf{a}, \mathbf{b})$ and the implicit surface S requires many field function evaluations. Recall that evaluating the potential field $f(\mathbf{p})$ require traversing the tree structure of the BlobTree.

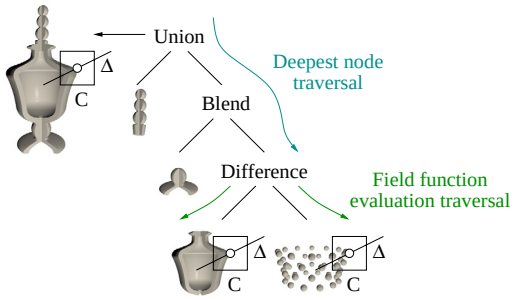

Figure 5. Optimized traversal of the BlobTree structure for accelerating the field function queries $f(\mathbf{p})$ performed in a given cell $\mathcal{C}$

To speed up the tree traversal process, we first find the deepest node in the tree $\mathcal{N}$ such that $(\mathbf{a}, \mathbf{b}) \in \mathcal{B}_{\mathcal{N}}$ where $\mathcal{B}_{\mathcal{N}}$ denotes the bounding box of node $\mathcal{N}$. Instead of evaluating $f(\mathbf{p})$ from the root node of the BlobTree model, all field function evaluations performed for points on the line segment $(\mathbf{a}, \mathbf{b})$ are performed from $\mathcal{N}$, so as to avoid traversing unnecessary nodes in the tree (Figure 5). This optimization is the more effective as the BlobTree model is complex, and as the cells $\mathcal{C}$ are small compared to the model.

4.3. Blending operators

Global blending Let $\mathcal{N}_C$ denote a blending node in the BlobTree and $\mathcal{N}_A$ and $\mathcal{N}_B$ its children nodes. In the general case, we first generate the set of particles $\mathcal{P}_A$ and $\mathcal{P}_B$ for the nodes $\mathcal{N}_A$ and $\mathcal{N}_B$. Then we remove the particles of $\mathcal{P}_A$ and $\mathcal{P}_B$ that lie in the bounded blending region $\mathcal{B}_R = \mathcal{B}_A \cap \mathcal{B}_B$. We generate the new particles in the region $\mathcal{B}_R$ by using a voxel particle generation algorithm (Figure 6). Finally, we converge to a uniform particle distribution by applying a local Witkin Heckbert algorithm [23] to the particles in the bounded region $\mathcal{B}_R$.

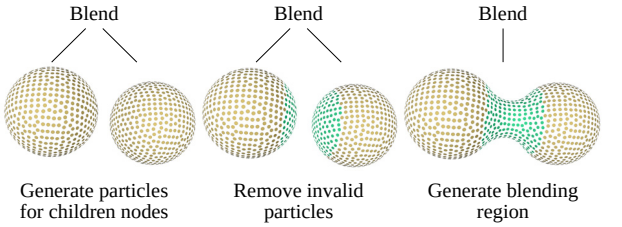

Figure 6. Outline of the particle generation algorithm for a blending node

This general algorithm is inefficient however if the two arguments models A and B overlap very much. In this case, the general algorithm wastes its time generating and deleting the sets of particles $\mathcal{P}_A$ and $\mathcal{P}_B$ before applying the voxel particle generation algorithm.

Therefore, we first evaluate how much the bounding boxes $\mathcal{B}_A$ and $\mathcal{B}_B$ of the nodes $\mathcal{N}_A$ and $\mathcal{N}_B$ overlap. Let V_A, V_B and $V_{A\cap B}$ denote the volume of $\mathcal{B}_A$, $\mathcal{B}_B$ and $\mathcal{B}_A \cap \mathcal{B}_B$ respectively. We define the overlapping ratio, denoted as ρ, between the two bounding boxes as:

$$\rho = \frac{V_{A\cap B}}{\min(V_A, V_B)} \qquad 0 \leq \rho \leq 1$$

If ρ is larger than a fixed threshold ρ_0, then the two boxes are overlapping very much, and we directly generate the set of particles $\mathcal{P}_C$ of $\mathcal{N}_C$ by using the voxel particle generation algorithm in $\mathcal{B}_C$. The user can provide a value for ρ_0 or directly specify which method should be used for every blending operator in the BlobTree. In our system, we use the fixed threshold $\rho_0 = 0.5$ as suggested in [4].

Local blending The particle generation algorithm for a local blending node resembles the previous algorithm. We first generate the set of particles $\mathcal{P}_A$ and $\mathcal{P}_B$ for the nodes $\mathcal{N}_A$ and $\mathcal{N}_B$. Then we remove the particles of $\mathcal{P}_A$ and $\mathcal{P}_B$ that lie in the bounded blending region $\mathcal{B}_R$ and eventually generate new particles by using a voxel particle generation algorithm.

4.4. Boolean operators

Contrary to blending operators, the set of particles $\mathcal{P}_C$ of a Boolean node $\mathcal{N}_C$ can be directly derived from the set of particles of $\mathcal{P}_A$ and $\mathcal{P}_B$ of its children and we need not use any voxel particle generation scheme (Figure 7).

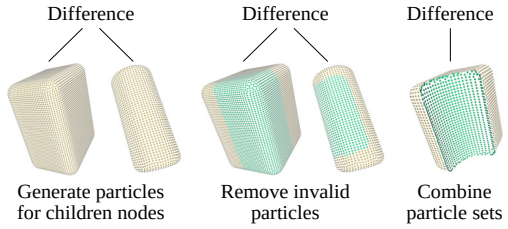

Figure 7. Outline of the particle generation algorithm for boolean nodes

The hierarchical particle generation scheme proceeds as follows: we first generate the set of particles $\mathcal{P}_A$ and $\mathcal{P}_B$ for the nodes $\mathcal{N}_A$ and $\mathcal{N}_B$, and then we remove the particles $\mathcal{P}_{A_i}$ and $\mathcal{P}_{B_j}$ of the sets $\mathcal{P}_A$ and $\mathcal{P}_B$ that are invalid as follows:

- For union $A \cup B$, remove the particles that satisfy $f_B(\mathcal{P}_{A_i}) > T$ and $f_A(\mathcal{P}_{B_j}) > T$.
- For intersection $A \cap B$, remove the particles that satisfy $f_B(\mathcal{P}_{A_i}) < T$ and $f_A(\mathcal{P}_{B_j}) < T$.
- For difference $A - B$, remove the subset of particles that satisfy $f_B(\mathcal{P}_{A_i}) > T$ and $f_A(\mathcal{P}_{B_j}) < T$.

One limitation of this efficient and straight forward approach is that the particles may not be distributed nicely along the sharp edges and corners produced by Boolean operations.

4.5. Warping operators

Let $\mathcal{N}_A$ denote the child node of a warping node. The general algorithm proceeds in three steps. First, we first create the set of particles $\mathcal{P}_A$ of the child node $\mathcal{N}_A$. Then the deformation is applied to the particles of $\mathcal{P}_A$ by transforming their vertices and normals. Eventually, we apply a local Witkin-Heckbert [23] algorithm to converge to a more uniformly sampled surface (Figure 8).

Translation and rotation preserve the uniform distribution of the particles, and need not invoke the Witkin-Heckbert algorithm. Uniform scaling uniformly stretches the set of particles: in this case the process is optimized as follows. Let α denote the scaling factor, the set of particles $\mathcal{P}_A$ of the child node $\mathcal{N}_A$ is created with a modified target average distance between particles defined as δ/α.

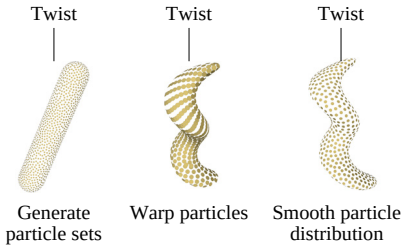

Figure 8. Outline of the particle generation algorithm for warping nodes

Other deformations such as non-uniform scaling, twisting, tapering and bending stretch the set of particles and require several steps of a Witkin-Heckbert algorithm to converge to a more uniformly sampled surface.

5. Results

The particle generation algorithms have been implemented in $C++$ as member functions of the classes defining the nodes of the BlobTree model. Timings were performed on a Mobile 2 GHz Pentium laptop using Microsoft Visual Studio compiler.

Model	Algorithm		Voxel	
	Time	Particles	Time	Particles
Statue	13.15	17097	2.54	17361
Wine	0.14	7384	0.40	7358
Flask	0; 42	19186	1.53	19049
Champagne	0.13	7181	0.39	7110
Candelabra	0.32	9049	0.67	9112
Table	0.21	4734	0.57	4632
Basket	1.53	24346	1.64	25118

Table 2. Timings (in seconds) for generating particles on the surface using our particle generation algorithm and a voxel decomposition of space scheme

Table 2 reports timings for generating particles over several complex models. The number of particles created on the surface is also reported. We compared our incremental algorithm with a voxel decomposition of space where particles were created for every cell straddling the implicit surface as described in Section 4.2. The resolution of the voxel

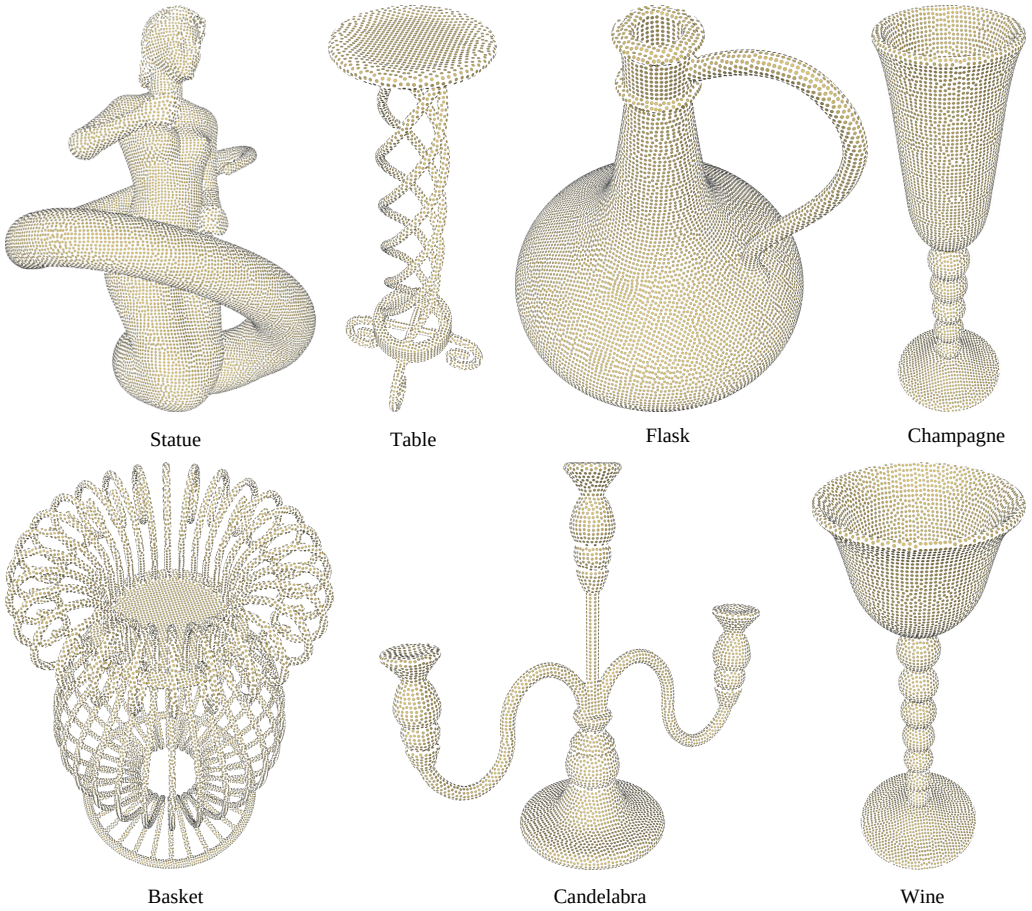

Figure 9. Some complex BlobTree models visualized by uniformly distributed particles

grid was set so that the number of particles should be approximately the same with both methods.

Timings show that our particle generation scheme performs faster than a standard voxel based surface sampling techniques, and provides a more uniform distribution of particles over the surface. The best accelerations are obtained for the Flask, Wine and Champagne models which are created by combining a few complex primitives. One notable exception to this is the statue model, which is created by blending 8486 point primitives together. As the primitives overlap very much, our recursive algorithm spends and wastes a lot of time evaluating if some nodes overlap, and accelerations provided by the procedural generation of samples for primitives and the hierarchical combination of those samples are canceled.

Table 3 shows how our algorithm performs when the average target distance between particles δ decreases. The number of generated particles, denoted as n, is proportional to the inverse squared target distance:

$$n \equiv 1/\delta^2$$

Model	$\delta = 1$	$\delta = 1/2$	$\delta = 1/3$	$\delta = 1/4$
Wine	$0'15$	$0'50$	$0'96$	$2'01$
Candelabra	$0'29$	$1'37$	$3'62$	$9'06$

Table 3. Timings (in seconds) for generating particles on the surface with a varying target distance between particles δ

The nodes of the tree combine particles generated by the skeletal primitives by computing the field function $f(\mathbf{p})$ in linear time, so the overall complexity of our method is $O(1/\delta^2)$, as demonstrated in Table 3.

6. Conclusion

We have proposed a particle system specifically designed for the BlobTree model. Our technique enables us to generate a large number of particles uniformly distributed on the

implicit surface at interactive rates. When interactively editing the BlobTree, only the nodes that have been modified regenerate particles, which enables us to incrementally update the particle system.

Blending and warping nodes need to invoke a local Witkin Heckbert scheme to smooth the particle distribution. This smoothing process is the most time consuming step of our algorithms, and would need improvements so as to take advantage of the hierarchical structure of the BlobTree. In the near future, we plan to investigate hybrid approach combining the interpolatory refinement technique proposed in [5] with our hierarchical particle generation scheme. We think that by mixing the procedural evaluation of the field function $f(\mathbf{p})$ with the interpolatory scheme, we could generate even more particles on the surface and obtain real time processing.

References

[1] B. Adams and P. Dutré. Interactive Boolean Operations on Surfel-Bounded Solids. *Computer Graphics (Siggraph Proceedings)*, **22**(3), 651–656, 2003.

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

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin and C. Silva. Computing and Rendering Point Set Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, **9**(1), 3–15, 2003.

[4] R. Allègre, A. Barbier, E. Galin and S. Akkouche. A Hybrid Shape Representation for Free-Form Modelling. *Proceedings of Shape Modeling International*, 7–18, 2004.

[5] G. Guennebaud, L. Barthe and M. Paulin. Interpolatory Refinement for Real-Time Processing of Point-Based Geometry. *Computer Graphic Forum (Eurographics Proceedings)*, **24**(3), 657–666, 2005.

[6] P. Crossno and E. Angel. Isosurface Extraction using Particle Systems. *Proceedings of the* 8^{th} *Conference on Visualization*, 495–505, 1997

[7] L. de Figueiredo, J. Gomes, D. Terzopoulos and L. Velho. Physically-based methods for polygonization of implicit surfaces. *Proceedings of Graphics Interface*, 250–257, 1992.

[8] M. Fox, C. Galbraith and B. Wyvill. Efficient use of the BlobTree for rendering purposes. *Proceedings of Shape Modelling International*, 2001.

[9] E. Galin and S. Akkouche. Incremental Polygonization of Implicit Surfaces. *Graphical Models*, **62**, 19–39, 2000.

[10] J. Hart. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer*, **12**(10), 527–545, 1996.

[11] D. Kalra and A. Barr. Guaranteed Ray Intersections with Implicit Surfaces. *Computer Graphics (Siggraph Proceedings)*, **23**(3), 297–306, 1989.

[12] T. Karkanis and A. Stewart. Curvature-Dependent Triangulation of Implicit Surface. *IEEE Computer Graphics and Applications*, **21**(2), 60–69, 2001.

[13] M. Meyer, P. Georgel and R. Whitaker. Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces *Proceedings of Shape Modeling International*, 124–133, 2005.

[14] T. Nishita and E. Nakamae. A Method for Displaying Metaballs by using Bézier Clipping. *Computer Graphics Forum (Eurographics Proceedings)*, **13**(3), 271–280, 1994.

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

[16] M. Pauly, R. Keiser, L. Kobbelt and M. Gross. Shape Modeling with Point-Sampled Geometry. *Computer Graphics (Siggraph Proceedings)*, 641–650, 2003.

[17] A. Rösch, M. Ruhl and D. Saupe. Interactive Visualization of Implicit Surfaces with Singularities. *Computer Graphics Forum*, **16**(5), 295–306, 1996.

[18] A. Sherstyuk. Fast Ray Tracing of Implicit Surfaces, *Computer Graphics Forum*, **18**(2), 139–147, 1999.

[19] J. Snyder. Interval Analysis for Computer Graphics. *Computer Graphics (Siggraph Proceedings)*, **26**(2), 121–130, 1992.

[20] R. Szeliski and D. Tonnesen. Surface Modeling with Oriented Particle Systems. *Computer Graphics (Siggraph Proceedings)*, **26**(2), 185–194, 1992.

[21] S. Tanaka, Y. Fukuda and h. Yamamoto. Sampling Implicit Surfaces based on a Stochastic Differential Equation with converging constraints. *Computer & Graphics*, **24**(3), 419–431, 2000.

[22] G. Turk. Re-Tiling Polygonal Surfaces. *Computer Graphics (Siggraph Proceedings)*, **26**(2), 55–64, 1992.

[23] A. Witkin and P. Heckbert. Using Particles to Sample and Control Implicit Surfaces. *Computer Graphics (Siggraph Proceedings)*, **28**, 269–278, 1994.

[24] W. Su and J. Hart. A Programmable Particle System Framework for Shape Modeling *Proceedings of Shape Modeling International*, 2005.

[25] G. Wyvill and A. Trotman. Ray–Tracing Soft Objects. *Proceedings of Computer Graphics International*, 467–476, 1990.

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