EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (*2005*), *Number 3*

Pen-and-Ink for BlobTree Implicit Models

K. Foster[1], P. Jepp[1], B. Wyvill[1], M. C. Sousa[1], C. Galbraith[1], J. A. Jorge[2]

[1] Dept. Computer Science, University of Calgary, Canada† [2] Dept. Information Systems and Computer Engineering, TU Lisbon, Portugal‡

Abstract

New techniques are presented for rendering complex hierarchical skeletal implicit models in several pen-and-ink styles. A particle system is employed to find interesting areas on the surface and perform stroke stylization guided by local shape features. Interesting areas include silhouette strokes and lines following local shape features, such as those caused by CSG junctions. Hidden line removal is performed either by applying a surfel technique for rapid prototyping or, more accurately, by using ray tracing. Examples drawn from simple to complex models illustrate the capabilities of our system.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation I.3.5 [Computer Graphics]: Constructive solid geometry (CSG) I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Pen-and-ink line drawing techniques are frequently used to depict form, tone and texture in artistic and scientific illustration [Hod03, Sim92, Woo79] (see Fig. 1). In non-photorealistic rendering (NPR), there has been considerable progress with the research on reproducing traditional pen-and-ink techniques for rendering 3D objects, mostly represented as polygonal surfaces [DS00,HZ00,PFS03,PHWF01, RKS00, SFWS03, WS94]. In contrast, few methods have been proposed for rendering other object representations, in particular implicit surfaces, where the focus has been on extracting and stylizing silhouette edges, and interior strokes depicting certain shape features [BH98, Elb98].

In this paper we present NPR techniques for rendering complex implicit surfaces in a style resembling traditional pen-and-ink illustrations (Fig. 1). Our approach distributes and moves particles across the implicit surface to determine stroke positions. This allows us to extract and render silhouettes and CSG feature lines as smooth stylized strokes, as well as stylized interior marks.

Unlike polygonal meshes, much is known about the surface properties of an implicit model. This can help to produce better NPR techniques as well as inform a viewer about the nature of the surface. Our approach can better support researchers and users in specific fields where implicit surfaces are the more appropriate object representation, such as

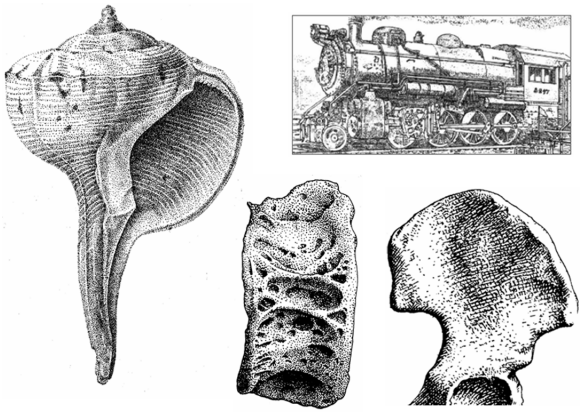

Figure 1: *Examples of real pen-and-ink artistic (train) and scientific illustrations: sea shell, fish vertebrae (courtesy of Emily S. Damstra, www-personal.umich.edu/ damstra/) and part of a hip bone.*

† {fosterk,pj,blob,mario,callum}@cpsc.ucalgary.ca

‡ jaj@inesc-id.pt

medicine, the natural sciences and engineering. It can also benefit artists who prefer to use implicit modelling tools to build smooth objects rather than using polygonal approximations. Since we only render certain parts of the visible surface, our methods have the potential to be used at interactive rates not possible with current polygonization techniques for complex implicit models.

Our work improves on previous techniques by (1) computing silhouette strokes more efficiently; (2) extracting strokes on feature lines along CSG junctions and abrupt blends (where the direction of the gradient changes rapidly); (3) providing an efficient stylization of contours and interior strokes and (4) introducing an efficient hidden line removal (HLR) approach.

2. Related Work

Pen-and-ink NPR algorithms operate mainly over images [DHvOS00, SWHS97] and 3D objects, mostly polygonal models [DS00, HZ00, PFS03, PHWF01, RKS00, SFWS03, WS94]. Relatively few papers have been devoted to pen-and-ink for implicit surfaces [BH98, Elb98, Ric73].

In one of the earliest works on implicit surfaces, Ricci used line drawings to visualize constructive solid models [Ric73]. His system extracts lines following the intersection of planar cross-sections of the surface and performs HLR.

Bremer and Hughes presented an approach for rendering implicit surfaces in a pen-and-ink style [BH98]. This method finds silhouettes by first intersecting random rays with the implicit model, then using numerical integration stepping from successful intersections toward the silhouette. Once the silhouette is found it is traced using a second numerical integration. This method used ray intersection tests for positioning short interior strokes at successful intersection points and for performing HLR. While this technique is the inspiration for our approach, we add to their method by supplying further options for stylization and rather than using random rays we support specific interior stroke placement. We also add a technique for rendering abrupt blends and constructive solid geometry (CSG) junctions.

Elber presented a particle based method for rendering implicit surfaces in a pen-and-ink style [Elb98]. In this method long flowing strokes are extracted and stylized along the interior of the implicit model or along silhouette lines. Extraction of the strokes uses either the principal curvature directions, or lines of constant angle between the surface normal and the view vector.

Akleman described a method for creating painterly renderings of implicit surfaces [Akl98a, Akl98b]. In this technique particles trace the surface, leaving a paint stroke in their path. Paint is simulated by considering the interaction of a paint brush with the paper. This method creates expressive effects by allowing the particles to move off the surface. While this gives an impression of a line drawing, it offers no pen-and-ink stylization. Akleman also describes implicit models built using CSG operations [Akl98a]. The rendering system is not separated from the model and assumes knowledge of the primitives to find and render the CSG junctions. In contrast our method sacrifices precision for efficiency and makes no assumptions about the primitives, relying only on implicit values supplied by the modelling system.

We attempt to improve the quality of rendering results by implementing ideas presented in other NPR research papers. We draw on point relaxation concepts of Pastor et al. [PFS03] which use particles attached to a polygonal mesh and a point hierarchy to create frame-coherent stipples. We also use shape- measure/threshold concepts to create and stylize strokes used in [WS94, PHWF01, SFWS03]. Finally, we apply concepts of curvature-oriented hatching [GIHL00, RKS00].

The BlobTree

An implicit surface [BW97] S, composed of the set of points $\mathbf{x} = (x, y, z)$, is derived from a potential function $f(\mathbf{x})$ as follows:

$$S = \{\mathbf{x} \in \Re^3 : f(\mathbf{x}) = iso\} \quad (1)$$

where *iso* is a constant value defining the iso-surface of interest. There are two key advantages to modelling with implicit surfaces. The first is that we can easily modify Equation 1 to define a volume (using $f(\mathbf{x}) \leq iso$ or $f(\mathbf{x}) \geq iso$). This allows for solid modeling operations to be easily applied. The second is smooth blending of implicit primitives using simple functional compositions and collision detection/response.

The BlobTree [WGG99] organizes implicit surface modelling in a manner that enables global and local operations to be exploited in a general and natural fashion. In the BlobTree, an implicit model is defined using a tree data structure which combines implicit modelling primitives as leaf nodes, and arbitrary operations as interior nodes. Evaluation of the potential function is then obtained by traversing this tree structure. Currently supported interior nodes include blending, controlled blending, bounded blending, CSG, precise contact modelling (PCM) and space warping.

For the techniques described in this paper, the potential function is treated as a *black box*. This means that the method is general and can apply to any implicit model definition where the gradient is computable everywhere (although it does not have to be continuous). The stroke extraction methods provided in this paper rely on the vector field created by the gradient $\nabla f(\mathbf{x})$ of the implicit surface and on field-test evaluations $f(\mathbf{x})$ for a surface with implicit value *iso*.

3. Overview

In our approach, shape measures are used to position and stylize strokes on an implicit surface. In particular, strokes

follow silhouette curves and feature lines created by abrupt blends and CSG discontinuities. A particle system is used to find the features of interest and then to position the strokes. Particles are placed on the surface and distributed using an attraction/repulsion method (Sec. 4). The silhouette and feature strokes are extracted using techniques based on the work of Bremer and Hughes [BH98]. In our case the trace starts from particles identified to be on the feature (Sec. 5, 6). Interior strokes are extracted at the exact positions of particles (Sec. 7). Once strokes are extracted, they can be stylized using various techniques (Sec. 8) and one of two HLR methods can be applied to remove occluded strokes (Sec. 9).

4. Placing Particles for Strokes

Stroke position is determined using particles placed on the implicit surface. The particle system is based on that presented by Witkin and Heckbert [WH94]. A summary of our method follows.

Random rays are used to initialize a predefined number n of particles on the surface, $P_i, i \in [1,n]$, with corresponding positions $\mathbf{x}_i$. This is done once as a preprocessing step. Particles are then distributed over the surface using an attractor/repulser method. The attractor force $\bar{F_i}$ pulls particles toward the surface, and the repulser force $\bar{R_i}$, repels particles from one another. $\bar{F_i}$ is defined as:

$$\bar{F_i} = (f(\mathbf{x}_i) - iso)\nabla f(\mathbf{x}_i) \tag{2}$$

The gradient of the potential field $\nabla f(\mathbf{x}_i)$, is assumed to represent the direction of the shortest path to the surface. This assumption is valid where particles are close to the surface.

The direction of the repulsion force between two particles is described by the line passing through their positions in space. The magnitude of the force is defined in terms of the distance between particles, and a distribution factor δ_i specific to each particle. A predefined maximum radius of influence r is defined, beyond which particles will not repel each other. Each particle P_i has $m_i \in [1, n-1]$ neighbouring particles with positions $\mathbf{x}_{ij}, j \in [1, m_i]$ in the radius of influence. $\bar{R_i}$ is defined as:

$$\bar{R_i} = \sum_{j=1}^{m_i} \frac{\delta_i(r - \|\mathbf{x}_i - \mathbf{x}_{ij}\|)(\mathbf{x}_i - \mathbf{x}_{ij})}{r\|\mathbf{x}_i - \mathbf{x}_{ij}\|} \tag{3}$$

The distribution factor δ_i is used to control the local density of the particles. To achieve a uniform distribution of strokes, a value of $\delta_i = 1$ is used. To place more particles in areas of high-curvature, δ_i is defined in terms of the mean curvature γ_i computed at each particle position as:

$$\delta_i = 1 - \frac{\gamma_i}{\gamma_{max}} \tag{4}$$

where γ_{max} is the maximum of all γ_i, with γ_i defined by the Hessian $h(\mathbf{x})$ of the potential function:

$$\gamma_i = \frac{1}{2}\left(\frac{trace(h(\mathbf{x}_i))}{\|\nabla f(\mathbf{x}_i)\|} - \frac{\nabla f(\mathbf{x}_i) h(\mathbf{x}_i) \nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)^3\|}\right) \tag{5}$$

The Hessian is the 3 by 3 matrix of second partial derivatives of the potential function, which are calculated numerically from the gradient. Additionally, *trace()* defines the vector created from the principal diagonal of the matrix.

To place more particles on parts of the surface closer to the viewpoint, δ_i may be defined as:

$$\delta_i = \frac{\|\mathbf{x}_i - \mathbf{x}_{eye}\|}{maxDepth} \tag{6}$$

where *maxDepth* is the maximum depth into the scene.

The repulsive force $\bar{R_i}$ will point off the surface, unless all of the particles referenced in Equation 3 lie in the plane tangent to the surface at $\mathbf{x}_i$. In areas of high curvature this can result in particles being pushed too far from the surface. To prevent this, $\bar{R_i}$ is projected onto the tangent plane yielding $\bar{R_i'}$ as:

$$\bar{R_i'} = \nabla f(\mathbf{x}_i) \times (\bar{R_i} \times \nabla f(\mathbf{x}_i)) \tag{7}$$

In each step of the simulation, each particle's position $\mathbf{x}_i$ is updated to give a new position $\mathbf{x}_i'$ as:

$$\mathbf{x}_i' = \mathbf{x}_i + \kappa_{adhere}\bar{F_i} + \kappa_{repel}\bar{R_i'} \tag{8}$$

where κ_{adhere} and κ_{repel} are user-defined scalar values. For the results in this paper, we used $\kappa_{adhere} = 0.85$ and $\kappa_{repel} = 0.5$. The choice of these values is highly dependent on the scale of the model used. For computational efficiency, all particles are stored in a regular spatial grid.

5. Silhouette Extraction

The silhouette curve for an implicit surface is defined in terms of the view vector $\bar{V}$, which for a point in space $\mathbf{x}$ is defined as:

$$\bar{V} = \frac{(\mathbf{x} - \mathbf{x}_{eye})}{\|(\mathbf{x} - \mathbf{x}_{eye})\|} \tag{9}$$

A complete continuous silhouette curve $c(t)$ exists on an implicit surface S if $c(t) \in S$ and the tangent plane to the surface at $c(t)$ contains $\bar{V}$ where $\mathbf{x} = c(t)$ for all t [BH98]. These conditions translate to:

$$f(c(t)) = iso, \forall t \tag{10}$$

$$\nabla f(c(t)) \bullet \bar{V} = 0, \forall t \tag{11}$$

To trace silhouettes, a numerical integration technique is used, based on Bremer and Hughes [BH98]. Unlike their technique, extraction begins by identifying particles P_i that lie near the silhouette using the following inequality:

$$|\bar{V} \bullet \nabla f(\mathbf{x}_i)| < \kappa_{threshold} \tag{12}$$

where $\kappa_{threshold}$ is user defined (we use 0.05). This avoids both the need to resample the surface (using ray tracing) and the numerical integration required to trace from intersection points toward the silhouette for each time-step.

Once silhouette particles have been identified, Equations 10 and 11 are used to step along the silhouette using

a predictor/corrector method. For a given point $\mathbf{x}$ on the silhouette (obtained from the list of silhouette particles), an estimate of the silhouette direction $\bar{U}$ is defined as:

$$\bar{U} = \kappa_{step} \frac{\nabla f(\mathbf{x}) \times \bar{V}}{\|\nabla f(\mathbf{x}) \times \bar{V}\|} \quad (13)$$

where κ_{step} is a user-defined step size. This estimate is used to step along the silhouette to a new point $\mathbf{x}' = \mathbf{x} + \bar{U}$. Two correction vectors are used to ensure that the stroke extraction remains on track. The first correction $U_{surface}$, corrects in the direction of the surface:

$$\bar{U}_{surface} = \kappa_{surface} \nabla f(\mathbf{x}')(iso - f(\mathbf{x}')) \quad (14)$$

where $\kappa_{surface}$ is a user-defined scalar value (we use $\kappa_{surface} = 0.5$). This equation uses the field value to scale the size of the surface corrector, as in Equation 2. The second correction $\bar{U}_{sil}$, is used to ensure that the integration follows the silhouette:

$$\bar{U}_{sil} = \kappa_{sil}(\bar{U} \times \nabla f(\mathbf{x}')(\bar{V} \bullet \nabla f(\mathbf{x}'))) \quad (15)$$

where κ_{sil} is a user-defined scalar value. As $\mathbf{x}'$ moves off the silhouette, the magnitude of $\bar{U}_{sil}$ increases, thus pulling the stroke back towards the silhouette. A new silhouette point $\mathbf{x}''$ is then defined as:

$$\mathbf{x}'' = \mathbf{x}' + \bar{U}_{sil} + \bar{U}_{surface} \quad (16)$$

Subsequent iterations repeat this procedure (Eqs. 13 to 16).

As points are calculated, their positions are saved into a fine spatial grid, denoting that a silhouette has been found in that position. This is used to prevent multiple silhouettes from being traced, and to determine when a silhouette loops.

Linking Silhouettes

The silhouette extraction process will generate looping and non-looping silhouettes. *Looping silhouettes* are identified when the silhouette extraction enters a grid cell where (1) its starting point exists and where (2) the angle between the direction towards the starting point and the $\bar{U}$ is less than a threshold. A *non-looping silhouette* is created when $\bar{U}_{sil}$ enters a cell where another silhouette already exists or when $\bar{U}_{sil}$ enters an area of an abrupt blend or a CSG junction. If the trace cannot find a subsequent chain point, the system returns to the original silhouette seed point and traces in the other direction. If the trace still cannot find a chain point the silhouette is identified as non-looping and terminated. After all silhouettes have been extracted, we link the endpoints from any two silhouette chains that are within a certain distance of one other. If there are still incomplete silhouettes after this step, the K step sizes are lowered, and extraction is attempted again.

6. Feature Line Extraction: The Dual Tracking Algorithm

The Dual Tracking Algorithm is introduced to extract lines following certain view-independent features on the surface. This uses a numerical integration inspired by that presented by Bremer and Hughes [BH98] (Sec. 5). Determination of feature lines is through the use of a *straddling* function $t(\mathbf{x}_{left}, \mathbf{x}_{right})$, which only returns true if points $\mathbf{x}_{left}$ and $\mathbf{x}_{right}$ are on opposite sides of a feature. In our implementation, the straddling function identifies points where the angle between their gradients is larger than some threshold. Thus, our straddling function identifies sharp CSG junctions and abrupt blends. The straddling function can be implemented as a traversal of the BlobTree or as a black box function. The former method isolates the dual tracking algorithm from the modelling system. The algorithm uses surface samples that straddle the junction, thus avoiding problems that can occur precisely on the discontinuity.

As with silhouette stroke extraction, this method starts with points identified by the particle system. In each step of the particle simulation, we test each particles' positions before and after the step using $t(\mathbf{x}_i, \mathbf{x}_i')$. Where this test identifies straddling points, the two positions are saved $\mathbf{x}_{left}$ and $\mathbf{x}_{right}$ (the assignment of left and right is an arbitrary one). The dual tracking algorithm then uses these two points as a starting point to extract feature lines.

6.1. Feature Line Point Determination

The first step in tracing the feature line is to determine a point $\mathbf{x}_f$ on or close to the feature. This is a variation of the Shrinkwrap method [vOW04] for approximating the intersection of two implicit contours. Initially, $\mathbf{x}_f$ is defined midway between the two straddle points. It is then corrected to lie on the surface (Fig. 2). Next, the angles between the gradients defined at points $\mathbf{x}_{left}$ (red) and $\mathbf{x}_{right}$ (blue), and the gradient at $\mathbf{x}_f$ are computed. The smaller of the two angles indicates the side of the feature on which the estimate $\mathbf{x}_f$ lies. $\mathbf{x}_f$ is used in place of the appropriate straddle point and the process is repeated until a user-defined level of accuracy is reached. $\mathbf{x}_f$ now lies approximately on the discontinuity. Note that in the BlobTree the gradient is computable everywhere, although discontinuous. If the gradient is undefined, the level of accuracy is the means of achieving a close approximation to the discontinuity.

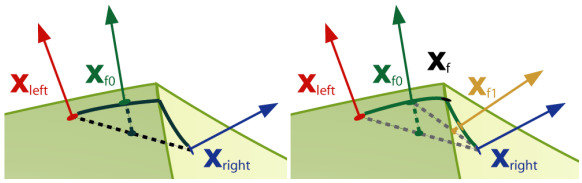

Figure 2: *Features are found by estimating and correcting $\mathbf{x}_f$ iteratively to a certain level of accuracy. $\mathbf{x}_{f0}$ and $\mathbf{x}_{f1}$ denote the current iteration of the feature point estimation*

6.2. Tracking the Feature Line

An estimate of the feature line direction is defined as the cross product of gradients at points $\mathbf{x}_{left}$ and $\mathbf{x}_{right}$:

$$\bar{D} = \kappa_{step} \nabla f(\mathbf{x}_{left}) \times \nabla f(\mathbf{x}_{right}) \quad (17)$$

New straddle points are found by moving in direction $\bar{D}$ and then correcting the points back onto the surface using Equation 7. In addition to this, a straddle-corrector $\bar{W}$ is used to keep the straddle points close to the feature line. $\bar{W}$ defines a correction in the direction of the feature line as:

$$\bar{W} = \frac{\mathbf{x}_f - \mathbf{x}_{side}}{\|\mathbf{x}_f - \mathbf{x}_{side}\|} * (\|\mathbf{x}_f - \mathbf{x}_{side}\| - \kappa_{distance}) \quad (18)$$

where *side* is $left$ or *right* and $\kappa_{distance}$ is the desired distance from the feature point. New points $\mathbf{x}'_{side}$ are then calculated with:

$$\mathbf{x}'_{side} = \mathbf{x}_{side} + \bar{D} + \bar{W} + \bar{U}_{surface} \quad (19)$$

$\bar{D}$ is a good approximation to the direction of the feature line, however, where the feature line is curved, a problem can occur. This is analogous to the situation on a *racetrack* where lanes on the inside are shorter than lanes on the outside. To overcome this problem, one straddle point is corrected to lie in the plane defined by the vector $\bar{D}$ (Fig. 3) and the other straddle point.

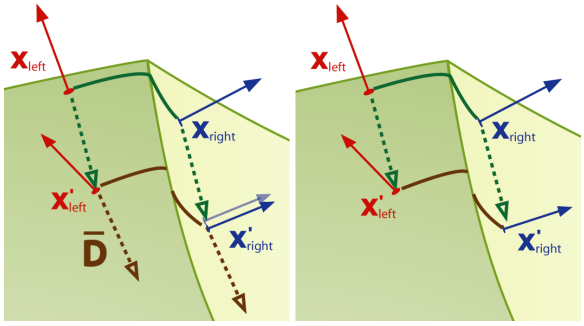

Figure 3: *Counteracting the racetrack problem: the straddle point positions are kept on a plane perpendicular to the approximation of the feature line.*

A second problem that may arise is where the displacement $\bar{D}$ can result in one of the straddle points crossing the feature line (detected using $t(\mathbf{x}_{side}, \mathbf{x}'_{side})$) (Fig. 4). In this case, the point $\mathbf{x}'_{side}$ is corrected back to the other side using the vector defined by $d = \pm(\mathbf{x}_{right} - \mathbf{x}_{left})$, where the direction used is dependent upon whether the left or right particle has crossed the feature line.

In both of these situations, particles are corrected to lie on the surface and to be close to the feature line as a final step. As points on the feature line $\mathbf{x}_f$ are calculated, their positions are saved into a fine spatial grid to prevent tracing of multiple feature lines, as with the silhouette.

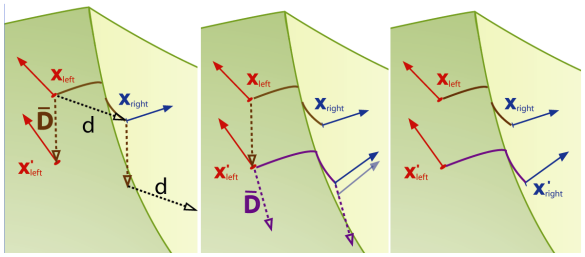

Figure 4: *When the side vector moves a straddle point to the other side of the feature, a displacement d ($\mathbf{x}_{right} - \mathbf{x}_{left}$) is used to direct it back to its correct side.*

7. Interior Stroke Extraction

We extract short interior strokes directly at particle positions. Selection of the particles that are used to draw strokes is accomplished by evaluating the following particle measures: depth, angle from the silhouette, mean curvature and lighting. Depth and curvature have already been discussed in Section 4. We now provide a brief summary of lighting and silhouette angle measures.

Lighting : We simulate point lights in our system. These are evaluated as:

$$light_i = \frac{\mathbf{x}_i - \mathbf{l}_{pos}}{\|\mathbf{x}_i - \mathbf{l}_{pos}\|} \bullet \frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|} \quad (20)$$

where $\mathbf{l}_{pos}$ is the light position.

Silhouette Angle: The silhouette angle (defined below) creates a measure which is the same as a point-light at the eye position in the scene. It is calculated as the angle between the gradient and the view direction:

$$silAngle_i = \frac{\mathbf{x}_i - \mathbf{x}_{eye}}{\|\mathbf{x}_i - \mathbf{x}_{eye}\|} \bullet \frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|} \quad (21)$$

Notice that to gain efficiency, we only calculate particle measures for front-facing particles as back-facing particles will not be visible when rendered. A particle is back-facing if $(\mathbf{x}_i - \mathbf{x}_{eye}) \bullet \nabla f(\mathbf{x}_i) < 0$.

Applying Measures to Selected Particles

Our system allows the user to select two values which define shape-measure areas, an upper-threshold *ut* and a lower threshold *lt* (Fig. 5). Any particle whose measure is less than *lt* is removed and any particle whose measure is greater than *ut* is used to create a stroke. Particles between these two values are blended for inclusion (Fig. 5). The system must blend particles in a way that appears to be random, but does not create flickering as the scene is redrawn. To do this, we use the particle's index *i* into the particle set *P* to create:

$$\varepsilon = \frac{i}{n} * (ut - lt) + lt \quad (22)$$

where n is the number of particles. The particles are included for rendering when the shape measure is greater than ε.

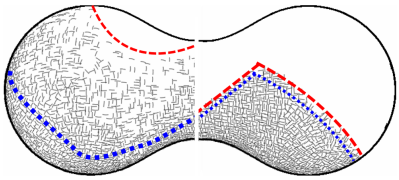

Figure 5: *Comparing different values for* ***ut*** *and* ***lt****: the blue dotted line denotes ut and the red dotted line denotes* ***lt****. Left: a large difference between* ***ut*** *and* ***lt*** *creates a smooth blend between areas with strokes and those without. Right: close* ***ut*** *and* ***lt*** *values creates a sharper transition.*

8. Stroke Stylization

Once extraction is complete strokes are rendered using OpenGL. We use simple rendering primitives, such as quads, lines and points to simulate mark-making techniques used in pen-and-ink. In this section we provide information on how strokes are stylized.

8.1. Silhouette and CSG Stylization and Rendering

We stylize silhouette and feature lines as dark ink-filled strokes that smoothly vary in width using the angled-bisector technique presented by Northrup and Markosian [NM00]. For shape feature strokes, each point $\mathbf{x}$ in a stroke is extruded and then transformed using curvature or depth to create points $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$. The ink- filled stroke is created by joining points ($\hat{\mathbf{x}}$, $\tilde{\mathbf{x}}$) with each of the points belonging to its neighbors in the feature line.

8.2. Interior Stroke Stylization

An efficient method is required for interior stylization because particles are not stationary (Sec. 4). We stylize approved particles (those selected using the technique in Sec. 7) as short curved lines or points. Stroke size can be controlled by the user or based on shape measures (Sec. 7). If points are used, our system simulates a stippling style (Fig. 6).

To create short directional strokes over the surface, a two-step process is used. First, the stroke direction is calculated. Then, it can optionally curve the stroke based on the amount of surface curvature in the stroke direction.

Stroke Direction: We create lines in three different directions (Fig. 6): the first and second principal curvature directions and the contour direction. To find the contour direction, the line orthogonal to the view direction and the gradient, we use:

$$contour_i = \nabla f(\mathbf{x}_i) \times \frac{(\mathbf{x}_i - \mathbf{x}_{eye})}{\|(\mathbf{x}_i - \mathbf{x}_{eye})\|} \qquad (23)$$

To extract the principal curvature directions our system uses Hessian analysis using the technique described in [SE02].

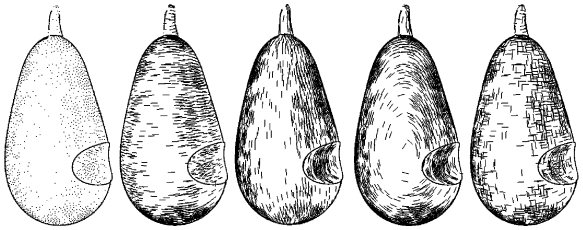

Figure 6: *From left to right: stipples, strokes in the contour direction, the first principal curvature direction (PCD), the second PCD, and in both the first and second PCDs.*

Rendering Strokes: To create strokes, our system projects two points g_0 and g_1 from each particle's position in the stroke direction as illustrated in Figure 7, left. The length of this line can be controlled by surface curvature or by the user. To allow for longer strokes, we curve strokes to follow the surface. The amount that a stroke bends is directly related to the curvature so that strokes in areas of little curvature will be drawn straight, while strokes in convex/concave areas will bend as desired (Figs. 6, 7). To accomplish this, a Bézier curve with four *control- points*, c_1, c_2, c_3, c_4 is defined:

$$c_1 = g_0 - (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \qquad (24)$$
$$c_4 = g_1 - (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \qquad (25)$$
$$c_2 = g_0 + (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \qquad (26)$$
$$c_3 = g_1 + (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \qquad (27)$$

where *scl* is the distance between g_0 and g_1 multiplied by a user selected scalar and $\gamma(\mathbf{x})$ is the surface curvature.

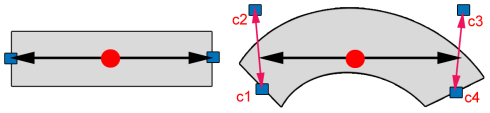

Figure 7: *Left: To model a straight stroke (represented by the grey polygon) a line is created between two endpoints projected out from the particle's position. Right: A curved stroke is modeled by extruding 4 points c_1, c_2, c_3, c_4 from each particle applying them to a Bézier function.*

9. Hidden Line Removal

We provide two methods for implicit model hidden line removal (HLR): an efficient approximation using surfels [PZvBG00] with z-buffer occlusion and a more computationally expensive, accurate approach using ray-tracing [BH98]. Both of these work for static or animated surfaces.

9.1. Surfel-Based HLR

The first approach we present to perform hidden line removal uses surfels and the z-buffer. Surfels are oriented ellipses which are used in point-based rendering to render surfaces. To use them for occlusion, we render them as white discs oriented with the gradient and displaced slightly behind each interior stroke. Displacement and orientation are used so that surfels do not improperly occlude the interior, silhouettes or feature strokes. To guarantee no artifacts a more robust HLR algorithm should be used, i.e. using ray tracing. The size of the surfels is related to the curvature of the surface.

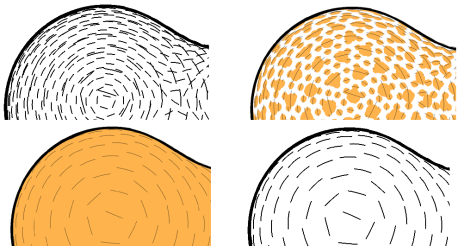

Figure 8: *Removing hidden lines with surfels, clockwise from top-left: surfels with radius 0, surfels with size 0.25 and surfels with size 0.5 in orange and white.*

The surfels must be modified at CSG junctions and abrupt blends so that they do not improperly occlude the feature line and remove strokes. We use the method described in Section 6.1 to calculate points at the feature and create a modified surfel, with the portion of it that crosses over the feature removed.

Figure 8 illustrates surfels for HLR. The size and displacement can be set by the user. This approach requires a suitable coverage of particles on the surface to provide accurate results. Despite this, it is very efficient and can remove all hidden lines from many surfaces. We have found that this technique can require significant user interaction and many particles for complex models with areas of extreme curvature, such as the seashell (Fig. 11).

9.2. Ray-Intersection HLR

The ray tracing method from Bremer and Hughes [BH98] provides a more accurate but more computationally expensive strategy for HLR. This method casts rays from points on the surface to the eye to determine if there is an object intersection and therefore whether the point on the surface is occluded or not. For silhouettes and strokes extracted by the dual-tracking algorithm (Sec. 6), our approach casts a ray from every *fourth* point in the stroke to the eye and checks for collisions with the surface. If a collision is detected, the test is repeated for the previous points to determine the point where visibility changes. The stroke is then marked as occluded until the system finds the point where it becomes visible again. The system checks only every *fourth* point for efficiency. For interior strokes, we trace a ray from one particle in each grid-cell for occlusion, and then set visibility for all particles in the cell by analyzing the 3 by 3 by 3 cell-neighbourhood surrounding a cell. If all test-particles are exclusively visible or occluded in this region then all particles in the cell are set to the same value. If test-particles in the neighbourhood are both visible and occluded then individual particles in the cell are tested for visibility.

Note that it is not necessary to perform a complete collision test for this step because we do not require the *exact* collision point; we only need to know if there is a surface between the test point and the eye. Thus, as soon as the collision test finds a point where the surface-value is greater than *iso* (Eq. 1), the occlusion method can stop. Even with this shortcut, it is computationally expensive to perform ray intersections with BlobTree implicit surfaces. Therefore this approach should be used when very accurate results are desired and computation time is not critical.

10. Results and Discussion

Our techniques successfully extract and stylize silhouettes, feature lines, and interior strokes from BlobTree implicit surfaces. Users can control parameters to generate several styles. Results from our techniques are illustrated in Figures 5-6 and 8-12.

Timings are presented In Table 1, gathered from a 3 GHz Pentium IV with Fedora Core I Linux, 1 gigabyte of RAM and OpenGL/nVIDIA Quadro FX 1300 graphics.

Model	Figure	Nodes	*Raytrace*	*Surfel*
Peanut	(5,8)	3	6	9
Pear	(6)	8	5	7
Goblet	(10)	7	5	7
Stalagmite	(12)	13344	1	2
Seashell	(11)	1497	0.1	0.2
Train	(9)	723	0.14	0.25

Table 1: *Performance results for our system. Each row lists model, number of nodes and speed (in frames per second) to extract strokes using the two HLR methods.*

Running times were calculated with 3000 particles on the surface. We found this to be an adequate number to provide a good depiction and rendition of the surface, although we used up to 10000 particles for higher quality images such as those in Figures 9 and 12.

The current system provides interactive rendering rates for simple to medium complexity models. The surfel HLR approach offers a large speed improvement over ray tracing. The train (Fig. 9) and the seashell (Fig. 11) require more computation time, primarily for silhouette and CSG stroke extraction. This increase in time is due to the complexity of field function evaluations for BlobTree models (Eq. 1).

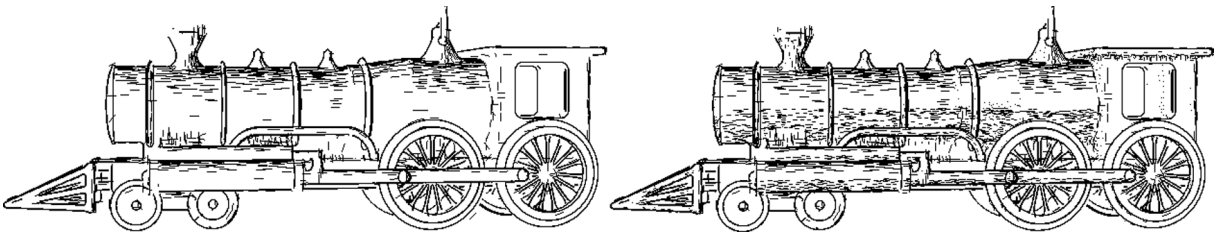

Figure 9: *Left: A train image with all three stroke types. The interior strokes follow the second principal curvature direction, placed with a light pointing from the ground up towards the train. Right: Results of adding more strokes and stipples using a light pointing downwards.*

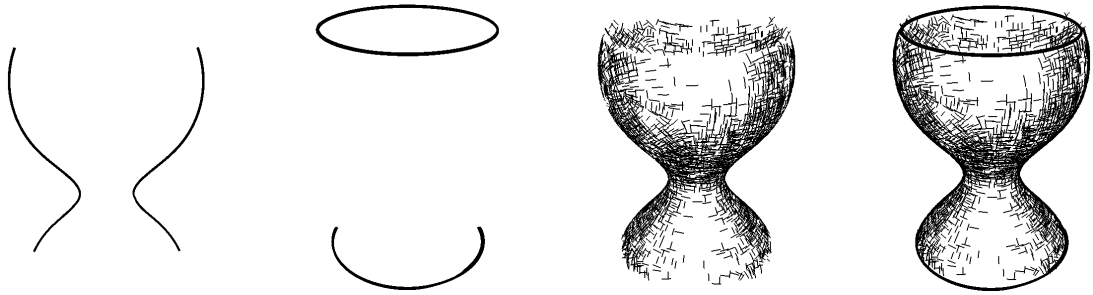

Figure 10: *A dissection of strokes extracted with out techniques from a goblet, from left to right: silhouette strokes, strokes highlighting certain feature lines and interior strokes. The rightmost image displays the results of combining all strokes.*

Our system generates promising results in approximating the pen-and-ink styles illustrated in Figure 1. We now present several important notes about our approach:

Parameter Setting: The particle system, and all three stroke-extraction methods require several parameters to be set to produce desired results. Although we have provided values that work for the models illustrated in this paper, these values are dependent on the scale of surface detail.

Particle System: Our particle system does not *guarantee* exact distributions on the entire surface (uniform by curvature or by depth) or particle adherence to the surface. For the models we used, no problems were experienced with particle adherence provided reasonable values for κ are set (Sec. 4).

Silhouette and Feature Line Strokes: Our technique does not guarantee that all silhouettes and feature line strokes will be extracted from the surface. An adequate coverage of particles on the surface is required to extract all strokes. For the models shown here, we required from 100 to 10000 particles to extract all strokes.

11. Conclusion and Future Work

We have described techniques to render complex hierarchical implicit surfaces in a pen-and-ink style. Our system can extract feature lines from the surface which can be a critical compliment to silhouette strokes (Fig. 10). Our interior stroke approach provides an efficient method to create stylized short strokes in several styles. Also, our surfel HLR approach and simplified silhouette extraction provide a way to decrease computational complexity.

It is likely that these techniques can render a few basic features to show important surface details considerably faster than a fully fledged polygonizer. One future avenue for research is to investigate the possibilities of real time interaction with complex implicit surfaces based on the approach proposed in this paper. Furthermore, this system should provide a good starting point for pen-and-ink rendering of animated implicit surfaces. A more formal evaluation with trained artistic, technical and scientific illustrators should be performed. Another useful investigation would be to compare our method with methods that guarantee extraction of all strokes.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions. We would also like to thank the many graduate student who have worked on the BlobTree, in particular Ryan Schmidt for his contributions. This research was supported by Discovery Grants from the Natural Sciences and Engineering Research Council of Canada and by the Portuguese Science Foundation (FCT) BSAB-458-2004.

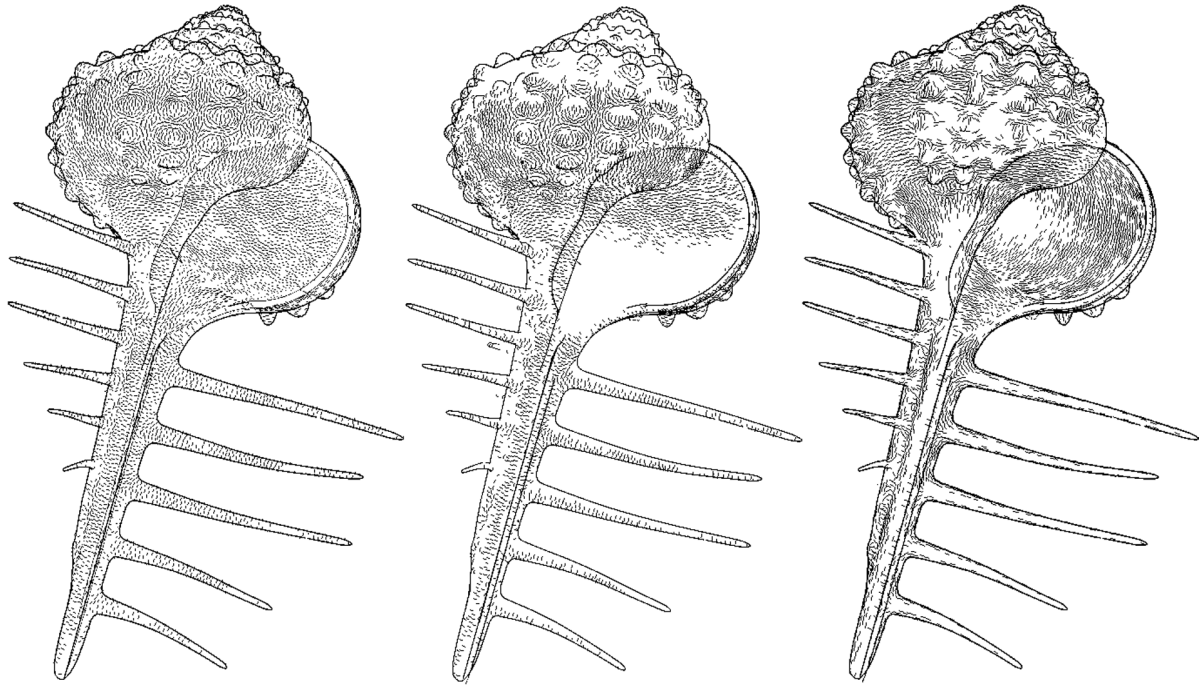

Figure 11: *A murex cabritii seashell. Left: all strokes displayed in the first and second principal curvature directions (PCD). Middle: strokes in the first PCD with a light pointing downwards at the shell. Right: strokes in the second PCD, thresholded by angle from the silhouette.*

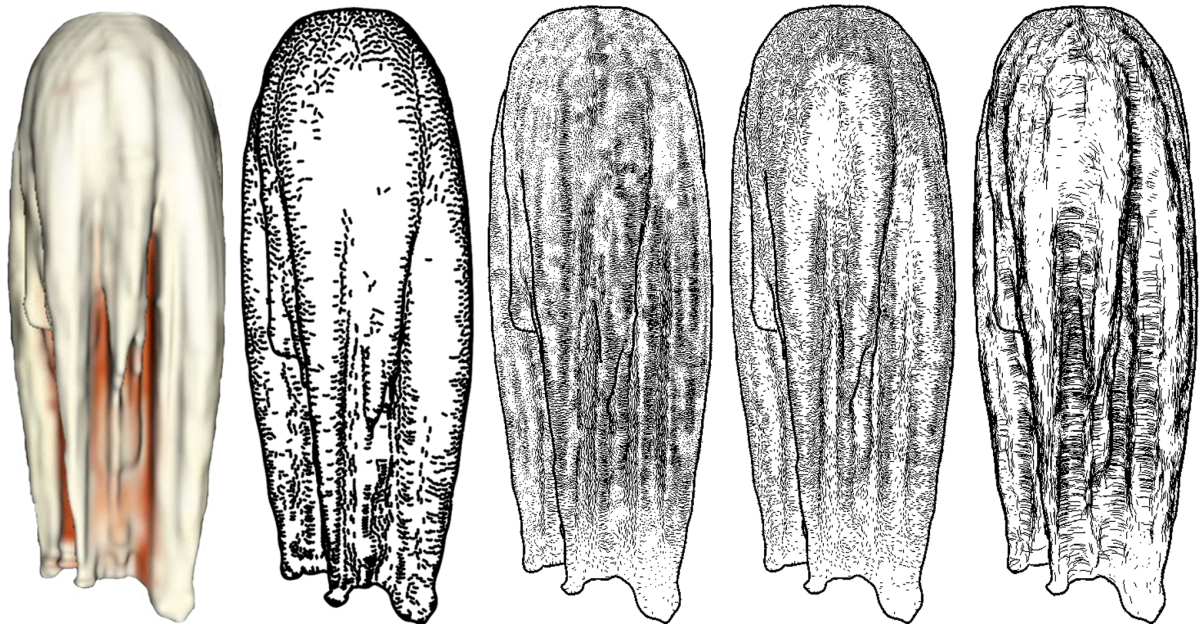

Figure 12: *From left to right: a photorealistic rendition of a stalagmite model and four images generated with our approach. The first pen-and-ink rendering uses 3000 particles and curved strokes in the first principal curvature direction (PCD). In the remaining three images, various results generated by our approach with 10000 particles creating strokes in the first PCD are displayed.*

References

[Akl98a] AKLEMAN E.: Implicit painting of CSG solids. In *Proc. of CSG '98, Set-Theoretic Solid Modelling: Techniques and Applications* (1998), pp. 99–113.

[Akl98b] AKLEMAN E.: Implicit surface painting. In *Proc. of Implicit Surfaces '98* (1998), pp. 63–68.

[BH98] BREMER D., HUGHES J.: Rapid approximate silhouette rendering of implicit surfaces. In *Proc. of Implicit Surfaces '98* (1998), pp. 155–164.

[BW97] BLOOMENTHAL J., WYVILL B. (Eds.): *Introduction to Implicit Surfaces*. Elsevier Science & Technology Books (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling), 1997. ISBN: 155860233X.

[DHvOS00] DEUSSEN O., HILLER S., VAN OVERVELD C., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum (Proc. of Eurographics '00) 19*, 3 (2000), 40–51.

[DS00] DEUSSEN O., STROTHOTTE T.: Computer-generated pen-and-ink illustration of trees. In *Proc. of SIGGRAPH '00* (2000), pp. 13–18.

[Elb98] ELBER G.: Line art illustrations of parametric and implicit forms. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 71–81.

[GIHL00] GIRSHICK A., INTERRANTE V., HAKER S., LEMOINE T.: Line direction matters: An argument for the use of principal directions in 3D line drawings. In *Proc. of NPAR '00* (2000), pp. 43–52.

[Hod03] HODGES E. R. S. (Ed.): *The Guild Handbook of Scientific Illustration, 2nd Edition*. Wiley, John Sons, Incorporated, 2003. ISBN: 0471360112.

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proc. of SIGGRAPH '00* (2000), pp. 517–526.

[NM00] NORTHRUP J., MARKOSIAN L.: Artistic silhouettes: A hybrid approach. In *Proc. of NPAR '00* (2000), pp. 31–37.

[PFS03] PASTOR O. M., FREUDENBERG B., STROTHOTTE T.: Real-time animated stippling. *IEEE Computer Graphics and Applications 23*, 4 (2003), 62–68.

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proc. of SIGGRAPH '01* (2001), pp. 579–584.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. of SIGGRAPH '00* (2000), pp. 335–342.

[Ric73] RICCI A.: A constructive geometry for computer graphics. *The Computer Journal 16*, 2 (1973), 157–160.

[RKS00] RÖSSL C., KOBBELT L., SEIDEL H.: Line art rendering of triangulated surfaces using discrete lines of curvature. In *Proc. of WSCG '00* (2000), pp. 168–175.

[SE02] SCHNEIDER P. J., EBERLY D. H.: *Geometric Tools for Computer Graphics*. Elsevier Science & Technology Books (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling), 2002. ISBN: 1558605940.

[SFWS03] SOUSA M. C., FOSTER K., WYVILL B., SAMAVATI F.: Precise ink drawing of 3D models. *Computer Graphics Forum (Proc. of Eurographics '03) 22*, 3 (2003), 369–379.

[Sim92] SIMMONS G.: *The Technical Pen.* Watson-Guptill Publications, 1992. ISBN: 0823052273.

[SWHS97] SALISBURY M., WONG M., HUGHES J., SALESIN D.: Orientable textures for image-based pen-and-ink illustration. In *Proc. of SIGGRAPH '97* (1997), pp. 401–406.

[vOW04] VAN OVERVELD K., WYVILL B.: Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface . *The Visual Computer 20*, 6 (2004), 362–369.

[WGG99] WYVILL B., GALIN E., GUY A.: Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum 18*, 2 (1999), 149–158.

[WH94] WITKIN A., HECKBERT P.: Using particles to sample and control implicit surfaces. In *Proceedings SIGGRAPH '94* (1994), pp. 269–277.

[Woo79] WOOD P.: *Scientific Illustration:A Guide to Biological, Zoological, and Medical Rendering Techniques, Design, Printing, and Display*. Van Nostrand Reinhold Company, 1979. ISBN: 0442295324.

[WS94] WINKENBACH G., SALESIN D.: Computer-generated pen-and-ink illustration. In *Proc. of SIGGRAPH '94* (1994), pp. 91–100.