Pen-and-Ink for BlobTree Implicit Models

Kevin Foster, Pauline Jepp, Callum Galbraith, Brian Wyvill, Mario Costa Sousa, Joaquim Jorge

University Of Calgary, Alberta, Canada

Abstract

In this work, new techniques for rendering hierarchical, skeletal implicit models in several pen-and-ink styles are presented. This method extracts and stylizes silhouette strokes, lines following local shape features, such as those caused by CSG junctions and abrupt blends, and short interior marks to reveal basic form. Our approach uses a particle system as a base for the stroke extraction. Stylization is performed using particle measures and rendering techniques with OpenGL. Finally, hidden lines are removed with either a raytracing or a surfel technique.

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

Traditionally, pen-and-ink line drawing techniques have been used extensively to depict form, tone and texture of a variety of subjects in the context of artistic, technical and scientific illustration [17, 22]. In non-photorealistic rendering (NPR), research has focused on the algorithmic reproduction of a variety of traditional pen-and-ink techniques for the automatic rendering of 3D models. The majority of non-photorealistic pen-and-ink renderers operate over 3D polygonal models [20, 12, 6, 10, 18, 8, 14]. In contrast, few methods have been proposed for rendering other object representations, in particular implicit surfaces. Related work in this area has been restricted to extracting and stylizing silhouette edges [4, 7], and interior strokes representing either planar cross sections of the solid [13], principal lines of curvature [7] or lines of constant angle between the surface normal and the view vector [4, 7].

In this paper we present NPR techniques for rendering complex implicit surfaces with a look resembling traditional pen-and-ink illustrations (Fig. 1). Our approach distributes and moves particles across the implicit surface to determine stroke positions. We extract and render silhouettes and CSG feature lines as long smooth stylized strokes. We also render interior marks according to thresholds and particle measures.

Unlike polygonal meshes, much is known about the surface properties of an implicit surface. This can help to produce better NPR techniques as well as inform a viewer about the nature of the surface. Our approach can benefit researchers and users in specific fields where implicit surfaces are the more appropriate object representation, such as medicine, the natural sciences and engineering. Furthermore, it can benefit artists who see fit to use implicit modeling tools to build smooth models rather than use polygonal approximations. Since we only render certain parts of the visible surface, our methods have the potential to be used at
interactive speeds 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 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 has been explored significantly in NPR, including 2D image-based techniques [5, 15], and 3D methods which either apply progressive stroke textures [20, 12] or procedurally generate strokes on surfaces [18, 10, 6, 8]. Despite this significant amount of research, relatively few papers have been devoted to pen-and-ink for implicit surfaces [13, 4, 7].

In one of the earliest works on implicit surfaces, Ricci used line drawings to visualize constructive solid models [13]. His system extracts lines following the intersection of planar cross-sections of the surface and also performs HLR.

Bremer and Hughes presented an approach for rendering implicit surfaces in a pen-and-ink style [4]. This method finds silhouettes by first intersecting random rays with the implicit model, then stepping from successful intersections toward the silhouette using numerical integration. Once the silhouette is found, it is traced using a second numerical integration. This method uses ray intersection tests for positioning small interior strokes at successful intersection points and to perform HLR. This approach offers few options for stylization and the dependence on random rays does not support specific interior stroke placement. Furthermore, the silhouette tracing algorithm requires a smooth surface, and thus is not guaranteed for more complicated implicit surface paradigms which employ abrupt blends and constructive solid geometry (CSG).

Elber presented a particle based method for rendering implicit surfaces in a pen-and-ink style [7]. In this method long flowing strokes are extracted and stylized along the interior of the implicit model. Extraction of the strokes uses either the principal direction of curvature, or lines of constant angle between the surface normal and the view vector. A limitation of this technique is that it does not perform HLR.

Akleman described a method for creating painterly renderings of implicit surfaces [1, 2]. 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 paper. This method creates expressive effects by allowing the particles to move off the surface, however it offers no pen-and-ink stylization.

None of these approaches trace strokes following important shape features such as those resulting from constructive solid geometry (CSG) operations. Furthermore, with the exception of Akleman [1, 2], these methods are presented only for simple blending surfaces. Our work attempts to address these issues. Furthermore, we attempt to improve the quality of rendering results by implementing several ideas presented in other NPR research papers. Namely, we draw on point relaxation concepts of Pastor et al. [10] who use particles attached to a polygonal mesh and a point hierarchy to create frame-coherent stipples. Furthermore, we use shape-measure/threshold concepts to create and stylize strokes used in many papers [20, 12, 18]. Finally, we apply concepts of curvature-oriented hatching [14].

2.1. The BlobTree

An implicit surface \( S \), composed of the set of points \( x = (x, y, z) \), is derived from a potential function \( f(x) \) as follows:

\[
S = \{ x \in \mathbb{R}^3 : f(x) = iso \}
\]

where iso is a constant value defining the iso-surface of interest. There are two key advantages to modeling with implicit surfaces. The first is derived from the fact that Eq. 1 is easily modified to define a volume (using \( f(x) \leq iso \) or \( f(x) \geq iso \)), which allows for solid modeling operations to be easily applied. The second is the ease with which smooth blends of component implicit models are achieved using simple functional compositions.

The BlobTree [23] paradigm has been introduced as a method of organizing implicit surface modeling in a manner that enables global and local operations to be exploited in a general and intuitive fashion. In the BlobTree, an implicit surface model is defined using a tree data structure which combines implicit model primitives as leaf nodes, and arbitrary operations as interior nodes. Evaluation of the potential function is then obtained by traversing the tree structure, which is referred to as the BlobTree. Currently supported interior nodes include blending, controlled blending, bounded blending, constructive solid geometry (CSG), precise contact modeling (PCM) and spatial 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 \( \nabla f(x) \) of the implicit surface and on field-test evaluations \( f(x) \) for a surface with implicit value \( iso \). The gradient of an implicit surface extends everywhere \( f(x) > 0 \). When used exactly on the surface, the gradient is perpendicular to the surface. For a complex object the gradient may not point directly to the surface, however the direction is generally an acceptable one.
3. Overview

In this 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. To position the strokes a particle system is used to find the features of interest. 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 [4]. 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 and two hidden line removal methods can be applied to remove occluded strokes, each with various tradeoffs (Sec. 8.3).

4. Placing Particles for Strokes

Stoke position is determined using particles placed on the implicit surface. The particle system is based on that presented by Witkin and Heckbert [21]. A summary of this method follows.

Random rays are used to initialize a predefined number of particles \( n \) on the surface, hereafter termed \( P_i, i \in [1, n] \), with corresponding positions \( x_i \). Unlike Bremer and Hughes’ method which performs this step for each rendered image, our method does this once as a preprocessing step. Particles are then distributed over the surface using an attractor/repulsor method. The attractive force \( F \) pulls particles toward the surface, and the repulsion force \( R \), repels particles from one another. \( F \) is defined as:

\[
F_i = (f(x_i) - iso) \nabla f(x_i) \tag{2}
\]

The gradient of the potential field \( \nabla f(x_i) \), 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 \( \delta \) specific to each particle. A predefined maximum radius of influence \( r \) is defined, beyond which particles will not repel each other. Each particle has \( m_i \in [0, n - 1] \) neighboring particles with positions \( x_{ij}, j \in [0, m_i] \) in the radius of influence. \( R \) is defined as follows:

\[
R_i = \sum_{j=1}^{m_i} \delta_j (r - \|x_i - x_{ij}\|) (x_i - x_{ij}) / r\|x_i - x_{ij}\| \tag{3}
\]

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

\[
\delta_j = 1 - \frac{\gamma_j}{\gamma_{max}} \tag{4}
\]

where \( \gamma_{max} \) is the maximum of all \( \gamma_j \), and \( \gamma_j \) is defined by Eq. 5 using the Hessian \( h(x) \) of the potential function. The Hessian is the 3 by 3 matrix of second partial derivatives of the potential function, which are calculated numerically from the gradient. Additionally, \( trace(f) \) defines the vector created from the principal diagonal of a matrix.

\[
\gamma_j = \frac{1}{2} \left( \frac{\text{trace}(h(x))}{\|\nabla f(x)\|} - \frac{\nabla f(x) h(x) \nabla f(x)}{\|\nabla f(x)\|^3} \right) \tag{5}
\]

To place more particles on parts of the surface closer to the viewpoint, \( \delta_i \) may be defined as:

\[
\delta_i = \frac{\|x_i - x_{eye}\|}{\text{maxDepth}} \tag{6}
\]

where \( \text{maxDepth} \) is the maximum depth into the scene.

\( R \) will point off the surface, unless all of the particles referenced in Eq. 3 lie in the plane tangent to the surface at \( x_i \). In areas of high curvature this can result in particles being pushed too far from the surface. To prevent this, \( R \) is projected onto the tangent plane yielding \( R' \) as follows:

\[
R' = \nabla f(x_i) \times (R \times \nabla f(x_i)) \tag{7}
\]

In each step of the simulation, each particle’s position \( x_i \) is updated to give a new position \( x_i' \) as follows:

\[
x_i' = x_i + \kappa_{adhere} F_i + \kappa_{repel} R' \tag{8}
\]

where \( \kappa_{adhere} \) and \( \kappa_{repel} \) are user defined scalar values, used to control the particle simulation. For the results in this paper, we used \( \kappa_{adhere} = \text{TODO} \) and \( \kappa_{repel} = \text{TODO} \). The choice of these values is highly dependent on the scale of the model used. For computational efficiency, all particles are stored in a spatial grid.

5. Silhouette Extraction

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

\[
\vec{V} = \frac{(x - x_{eye})}{\|x - x_{eye}\|} \tag{9}
\]

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 \( \vec{V} \) where \( x = c(t) \) for all \( t \) [4]. These conditions translate to:

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

\[
\nabla f(c(t)) \cdot \vec{V} = 0, \forall t \tag{11}
\]

To trace silhouettes, the numerical integration from Bremer and Hughes [4] is used. Unlike their technique, extraction
begins by identifying particles $P_i$ that lie near the silhouette using the following inequality:

$$|\vec{V} \cdot \nabla f(\vec{x})| < k_{\text{threshold}}$$

(12)

where $k_{\text{threshold}}$ is user defined. 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, Eq. 10 and 11 are used to to step along the silhouette using a predictor/corrector method. For a given point $\vec{x}$ on the silhouette (obtained from the list of silhouette particles), an estimate of the silhouette direction $\vec{U}$ is defined as follows:

$$\vec{U} = k_{\text{step}} \frac{\nabla f(\vec{x}) \times \vec{V}}{\|\nabla f(\vec{x}) \times \vec{V}\|}$$

(13)

where $k_{\text{step}}$ is a user defined step size. This estimate is used to step along the silhouette to a new point $\vec{x}' = \vec{x} + \vec{U}$. To ensure that the stroke extraction remains on track, two correction vectors are used. The first correction $\vec{U}_{\text{surf}}$, corrects in the direction of the surface as follows:

$$\vec{U}_{\text{surf}} = k_{\text{surf}} \nabla f(\vec{x}') (\text{iso} - f(\vec{x}'))$$

(14)

where $k_{\text{surf}}$ is a user defined scalar value. This equation also uses the field value to scale the size of the step taken back towards the surface, as in Eq. 2. The second correction $\vec{U}_{\text{st}}$, is used to ensure that the integration follows the silhouette:

$$\vec{U}_{\text{st}} = k_{\text{st}} (\vec{U} \times \nabla f(\vec{x}')) (\vec{V} \cdot \nabla f(\vec{x}'))$$

(15)

where $k_{\text{st}}$ is a user defined scalar value. As $\vec{x}'$ moves off the silhouette, the magnitude of $\vec{U}_{\text{st}}$ increases, thus pulling the stroke back toward the silhouette. A new silhouette point $\vec{x}''$ is then defined as follows:

$$\vec{x}'' = \vec{x}' + \vec{U}_{\text{st}} + \vec{U}_{\text{surf}}$$

(16)

Subsequent iterations repeat this procedure from Eq. 13.

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 $\vec{U}$ is less than a threshold. A non-looping silhouette is created when $\vec{U}_{\text{st}}$ enters a cell where another silhouette already exists or when $\vec{U}_{\text{st}}$ enters an area of sharp curvature or a CSG junction and the extraction fails. In both of these cases, the system returns to the original silhouette point and traces in the silhouette in the other direction. After all silhouettes have been extracted, we link the endpoints from any two silhouette chains that are within a certain distance of each 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 [4] (Sec. 5). Determination of feature lines is done through the use of a straddling function $f(\vec{x}_{\text{left}}, \vec{x}_{\text{right}})$, which only returns true if points $\vec{x}_{\text{left}}$ and $\vec{x}_{\text{right}}$ are on opposite sides of a feature. In our implementation, the straddling function identifies points where the angle between their gradients is larger than a threshold. Thus, our straddling function identifies sharp CSG junctions and abrupt blends. Although this function is treated as a black box, giving the dual tracking algorithm a general nature, it can also be implemented as a simple trace traversal in the BlobTree. Since the implicit function is not differentiable at CSG junctions, we take samples to estimate the general region in which the gradient discontinuity exists. The dual tracking 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 $f(\vec{x}, \vec{x}')$. Where this test identifies straddling points, the two positions are saved $\vec{x}_{\text{left}}$ and $\vec{x}_{\text{right}}$ (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 on or close to the feature $\vec{x}_f$. This is achieved using a variation of the Shrinkwrap method [19] for approximating the intersection of two implicit contours. Initially, $\vec{x}_f$ is defined as the midpoint of the two straddle points which is corrected to lie on the surface (Fig. 2). Next the angles between the gradients defined at points $\vec{x}_{\text{left}}$ and $\vec{x}_{\text{right}}$, and the gradient at $\vec{x}_f$ are computed. The smaller of the two angles indicates the side of the feature that the estimate $\vec{x}_f$ is on. This process is then repeated using $\vec{x}_f$ and the straddle point on the other side of the feature line (Fig. 2) until a user-defined level of accuracy is reached.

#### 6.2. Tracking the feature line

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

$$\vec{D} = k_{\text{step}} \nabla f(\vec{x}_{\text{left}}) \times \nabla f(\vec{x}_{\text{right}})$$

(17)
Figure 2: Features are found by estimating and correcting recursively to a certain level of accuracy.

New straddle points are found by moving in direction $\vec{D}$ and then correcting the points back onto the surface using Eq. 7. In addition to this, a straddle-corrector $\vec{W}$ is used to keep the straddle points close to the feature line. $\vec{W}$ defines a correction in the direction of the feature line as follows (where side is left or right):

$$\vec{W} = \frac{x_f - x_{side}}{\|x_f - x_{side}\|} \ast (\|x_f - x_{side}\| - \kappa_{distance})$$ (18)

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

$$x'_{side} = x_{side} + \vec{D} + \vec{W} + \vec{U}_{surface}$$ (19)

$\vec{D}$ is a good approximation to the direction of the feature line, however, situations arise where the displacement can result in one of the straddle point crossing the feature line (detected using $t(x_{side}, x'_{side})$) (Fig. 3). In these cases, the point $x'_{side}$ is corrected back to the other side using the vector defined by $\pm (x_{right} - x_{left})$, where the direction used is dependent upon whether the left or right particle has crossed the feature line.

Where the feature line is curving, the racetrack 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 $\vec{D}$ (Fig. 4) and the other straddle point.

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

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 Sec. 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:

$$\text{light}_i = \frac{x_i - \text{light}}{\|x_i - \text{light}\|} \ast \frac{\nabla f(x_i)}{\|\nabla f(x_i)\|}$$ (20)

where $\text{light}$ is the light position.

**Angle from Silhouette:** The angle from the silhouette 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 to the view direction:

$$\text{silAngle}_i = \frac{x_i - x_{eye}}{\|x_i - x_{eye}\|} \ast \frac{\nabla f(x_i)}{\|\nabla f(x_i)\|}$$ (21)

**Notes:** To gain efficiency, we only calculate particle measures for front-facing particles since back-facing particles will not be visible when rendered. A particle is back-facing if $(x_i - x_{eye}) \ast \nabla f(x_i) > 0$.

**Applying Measures to Select Particles:** Our system al-
allows the user to select two values, an upper-threshold $ut$ and a lower threshold $lt$, which define shape-measure areas (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:

$$et = \frac{i}{n} * (ut - lt) + lt$$

(22)

where $n$ is the number of particles. To create a blended effect between $ut$ and $lt$, particles are included for rendering when the shape measure for particle $i$ is greater than $et$.

8. Stroke Stylization

Once stroke extraction is complete, they 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 and describe two hidden line removal methods (Sec. 8.3).

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 [9]. For shape feature strokes, each point $x$ in a stroke is extruded in opposite angled-bisector directions, scaled by curvature or depth, to create points $\hat{x}$ and $\tilde{x}$. The ink-filled stroke is created by joining points $(\hat{x}, \tilde{x})$ 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 must be redrawn every frame to display particle motion. We stylize approved particles (those selected using the technique in Sec. 7) as short curved lines or points. In both cases, 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). They are used in conjunction with line strokes and short marks in Fig. 9, 11 and 12.

To create short directional strokes over the surface, a two-step process is used. First, the system calculates the stroke direction. Then, it can then 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 directions of curvature and the contour direction. To find the contour direction, the line orthogonal to the view direction and the gradient, we use:

$$contour = \nabla f(x) \times \frac{(x_l - x_{eye})}{\| (x_l - x_{eye}) \|}$$

(23)

To extract the principal directions of curvature our system uses Hessian analysis using the technique described in [16].

**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 Fig. 7, left. The length of this line can be controlled by surface curvature or by the user. When this length is kept short, the system can produce reasonable output. Unfortunately, straight strokes with longer lengths do not adequately describe the surface. 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 (Fig. 6). To accomplish this, the system draws a Bézier curve with four control-points, $c_1, c_2, c_3, c_4$ defined as follows:

$$c_1 = g_0 - (\nabla f(x) \times scl \cdot \gamma(x))$$

(24)

$$c_2 = g_0 + (\nabla f(x) \times scl \cdot \gamma(x))$$

(25)

$$c_3 = g_1 - (\nabla \gamma(x) \times scl \cdot \gamma(x))$$

(26)

$$c_4 = g_1 + (\nabla \gamma(x) \times scl \cdot \gamma(x))$$

(27)

where $scl$ is the distance between $g_0$ and $g_1$ multiplied by a user selected scalar and $\gamma(x)$ is the surface curvature.
8.3. Hidden line removal

We provide two methods for implicit model hidden line removal: an efficient approximation using surfels \cite{11} with z-buffer occlusion and a more computationally expensive exact approach using raytracing \cite{4}. Both of these work for static or animated surfaces.

8.3.1. Surfel HLR

The first approach we present to perform hidden line removal uses surfels and the z-buffer to occlude strokes. 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 in their immediate vicinity.

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 Sec. 6.1 to calculate points at the feature and create a modified surfel, with the portion of it that crosses over the feature removed.

Fig. 8 illustrates sizing the surfels for HLR. In implementation, the size and displacement must be set by the user based on the number of particles in the system. This approach requires a suitable coverage of particles on the surface to provide accurate results. Despite this, it is very efficient and can properly remove all hidden lines from many surfaces. We have found that this technique can require significant user interaction and many particles for surfaces with areas of extreme curvature, such as the seashell (Fig. 11). Furthermore, this method is not guaranteed to remove hidden strokes, a property provided by the next method.

8.3.2. Ray-intersection HLR

The raytracing method from Bremer and Hughes \cite{4} provides a more accurate, but also much more computationally expensive avenue for HLR. This method casts rays from points on the surface to the eye to determine if there is an object intersection and, thus, whether the point on the surface is occluded or not. For silhouettes and strokes extracted by the dual-tracking algorithm, 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 only checks 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-neighborhood surrounding a cell. If all test-particles are exclusively visible or occluded in this region, all particles in the cell are set to the same value. If test-particles in the neighborhood are both visible and occluded, 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, the occlusion method can stop. Even with this shortcut, it is computationally expensive to perform ray intersections with BlobTree paradigm implicit surfaces. Thus, this approach should be used when very accurate results are desired and computation time is not critical.

9. 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 Fig. 5-6 and 8-12.

We provide timings, gathered from a 3 GHz Pentium IV with Red Hat Linux, 1 gigabyte of RAM and OpenGL/nVIDIA Quadro FX 1300 graphics, in Table 9. This table includes the number of BlobTree nodes for the model, a measure of the complexity of the surface. Timings are divided in the time required to completely extract all strokes (column 3), extract interior strokes with raytracing HLR (column 4) and extract interior strokes with surfel

\[ \text{Figure 7: Left: To create a straight stroke, a line is created between two endpoints projected out from the particle’s position. Right: A curved stroke is created by extruding 4 points from each particle applying them to a Bézier function.} \]

\[ \text{Figure 8: Removing hidden lines with surfels, clockwise from top-left: surfels with radius 0, surfels with size 0.25 and surfels with size 0.5 in orange and white.} \]
HLR (column 5). Running times were calculated with 3000 particles on the surface and are listed in frames per second. We found this to be an adequate number to provide a good impression of the surface, although we used up to 10000 particles for higher quality images such as those in Figs. 9 and 12.

<table>
<thead>
<tr>
<th>Model (Fig)</th>
<th>Nodes</th>
<th>All Strokes</th>
<th>Redraw (Raytrace)</th>
<th>Redraw (Surfel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peanut (5,8)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pear (6)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Goblet (10)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Stalagmite (12)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Seashell (11)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Train (9)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

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

**Parameter Setting:** The particle system, and all three stroke-extraction methods require several parameters to be set to work well. 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 $\kappa$ are set.

**Silhouette and Feature Line Strokes:** Our technique does not guarantee that all silhouettes and feature line strokes will be extracted from the surface. This is because it relies on a particle detection to initialize these extractions. Thus, an adequate coverage of particles on the surface is required to extract all strokes. For the models shown here, we required from 100 to 10,000 particles to extract all strokes.

**10. 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). Furthermore, our interior stroke approach provides an efficient avenue to create stylized short strokes in several styles and 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 full 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 of animated implicit surfaces. Furthermore, a formal evaluation of our results should be run and methods that are guaranteed to extract all strokes should be investigated.

**References**


© The Eurographics Association and Blackwell Publishing 2005.
Figure 9: Left: A train image with all three stroke types. The interior strokes follow the second principal direction of curvature, 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.


Figure 11: A murex cabritii seashell shell. Left: all strokes displayed in the principal direction of curvature. Middle: strokes in the first principal direction of curvature with a light pointing downwards at the shell. Right: strokes in the secondary principal direction of curvature, thresholded by angle from the silhouette.

Figure 12: A stalagmite model. A photorealistic polygonization of the image and four images generated with our approach. The leftmost pen-and-ink rendering uses 3000 particles and curved strokes in the primary principal direction of curvature. In the remaining three images, various results generated by our approach with 10000 particles creating strokes in the primary principal direction of curvature are displayed.

© The Eurographics Association and Blackwell Publishing 2005.