


memory and a Reality graphics board. In all the video segments, the control over $I_{i}^{Di,j}$ via $P_{i}^{Di,j}$ (Equation (4)) and $I_{i}^{Sil}$ via $P_{i}^{Sil}$ (Equation (5)) is demonstrated, as well as the ability to generate several frames per second of line art drawings in various styles. The video segments include:

- A recording of the wavy torus with three stroke styles, along isoparametric directions, along lines of curvature, and along silhouette lines. See Figure 3. The strokes along the silhouettes also include shaded display of the original wavy torus surface.
- A recording of the Utah teapot with a binary decision on the drawing of the strokes, yielding uncomfortable discontinuous appearance and disappearance of strokes is compared to a continuous drawings of strokes. See Figure 4.
- A recording of a transparent light bulb. See Figure 7.
- A recording of a chess set. See Figure 5.

References


Figure 7: A light bulb. The transparency effect in real time line art drawings is achieved by exploiting the transparent object only during the strokes’ generation stage. In (a), strokes are emphasizing the silhouette areas. In (b), light sources to the left and to the right of the light bulb are placed while the silhouettes emphasis is set to the bare minimum. In (c), strokes are following a constant layered wood style affect throughout the model. Recorded off a 150MHz R5000/GR3-XZ graphics board SGI Indy system.

This introduced capability may exploit many other techniques common in computer graphics. Because the strokes’ generation phase is not part of the interaction stage, one can invest more effort into this phase. In [8], strokes were also developed to follow the gradients of given parametric texture maps, or to follow a prescribed fixed univariate pattern. These two types of strokes’ styles can clearly be incorporated into this work, as well as any other style of strokes, for example, strokes that follow some prescribed texture functions. In [8, 15], strokes are developed for implicit surfaces of medical data sets, along lines of curvature. These line strokes can now be classified into buckets based on their normals as is done in this work for parametric surfaces, providing real time display of line art rendering of implicit surfaces.

In a similar way, the shader can be arbitrary. One can foresee shaders that emphasize geometric properties of the surface such as curvature, convex regions vs. concave regions, or lines of discontinuities. Alternatively, the shader can be modified to consider specular lighting.

This work proposed to subdivide the unit sphere using cones into spherical cap regions that form a coverage of $S^2$. If the number of strokes necessitates that, one can form a hierarchy of such subdivisions, forming a crude coverage of $S^2$ with few cones that leads to several layers of finer and finer coverages of $S^2$ that finally ends up at the smallest cones that contain the normals of the strokes. The potential benefits, in complex geometric scenes, of forming such an hierarchy compared to the single layer of cones that was employed in this work, is yet to be considered.

5 The Video

The video that is included with this paper contains several segments similar to the ones from which the various figures in this paper were extracted. The recording was conducted directly from the computer graphics hardware. All the video segments were recorded off a 180Mhz R5000 O2 SGI system with 64MB of real memory and CRM graphics graphics card. The only exception is the segment of the chess set that was recorded off a 180Mhz R10000 Onyx2 with 192MB of real
In interactive Line Art Rendering system. The strokes of the object may preserve the original object’s color, and in Figure 6, three interlocked rings are presented, each one with strokes in a different color.

One can combine the real time line art illustrations generated in this work with a shaded display of the same model, as is done for example in [3, 4]. Furthermore, one can exploit transparency emulation capabilities in the graphics hardware and display transparent objects along with line art illustrations. The simulation of transparency has a simple alternative in the context of this work. One could employ the transparent surface during the strokes’ generation stage while completely ignoring the same surface during the hidden surface removal stage. Figure 7 shows three instances of such an approach to the generation of transparency effects, for a model of a light bulb which is mostly transparent.

4 Conclusion

The meaning of real time algorithms in computer graphics is quite vague. An algorithm that several years ago went minutes to completion takes seconds today and hence approach real time interaction levels. This inconsistency suggests that one should seek a different measure for real time computer graphics algorithms. An algorithm in computer graphics should be considered a real time interactive algorithm provided it has the capability to response with several frames a second on a whole range of computing platforms. In other words, the algorithm should adapt to the computation power available and generate its output (images) at a fixed real time rate, at the possible cost of the output’s quality. Such an algorithm should also be capable of automatically exploiting the computational capabilities that will be introduced by faster machine, in the future. For example, herein, the number of line art strokes could be adjusted according to the available computational resources, making the presented algorithm feasible for use on low end Pentium based machines with software based rendering capabilities as well as high end graphics workstations.

In this work, we have presented tools that make interactive line art rendering of freeform surfaces,
Figure 4: Samples out of an interactive display of the Utah teapot on a 150MHz R5000/GR3-XZ graphics board SGI Indy system. In (a), the strokes are developed to follow shader $I_i^{DI}$ in Equation (4). In (b), the strokes are developed to follow shader $I_i^{Sil}$ in Equation (5). (c) shows the two shaders combined, following Equation (6).

Figure 5: A Chess set. Highly complex scenes can also be displayed in real time on high end machines. In (a), the silhouettes are emphasized while in (b) an illumination from the left is being added. Recorded off a 180MHz R10000/Reality Graphics Board Onyx2 SGI system.

a shaded display, in real time. Similarly, this scene of a chess set requires a high end machine to provide real time line art rendering.

Going to very low end systems with software based rendering, the line art renderings of Figure 6 was created using an implementation of the proposed algorithm on a Windows NT Pentium based
Figure 3: a wavy torus. Strokes can be drawn in different styles. Here, strokes are drawn on top of a wavy torus along isoparametric curves (a), along lines of curvature (b), and along silhouette lines (c). Recorded off a 150MHz R5000/GR3-XZ graphics board SGI Indy system.

3 Examples

In this section, we present snapshots from interactive sessions that employ the real time line art sketching presented in this work. All the figures in this section were downloaded from the displays of the different hardware platforms on which the presented algorithm was implemented with the aid of the IRIT [16] modeling environment. Strokes using the developed tools can follow, in our implementation, isoparametric curves as can be seen in Figure 3 (a), follow lines of curvature [2] (Figure 3 (b)) or lines of constant angle of the surface normal with respect to some (viewing) direction, lines we denote silhouette lines (Figure 3 (c)). These generalized silhouette lines are closely related with Isophotes [14] or line of constant illumination.

The strokes are drawn to follow the intensity that is prescribed by the shaders of Section 2.3. In Figure 4 (a), the strokes are drawn employing only diffuse lighting, $I^D$, following Equation (4). In Figure 4 (b), the strokes are drawn by taking into account only silhouette areas, $I^{Sil}$, following Equation (5). Finally, in Figure 4 (c), the two affects are combined following Equation (6). In all sequences of Figure 4, five snapshots from different viewing directions out of a similar animation sequence are presented.

The time that is necessary to generate the strokes of all the examples in this section is typically measured in seconds and in complex cases, in a minute or two. The rendering itself took a fraction of a second in all cases, satisfying the real time constraint of several frames per second. The preprocessing stage is clearly more intensive for complex strokes’ styles. Yet, generating strokes along isoparametric curves was found to be about twice as fast than the generation of strokes along silhouette lines and about three times faster than the generation of strokes along lines of curvature. Interestingly enough, in many instances the visual differences between the different strokes’ styles were found to be minor. The simplest strokes’ style of isoparametric curves served well and was surprisingly satisfactory.

The memory requirements that are imposed on the display hardware is comparable to the load that is placed by a polygonal approximation of the freeform surfaces that is typically used for shaded display of these surfaces. Simple scenes like in Figures 3 and 4 can be interacted with on low end graphics machines such as SGI Indys or even Windows NT, Pentium based systems. One can expect that complex scenes such as the one in Figure 5, would require a high end system to provide...
Algorithm 1

**Input:**
- \( S \), a freeform surface.
- \( C_m = \{ C(V_i, \alpha_m) \}_{i=1}^m \), a coverage of \( S^2 \) using \( m \) cones of opening angle \( \alpha_m \).
- \( \{ L_j \}_{j=1}^n \), unit vectors of \( n \) directional light sources in the scene.
- \( \mathcal{V} \), Viewing direction.

**RealTimeLineArt**( \( S, C_m, L_j, \mathcal{V} \ ))

begin
1. \( m \leftarrow 1; \)
   \( \{ P_i \}_{i=1}^r \leftarrow \) A uniform point set distribution of \( r \) points, covering \( S; \)
   \( \{ S_i \}_{i=1}^r \leftarrow \) A set of \( r \) strokes grown from \( \{ P_i \}_{i=1}^r ; \)
   for \( i \) from 1 to \( r \) do
      \( N_i \leftarrow \) Normal of \( S \) at \( P_i; \)
      \( s_i \leftarrow \) Arc-length of \( S_i; \)
      \( m_i \leftarrow 1; \)
      for \( l \) from 1 to \( m \) do
         if \( (\langle V_i, N_i \rangle > \cos(\alpha_m) ) \)
            \( C(V_i, \alpha_m) \leftarrow S_i; \)
      endfor;

   do
   \( \mathcal{V} \leftarrow \) Updated viewing direction;
2. \( m \leftarrow m + 1; \)
   for \( l \) from 1 to \( m \) do
      if ( cone \( C(V_i, \alpha_m) \) can contribute visible strokes ) then
         for all \( ( S_i \in C(V_i, \alpha_m) ) \) do
            if ( \( m_i \neq m \) ) then
               \( \mathcal{I}_i \leftarrow \) Evaluated shader for normal \( N_i; \)
            if ( \( \mathcal{I}_i > 0 \) ) then
               Display the relative portion \( s_i \mathcal{I}_i \) out of stroke \( S_i \)
            \( m_i \leftarrow m; \)
         endif;
      endif;
   endfor;
endwhile ( Display Line Art );
end;

S and with the aid of the hardware Z-buffer in resolving the hidden surface removal question, we are able to provide real time line art renderings. Section 3 provides several examples.
Both $I_i^{Dij}$ and $I_i^{Sil}$ are bounded between zero and one and so is $I_i$. Assume one thresholds the end result of $I_i$ to zero if $I_i < \frac{1}{2}$ and to one otherwise. Then, for a result of one, the stroke will be drawn while a truncation into a zero would cause the stroke to be skipped. Such a binary decision whether to draw or to skip a stroke could yield uncomfortable discontinuous appearance and disappearance of strokes, in real time animations. Nevertheless, $I_i^{Dij}$ and $I_i^{Sil}$ are typically fractional and continuous. Then, one could employ the continuity of $I_i$, and construct a smooth transition of the line art rendering from a completely visible stroke to one that is entirely hidden. A-priori compute the arc-length, $s_i$, of each stroke, $S_i$, and keep $s_i$ as part of the stroke. Given a fractional intensity, $0 \leq I_i \leq 1$, a relative portion of $S_i$ would be displayed, with the arc-length of $s_i \cdot I_i$.

Given a viewing direction, $V$, the process of rendering the strokes of the line art involves the evaluation of the possible contribution of each of the cones’ buckets holding the normals of the strokes and skipping the buckets with no contribution. The content of the buckets that are found to possibly hold contributing strokes are further processed. The illumination, $I_i$, at each surface point, $P_i$, that generated stroke $S_i$ is computed and $S_i$ is drawn with a length that is proportional to the intensity that is computed by the shader. Algorithm 1 summarizes this sequence of operations.

Recall that a stroke might appear in more than a single bucket due to the redundancy in the coverage of $S^2$ using cones. In order to prevent from rendering the same stroke more than once at the same display cycle, we employ a magic number that is different for each display cycle. The magic number of the current display cycle is initialized in line (1) of Algorithm 1 and the magic numbers of all the strokes is initialized in line (2). In each display cycle, the magic number is incremented by one (in line (3)). In line (4) of Algorithm 1, the two numbers are compared and only if they differ, meaning this stroke is yet to be drawn in this iteration, the stroke is displayed. In the latter case, the stroke’s magic number is updated (line (5)), preventing a second display of the stroke during this cycle. An alternative solution that is more efficient, yet introduces an uneven construction of the strokes’ distribution, simply forces the placement of every stroke, during the preprocessing stages, into exactly one cone bucket.

The ability to remove the invisible portions of the scene is crucial for depicting useful and appealing information. Real time hidden surface removal is readily available in hardware, even in low end workstations. In this work, we exploit these capabilities to render the art line scene, in real time. In order to achieve real time hidden surface removal, the freeform surfaces are a-priori approximated using piecewise linear polygons. These polygons are then rendered into the hardware $Z$-buffer in a background color affecting only the $Z$ depth in the $Z$-buffer. Then, strokes are rendered into the $Z$-buffer in the object’s original color, leaving the final rendering with the set of visible line art strokes only. The line art strokes are transformed a small amount in $Z$ toward the viewer to prevent from $Z$ fighting or collision with the previously rendered polygons of the same objects and therefore at the same $Z$ depths.

If the graphics’ hardware supports back face culling or transparency, these capabilities may be exploited in the real time line art rendering scheme we are presenting. Assume the object is closed and that the normals are pointing out of the object. Then, if $\langle N_i, V \rangle > 0$, point $P_i$ is in the back portion of the object and is clearly invisible. Furthermore, given a cone’s axis, $V_i$, if $\langle V, V_i \rangle > \sin(\alpha_m)$, then all the strokes contained in cone $C(V_i, \alpha_m)$ are invisible as well. In other words, the cones’ buckets on $S^2$ of the normals of $S$ can also aid in the efficient back face culling of the invisible strokes. If transparency is supported, hidden portions of the object might be seen as well. Moreover, transparency of surface $S$ can be emulated by drawing strokes on $S$ while ignoring $S$ in the polygonal hidden surface removal stage.

With a data structure that classifies the normal of $S$ using cones that covers the Gauss map of
vectors of the cone, \( \{V_i\}_{i=1}^m \), finding that approximately \( mF \) of the cones contain strokes that must be displayed, further processing the \( mF \) strokes in \( mF \) cones.

The above assumptions are inexact. Most noticeably, a uniform distribution in the Euclidean space, as it is constructed, would yield a uniform distribution of normals on \( S^2 \), only if \( S \equiv \gamma S^2 \), for some scaling factor \( \gamma \). In fact, this distribution can be highly non uniform, with the extreme case of \( S \) being a planar surface for which all its normals concentrate in a single point in \( \mathcal{G} \). Further, a single cone can simultaneously contain strokes that must be displayed as well as hidden strokes. Nevertheless, we seek a rough approximation on the number of cones, \( m \), to employ, and while inexact, these assumptions can serve us quite well toward this decision. Because each iteration requires one to test all \( m \) axes of all the \( m \) cones, a large number of cones would clearly outweigh the number of strokes we have, \( r \). The expectation is that \( rF \) strokes are going to be displayed and hence we should employ the largest number of cones possible without outweighing the expected drawing cost of \( rF \) strokes. Therefore, one should select \( m \) to be not more than \( rF \). Assuming \( F \approx 0.1 \), if \( r \) is in the thousands, \( m \) should be in the order of several hundreds.

In summary, if the shader expects that an angular deviation of up to \( a_m \) from the axis of the cone, \( V_l \), would yield strokes that should be displayed, the shader must evaluate the entire content of the cone’s bucket. Otherwise, the entire set of strokes in the bucket could be skipped. This coverage of the Gauss map allows one to efficiently concentrate on the strokes of interest, given the viewing direction and the light sources’ positions, as is discussed in Section 2.3.

### 2.3 The Surface Shading and the Hidden Surface Removal Problems

Let \( L_j \), \( j = 1, \ldots, n \) denote the unit vectors of \( n \) directional light sources in the scene. Unlike the traditional shader models in rendering applications that have global illumination support, herein we potentially need to strictly the effect of the shader to a small range of normal deviations. Simply put, strokes enhancing the silhouette areas must be confined to silhouette areas!

Consider the intensity of the illumination at point \( P_i \in S \) using a very simple cosine shader that assume a diffuse lighting of the model, \( I_i^{Di} \), from \( n \) light sources:

\[
I_i^{Di} = \max \left( \frac{1}{n} \sum_{j=1}^{n} (|\langle N_i, L_j \rangle|^c - T^{Di}, 0) \right)
\]  

(4)

where \( c \) is the decay factor of the diffuse lighting as we move away from the normal direction, and \( T^{Di} \) is the truncation level. For simplicity, we select \( c \) and \( T^{Di} \) to be the same for all surfaces in the scene.

The silhouette areas of the model from a given viewing direction, \( V \), are areas that one is typically interested in enhancing. We define the silhouette intensity in and near point \( P_i \in S \) as,

\[
I_i^{Sil} = \max \left( (1 - |\langle N_i, V \rangle|)^d - T^{Sil}, 0 \right).
\]  

(5)

where \( d \) is the decay factor of the silhouette enhancements as we move away from the silhouette curves, and \( T^{Sil} \) is the truncation level. For simplicity, we select \( d \) and \( T^{Sil} \) to be the same for all surfaces in the scene.

The shader employed as part of this work weighs at \( P_i \) the two factors that were introduced in Equations (4) and (5), while providing interactive and continuous control on the two truncation factors, \( T^{Di} \) and \( T^{Sil} \),

\[
I_i = \frac{I_i^{Di} + I_i^{Sil}}{2}.
\]  

(6)
In interactive Line Art Rendering

Figure 2: A uniform distribution of \( \{V_i\}_{i=1}^m \) unit vectors (here \( m = 130 \)) on \( S^2 \) in (a) can be used to cover a surface by associating a cone \( C(V_i, \alpha) \) with each such vector (b). The smallest angle \( \alpha \) for which \( \forall P \in S^2, \exists l \) such that \( P \in C(V_i, \alpha) \) is denoted a covering angle \( \alpha_m \). (c) shows the coverage of a sphere using 130 cones with \( \alpha_m = 11.3165625 \) degrees (see [10]).

<table>
<thead>
<tr>
<th>( m )</th>
<th>( \alpha_m ) (degrees)</th>
<th>( \mathcal{R} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>70.5287794</td>
<td>1.3333333</td>
</tr>
<tr>
<td>20</td>
<td>29.6230958</td>
<td>1.30704248</td>
</tr>
<tr>
<td>50</td>
<td>18.3000226</td>
<td>1.26436616</td>
</tr>
<tr>
<td>100</td>
<td>12.9360973</td>
<td>1.26898253</td>
</tr>
<tr>
<td>130</td>
<td>11.3165625</td>
<td>1.26373104</td>
</tr>
</tbody>
</table>

Table 1: Given a uniform distribution of unit vectors on the unit sphere \( S^2 \), \( \alpha_m \) is the optimal angular span (following [10]) of the cones defined over the unit vectors so that they cover \( S^2 \). \( \mathcal{R} \) presents the relative amount of redundancy in this coverage.

Table 2.2 also presents the expected redundancy, \( \mathcal{R} \), for various values of \( m \). The empirical experience that was gained throughout this work completely agrees with this somewhat surprising result of almost constant amount of redundancy for various \( m \) values and in all the examples presented throughout this work, this redundancy was found to be very persistently between 125% to 130%.

Starting with \( r \) strokes, typically in the thousands, one needs to handle only \( m \) cones or buckets. Hence, usually \( r \gg m \). Given a bucket, its cone’s direction, \( V_i \), should be evaluated by the shader of the line art renderer. Having a minimal overlap of 25%-30% between the different covering cones and assuming a uniform distribution of normals on \( S^2 \), each cone contains approximately \( \frac{r}{m} \) strokes. Further, assume that only a fraction \( 0 \geq \mathcal{F} \geq 1 \) of the strokes is to be displayed in each iteration. The likelihood of displaying one stroke in some cone, knowing that another stroke in the same cone is indeed displayed, is high. This locality suggests that one needs to evaluate the shader on the \( m \)
When the geometry is displayed, areas of special interest should be enhanced. The silhouette area, or the area near the horizon of the object, is one example for an area of special interest. Denoting the viewing direction by $V$, every stroke, $S_i$, for which $N_i$ is almost orthogonal to $V$ should be sought and classified as a silhouette area. The intensity of the illumination at some surface point $P_i$ should also affect the display of its associated stroke $S_i$. Any attempt to estimate the light intensity at $P_i$ using a shader model, is expected to depend on the normal of the surface at that location, $N_i$.

It is therefore plausible to attempt and join the strokes into groups of similar directions of normals. Given the viewing direction and the light sources’ positions, a representative from each group could then be tested in order to determine the feasibility of having strokes in the group contributing to the display.

In [10], uniform distributions of $m$ unit vectors, $m = 4, ..., 130$, on the unit sphere, $S^2$, toward optimal covering of $S^2$ are described. The problem of uniform distribution on $S^2$ has simple regular solutions only for $m = 4, 6, 8, 12, 20$ in close relation to the number of vertices of the platonic or regular polyhedra, and exploiting the positions of these vertices. Associate an infinite cone, $C(V_i, \alpha)$, of opening angle $\alpha$ with each such vector $V_i$ (See Figure 2):

**Definition 1** A set of $m$ cones $\mathcal{C} = \{C(V_i, \alpha)\}_{i=1}^{m}$ is considered a coverage for the unit sphere, $S^2$, if for every point $P \in S^2$, there exist a cone $C(V_i, \alpha) \in \mathcal{C}$, such that $P \in C(V_i, \alpha)$.

Given a uniform point distribution, one can define the optimal coverage as

**Definition 2** Given $m$ unit vectors $\{V_i\}_{i=1}^{m}$, the minimal angle $\alpha$ for which the set $\mathcal{C} = \{C(V_i, \alpha)\}_{i=1}^{m}$ covers $S^2$ is denoted the covering angle of the distribution. We call this optimal angle $\alpha_m$.

There is an obvious inverse relation between $m$ and $\alpha_m$; the larger $m$ is, the smaller $\alpha_m$ can be. Consider $m$ unit vectors, $\{V_i\}_{i=1}^{m}$, defining $m$ cones that covers $S^2$ with a covering angle of $\alpha_m$. Denote the cones of this coverage by $\mathcal{C}_m = \{C(V_i, \alpha_m)\}_{i=1}^{m}$. Given a freeform surface, $S$, its normal or Gauss map, $\mathcal{G}$ [2], is a map from $S$ to the unit sphere $S^2$. The set of normals, $\{N_i\} \subset \mathcal{G}$, that is associated with the set of strokes $\{S_i\}$ of $S$, can be grouped into buckets in the form of a set of covering cones $\{C(V_i, \alpha_m)\}_{i=1}^{m}$ as follows,

$$N_i \in C(V_i, \alpha_m) \iff \langle N_i, V_i \rangle > \cos(\alpha_m). \quad (1)$$

In other words, we are subdividing $S^2$, and therefore subdividing $\mathcal{G}$ of $S$, into (overlapping) regions of similar normal orientations. Because the cones cover $S^2$, $N_i$, must be placed in at least one cone. Nonetheless, few normals might find themselves in two or more cones, a case one must properly handle in order to prevent from drawing the same stroke several times. The amount of the introduced redundancy, having normals in more than one cone, can be easily computed.

Following [10], given $m$, one can compute the uniform point distribution on $S^2$ and derive $\alpha_m$. Table 2.2 presents some values of $\alpha_m$ for different $m$ values. Given $\alpha_m$, the area, $A_m$, of the spherical cap that is the intersection of the unit sphere and the cone, $S^2 \cap C(V_i, \alpha_m)$, equals,

$$A_m = 2\pi(1 - \cos(\alpha_m)). \quad (2)$$

Then, the amount of redundancy, $R$, equals to the accumulated areas of all $m$ cap divided by the area of a unit sphere, $4\pi$,

$$R = \frac{m A_m}{4\pi} = \frac{m(1 - \cos(\alpha_m))}{2}. \quad (3)$$
Figure 1: Line art rendering using a point coverage, following [8]. In (a), a uniform distribution of points on the surface is constructed. (b) presents the line art strokes along isoparametric directions that were developed from the point set of (a). (c) employs strokes that emphasize silhouette areas.

a major role in this line art rendering method. Therefore and because non isoparametric strokes are non rational, in general, all strokes are approximated as piecewise linear curves. Strokes are allowed to follow one major flow of direction on the surface, or vector field, such as the $u$ isoparametric direction or two major flows of directions, for example the two principal directions, along lines of curvature [2]. See [8] for the exact numerical marching procedures of the different strokes’ styles.

Each stroke can be rendered when the need comes by plotting the polyline that represents the stroke. A decision whether or not to plot stroke $S_i$ depends on the intensity of the illumination of $S$ at the surface neighborhood of $P_i$. In order to be able to evaluate this illumination in our shader in Section 2.3, the unit normal of the surface at $P_i$, denoted $N_i$, is computed for each developed stroke and is kept along with $S_i$. The illumination is evaluated only at $P_i$, and hence it is crucial to limit the stroke $S_i$ to a local neighborhood of the point $P_i$ from which $S_i$ was numerically grown and developed.

With the completion of the construction of the line art’s strokes, one must attempt to sort the strokes into a data structure that would ease and speed up the access of the relevant and visible strokes, once the light sources’ positions and a viewing directions are determined. Optimal access of the fraction of the visible rendered strokes, given the viewing and light sources’ directions, is a crucial optimization component toward interactive line art rendering. In Section 2.2, we propose such a strokes’ classification data structure.

### 2.2 The Classification of the Strokes

Every time the scene is rendered, only a fraction of the strokes is drawn. This observation suggests that given a viewing direction and light sources’ positions, one should seek a representation that would enable efficient extraction of the visible strokes out of the entire strokes’ set.
2 Algorithm

The rendering techniques known as line art or sketching based rendering methods require the ability to generate line strokes on the surfaces of the given geometry, make the strokes obey some shading model and line styles, and obviously honor the visibility relations between the different portions of the scene, all while the scene is viewed from a prescribed viewing direction.

Any attempt to handle this non photo-realistic rendering method in interactive speeds must provide answers in real time to all these stages. In this section, we examine possible solutions to the different stages, that together make real time line art rendering in interactive environments a feasible option. In Section 2.1, the question of a-priori construction of line art strokes is dealt with. Not all strokes are visible at all times and in Section 2.2, these strokes are carefully classified so that during the interaction stage, one can access the visible portions of the strokes in a highly efficient way. Finally, in Section 2.3, we consider the hidden surface removal and surface shading problems in the context of real time line art rendering, bringing together all the different pieces of the algorithm.

2.1 Constructing the Strokes

A regular wire-frame display of a polynomial or a rational surface typically exploits an a-priori computed set of isoparametric curves. Similarly, a shaded display of the same surface would necessitate an a-priori computation of a piecewise linear polygonal approximation of the surface. A line art sketch renders a single surface using thousands of line strokes, with each stroke formed out of dozens of sampled points on the freeform polynomial or rational surface. Considering that a typical model consists of tens of surfaces, and realizing that real time display requires several scene updates per second, one can immediately conclude that real time line art rendering, using contemporary computational capabilities, might be feasible only if these line strokes are a-priori computed and cached, as in wire-frame and shaded displays.

In [8], a sketching scheme has been proposed for freeform surfaces. The strokes are uniformly spread across the freeform surfaces, using a two steps algorithm. Given surface $S$, the first stage builds a uniform point distribution, $\mathcal{P} = \{P_i\}_{i=1}^n$, $P_i \in S$, computed so that it covers the entire surface area. This distribution is randomly computed for parametric surfaces by taking into account and compensating for the local change in the area-differential of the surface, $\left\| \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right\|$. For surfaces with a polygonal approximation, $\mathcal{P}$ may be derived by randomly spreading points on each polygon in quantities that are proportional to the area of the polygon. In the second step of the construction of the line art, a stroke, $S_i$, is grown from each point in the distribution, $P_i$, with the aid of local numerical marching on the freeform shape. The strokes are now completely independent of the surface’s parameterization and may be grown not only along isoparametric curves of the surface, but also along the directions of the principal curvatures [2], silhouette directions, or even arbitrary prescribed directions. Figure 1 shows a freeform line art rendering of a freeform surface that was generated using this approach. The uniform point distribution is shown in Figure 1 (a) while strokes that were developed from this point set are presented along isoparametric directions in Figure 1 (b) and along silhouette directions in Figure 1 (c).

Strokes along isoparametric directions can be represented in exact rational form. Nevertheless and as would be shortly revealed, the length of the stroke, a non rational function in general, plays
Traditionally, line rendering has been mostly seen, in computer graphics, as a simple interactive display method. More recently, line rendering was exploits as a tool to augment synthetic shaded imagery as well as self sustaining aesthetic and artistic line art drawings.

In [3, 4], steps were made toward the use of different line styles, such as line width or dashed or dotted line styles, to convey different level of occlusions, for both line and shaded drawings. In [20], ray tracing and various texture mapping based techniques are employed to hatch three dimensional models, by thresholding the result. In [6], line art drawings were similarly created exploiting halied lines, introduced to computer graphics in [1, 9], and depth cueing using line width and gray levels.

The derivation of a line art illustration directly from a two dimensional image was also explored in the image processing as well as the computer graphics communities. In [23], a system inspired by Dürer's drawings, denoted DigiDürer, has been developed to create an appealing surface coverage using line art strokes by processing a given regular image of the same scene. Beautiful results using a semi-automatic approach that honors user’s prescriptions of strokes’ styles were created in [25, 26]. In [24, 19], systems that capture and exploit depth information from the three dimensional model as a z-map are presented, allowing the user to select certain regions in the scene and paint them with prescribed surface normals and material types and to enhance special features such as discontinuities and silhouette lines. The quest for line art renderings is closely related to the problem of covering or filling the area of freeform surfaces. In [27], space filling curves are used in the image plane to produce the digital half-toning. The classical space filling curves of Peano, Hilbert, and Sierpinsky are considered.

The ability to employ three dimensional scenes toward line art illustrations is potentially more powerful than from two dimensional images. With the three dimensional model, one can, for example, employ arbitrary shaders, exploit depth information and recover casted shadows. In [7, 28, 29], three dimensional freeform models are employed toward line art renderings. Line art strokes are developed along, mostly, isoparametric curves. The spacing and/or the width of the strokes are forced to obey different shaders, shadow areas, texture maps, and obviously hidden surface regions.

In [30], uniform point distribution on implicit freeform surfaces is used to depict the surface’s shape. In [22], the uniform point distribution is derived from a polygonal approximation of the freeform surface toward painterly style pictures. In [15], improved image understanding is achieved for iso-surfaces of medical data by growing strokes along lines of curvature from a uniform distribution of points and displaying the strokes along with the shaded iso-surface. In [8], a uniform point, distribution allows the creation of line art renderings of freeform implicit and parametric surfaces using arbitrary style strokes, that do not necessarily follows isoparametric directions.

The line art rendering is closely associated with the hidden line removal problem. Numerous methods were developed to remove hidden lines from polyhedra and freeform models. Clearly, the direct solution to freeform models, with no polygonal approximation is more difficult. In [5, 13], such direct attempts have been reported. Nevertheless, the computations involved suggest that this direct solution is too difficult for real time applications. In [21], real time hidden surface removal and silhouette extraction is proposed for polyhedra models toward sketching style line art illustrations.

In this work, we draw from [8, 15, 21, 22, 30] and exploit a uniform point distribution to provide real time interaction with line art illustrations of freeform geometric models. The stroked line art image obeys shaders that can take into account both regions of great interest such as silhouette areas as well as the illumination’s intensity.

This paper is organized as follows. In Section 2, the main stages of the real time line art rendering algorithm of freeforms are presented. In Section 3, examples are considered and discussed, while we conclude in Section 4.

All the examples presented in this work, on all platforms, were created using an implementation
Interactive Line Art Rendering
of
Freeform Surfaces

Gershon Elber
Computer Science Department
Technion
Haifa 32000, Israel
Email: gershon@cs.technion.ac.il

May 25, 1998

Abstract
In recent years, synthetically created sketched based drawings and line art renderings has reached quality levels that are both esthetically pleasing and informatively enhancing. While a growing interest in this type of rendering methods has yielded successful and appealing results, the developed techniques were, for the most part, too slow to be embedded in real time interactive display.

This paper presents a line art rendering method for freeform polynomial and rational surfaces that is capable of achieving real time and interactive display. A careful pre-processing stage that combines an a-priori construction of line art strokes with proper classification of the strokes, allows one to significantly alleviate the computational cost of sketching based rendering, and enable interactive real time line art display.


Key Words: Sketches, Illustrations, Line Drawings, Freeform Surfaces, NURBS, Surface Coverage, Real Time, Interaction.

1 Introduction
In recent years, line art rendering has captured significant attention as a non photo-realistic rendering scheme that is highly aesthetic. Moreover, this rendering scheme is capable of augmenting existing shaded rendering techniques by depiction additional visual information.

Line rendering as an art goes a long way back. In classic wood carving and art [11, 17, 18], one finds that line art rendering played a major role. See, for example, the renaissance artist Albrecht Dürer.