Interactive Artistic Rendering

Matthew Kaplan **Bruce Gooch** **Elaine Cohen**

Department of Computer Science
University of Utah
http://www.cs.utah.edu/

Abstract

We present an algorithm for rendering subdivision surface models of complex scenes in a variety of artistic styles using an interactively editable particle system. The algorithm is suitable for modeling artistic techniques explicitly by the user, or automatically by the system. Our approach can simulate a large number of artistic effects due to the fact that almost any type of mark made on paper or canvas can be imitated. Any of our artistic effects is customizable by the user through a particle editing interface. The algorithm maintains complete frame-to-frame coherence, a characteristic required for good animation, and runs at interactive rates on current computer graphics workstations.

CR Categories: I.3.0 [Computer Graphics]: General; I.3.6 [Computer Graphics]: Methodology and Techniques.

Keywords: interaction, illustration, non-photorealistic rendering, silhouettes, lighting models.

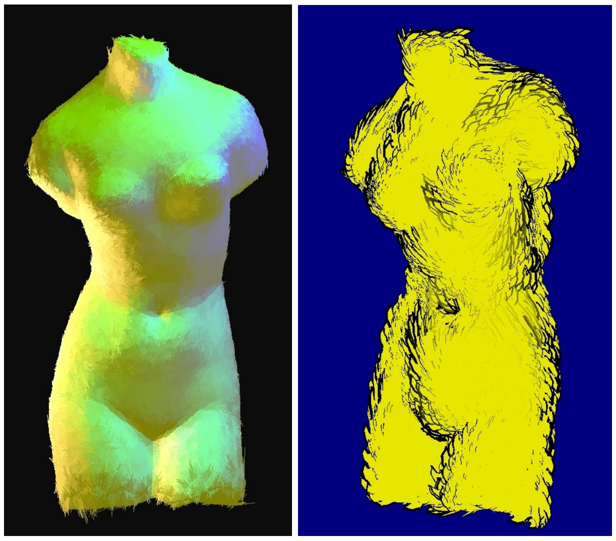


Figure 1: (a) An painterly rendering of the Venus de Milo. (b) Venus de Milo with a fur texture applied.

1 Introduction

Both humans and computer programs can take an input image, scene or geometric representation and produce an output image. Computers excel at producing photorealistic images that are difficult for humans to reproduce by hand. On the other hand, artists have developed systems for expressive abstraction to represent complex systems of texture, geometry, tone, and lighting using a few simple pen and ink or paint strokes. This allows artists to concentrate on conveying an idea or feeling instead of being forced to focus on the details that a photorealistic representation entails.

Artists use different abstractions in different circumstances. They represent scenes of staggering visual complexity such as grass, trees, fur or scales using expressive strokes that are indicative of the texture but do not describe it in complete detail. To achieve additional effects, such as dramatic lighting on a smooth surface, an artist may use a cross hatching technique to indicate the relative light on the object at any point. While these methods are only stylized approximations of the actual texture, object geometry, and lighting, they can be more expressive than photorealistic methods.

By expressive, we mean “effectively conveying the meaning or feeling”. A motivating question behind our work is : “How do human artists portray the fundamental meaning or feeling of a scene?” A goal has been to find ways in which non-artists can use computers to generate expressive images or scenes. Evidence can be found in the psychophysical research literature to support the idea that the very simplicity of hand-crafted media serves as a means of interaction between the artwork and the viewer [8]. When an artist leaves a scene incomplete, some of the marks on the medium act as “clues” for the viewer. The viewer can use these simple ideas or clues to reconstruct or complete the image by inference [14]. This type of transaction between art and viewer allows the viewer to fill in details for himself and, hence, become more involved in the image. The more detailed the image becomes, the less able the viewer is to “fill in the blanks.” When little is left to the imagination, there are fewer possibilities for meaning or feeling to be assigned to a piece of art by the viewer. The simple nature of art allows a greater personification of art on an individual level. Stroke based rendering may enable us to use a computer to emulate handmade art.

Goals of this research were an algorithm and related system framework that supports artistic rendering with complete frame-to-frame coherence at interactive rates, to allow a user to completely control particle placement, size, shape and orientation on a per object basis, to generate a variety of artistic effects of significant complexity, and to provide a framework though which other artistic techniques can be rendered with ease. The program renders scenes in a variety of artistic styles based on user defined parameters such as shading models or texture and strokes styles.

Our technique associates particles with the geometry of subdivision surface models. Particles are used to represent hand drawn

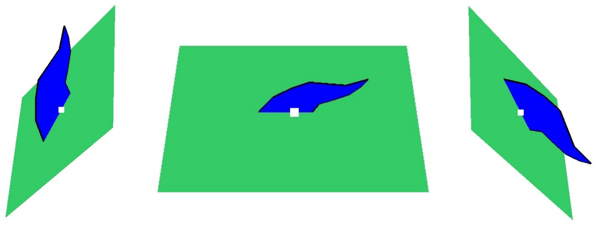


Figure 2: A blue fur/leaf geograftal has been placed on a green regular quadrilateral surface. The white dot illustrates the point at which the geograftal is located on the surface.

strokes or suggestive geometric features not present in the original model. These particles can then be interactivly edited by a user to create artistic effects. The geometry associated with the particles are rendered at interactive speeds on top of the underlying geometry based on information derived from the associated surface's position in screen space.

2 Related Work

Particle systems in general have been used extensively in photorealistic scenes to render complex textures and objects on top of simple geometric structures. Reeves [9] used particle systems to create trees and other complex objects. Fleischer *et al.* [2] used particle systems to render biologically based cellular textures. Meier [7] showed that particle based brush strokes could be used to produce painterly renderings. Her work is a precursor to our technique, since she introduced the method of using particles to locate feature defining strokes in order to maintain complete frame-to-frame coherence.

Smith introduced the idea of graftals as parallel graph grammar languages that were used with a particle system to create complex tree models [12]. An inspiration for this research, Kowalski *et al.* [4] used graftals to render complex textures such as fur and foliage on top of simple models. Their graftal textures demonstrated how different hand drawn effects could be generated by a computer. They also introduced the idea of a graftal texture as an aggregate of individual graftals with a fixed location on the visible part of a model.

The word *graftal* has grown from Smith's original definition to describe a structure that creates surfaces via an implicit model and produces data upon request. We further generalize the definition of graftals to include procedural geometric entities. In this way we can precompute as much information as possible about the graftal, including the normal, position, and color, while waiting until runtime to calculate view or lighting dependent qualities such as size, orientation, or highlight placement and color. For clarity, in the rest of this paper we will refer to graftals which use our augmented definition as *geograftals*. Figure 2 illustrates the concept of a geograftal being applied to a surface.

Our research combines and extends the work of both Meier and Kowalski *et al.* By using geograftal objects, viewpoint invariance is maintained for purposes of animation. Moreover, since each geograftal is statically placed on a model's surface, the creator of a scene can edit the attributes of any geograftal to obtain full control over the look and feel of the hand drawn effects that are produced by our system. We show how this system can be used to automatically create other hand drawn effects such as pointillism, colored pencil, oil paintings, impressionist paintings and pen and ink drawings.

Winkenbach and Salesin [16] showed how pen and ink drawings could be created for parametric surfaces. Our system can create simulated pen and ink drawings that are similar to their results for general polyhedral meshes. Salisbury *et al.* [10] demonstrated how impressive pen and ink drawings could be rendered when the user chose the principal directions for strokes to be rendered. Elber [1] showed that line art for freeform surfaces could be displayed interactively. This paper develops methods that use the principal curvatures of the polyhedral mesh to decide stroke directions automatically, which can produce effects similar to both algorithms. Tone is defined as the relative amount of light at a surface seen by the viewer. Pen and ink renderings have the property of changing tonal values when the same strokes are drawn for a model seen at different scales. Inspired by the method of Salisbury *et al.* [11] for creating scale independent tone for pen and ink drawings, we use scaling functions on our geograftals to recreate the effect of scale independent image tone for pen and ink drawings.

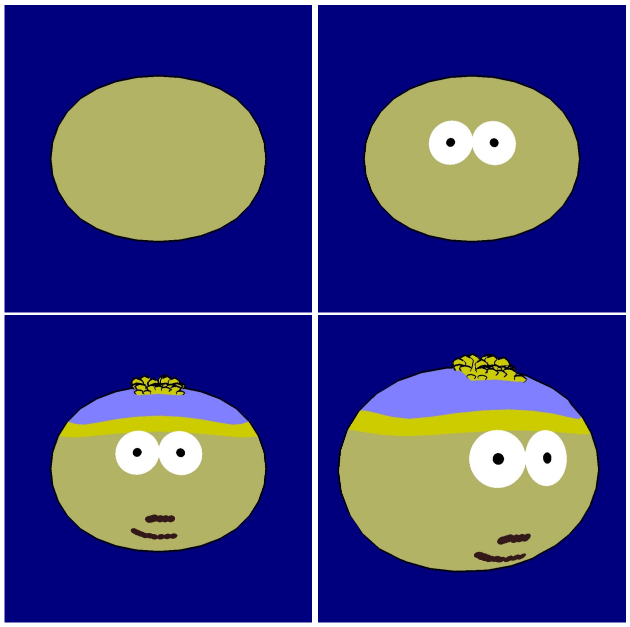


Figure 3: Geograftals are painted onto a ellipsoid model: First two geograftals representing eyes are placed. Then geograftal dots are used to paint facial features on the model. Next, an area of the model is randomly annotated with geograftal a "hair" texture, and finally the model and its associated geograftal textures are rotated. Character likeness courtesy of Comedy Central.

3 Geograftal Implementation

Section 3.1 describes the system implementation. Section 3.2 presents geograftal generation. In Section 3.3 we discuss drawing geograftals, and the effect of stored attributes and scaling functions on how the geograftals are drawn on the screen. Finally, in Section 3.4, we introduce multilayer editing of geograftals in an interactive environment to allow simple creation of more complex effects.

3.1 Software Framework

The system renders a mesh of connected quadrilateral sufaces using OpenGL. Each model within a scene is stored as a list of constituent quadrilateral surfaces which define the mesh. Individual geograftal objects are stored with each surface. Attributes such as location, width, height, type and color are stored with each geograftal object.

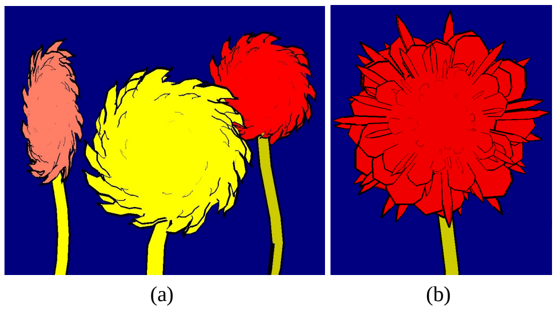


Figure 4: (a) An imitation of Dr. Seuss's truffula trees. (b) A multilayer geograftal texture.

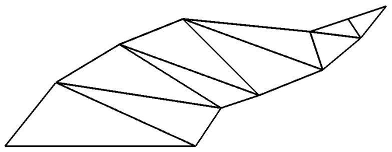


Figure 5: Geograftals are drawn as triangle strips in OpenGL. The basis vectors for this geograftal are simply unit vectors on the X and Y axes.

The type value denotes both the shape of the geograftal and stroke effects such as pen, pencil, or paint brush.

The research we present uses geometry-based procedural objects rather than the graftal texture approach used previously by others. We associate individual geograftal objects with a surface object that represents the geometry of the model. Each geograftal determines at run time how it is to be drawn on the screen. Geograftals also have associated scaling functions to control their size and shape.

3.2 Generating Geograftals

There are two goals when generating geograftal objects. One is to achieve the simulation of random placement over the surface of the object to imitate the hand drawn quality we desire. The other is to have complete control over the placement of specific geograftals for special cases. We can achieve precision placement by allowing the user to select the point on the surface to place the geograftal. We achieve a random placement on a surface by parameterizing the surface quadrilateral and then choosing random U and V values. These two cases are illustrated by Figure 3. Geograftal normals are calculated as a linear combination of the normals at the four corner points. The type, width and height attributes are specified by the user.

We produce effects similar to those in Kowalski *et al.* by choosing random geograftals, based on a user defined density, for all of the surfaces to be textured. The system also allows the user to add more geograftals interactively to create the appearance of non-uniform or denser texture coverage of the surface. Due to the initial random placement of particles, the appearance of a random texture is created. Since geograftals are associated with the surface rather than a texture, we can apply more than one distinct type of geograftal to a surface with this method, resulting in the appearance of multiple textures on the model. An example of geograftal textures is given in Figure 4(a).

3.3 Drawing Geograftals

Each geograftal is displayed as an OpenGL triangle strip and each geograftal silhouette as a OpenGL line strip. A triangle strip for a

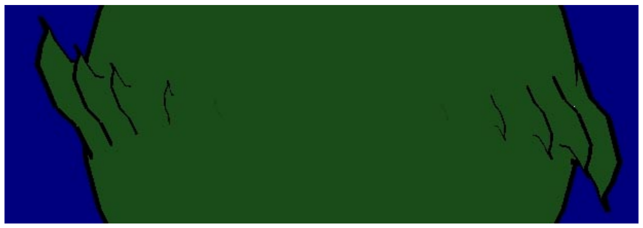


Figure 6: Leaf geograftals on this sphere are shown at even intervals along the X axis. Towards the interior, the geograftals disappear completely.

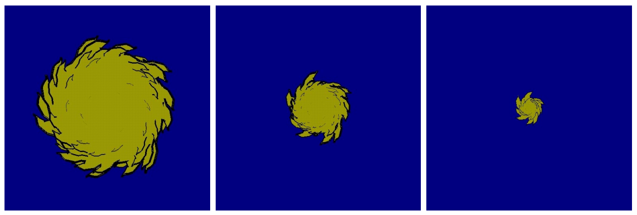


Figure 7: As objects move away from the viewer some of the geograftals shrink in size while a randomly selected few grow in size.

fur/leaf geograftal is shown in Figure 5. Points that make up these strips are described as a linear combination of two basis vectors. Rather than forming an orthonormal basis with the view vector, we use the geograftal normal and the cross product of the geograftal normal with the view vector as basis vectors. The length of the basis vectors is specified by user defined width and height values. This has the effect of creating realistic foreshortening on the geograftals while maintaining a consistent orientation via the view vector.

The hand drawn effect that we are trying to attain has the property of being most evident near silhouette edges. Therefore, geograftal objects are drawn as flush to the screen and as large as possible when the geograftal's normal nears perpendicular with the view vector. The property of foreshortening is desirable because it maximizes the screen space of geograftal objects near silhouettes and interior cusps.

Frame-to-frame coherence is maintained by drawing all of the geograftals for every frame. While it isn't necessary to render back-facing surfaces, all of the geograftals must be drawn since they may poke out from behind the object. In any case, they are more likely to be visible after the surface with which they are associated disappears or before the surface appears. If only geograftals located on front facing surfaces are drawn, jarring "popping" effects occur when a geograftal's associated surface switches between front or back facing. Popping occurs when a geograftal suddenly appears or disappears. The same disturbing popping occurs if geograftals are no longer drawn in the interior of an object. In order to create the illusion that geograftals gradually appear and disappear, we use scaling functions that affect the size of a geograftal based on the geograftal's position and orientation in relation to the view vector. Because all geograftals are drawn during every frame and because we now change the attributes of the geograftals to create the desired effects, popping no longer occurs. We define the general scaling function:

$$F_G = 1 - |\textit{Geograftal Normal} \cdot \textit{View Vector}| \qquad (1)$$

where F_G is the scaling factor. This scaling function produces a value between 0 and 1, where 0 means the geograftal normal is directly opposite the view vector and 1 means that the geograftal's normal is perpendicular to the view vector. Then, the basis vectors that define each geograftal are multiplied by F_G. This has the effect of changing the geograftal's size and shape continuously over the surface of the object. The popping effect is eliminated while maintaining the style of drawing only geograftals near silhouettes. This creates large geograftals near silhouettes and tiny geograftals in the interior.

Unfortunately, the black edge lines of the small geograftals still disrupt the effect of displaying information only near silhouette edges. We solve this problem by creating additional scaling functions. The first scales the geograftal edge width, making the edges drawn near silhouettes more prominent and those drawn in the interior less visible. This is an effect that artists strive for, making the outline of an object bold to create a cohesive effect for the whole object while giving less importance to individual details [6]. The second scaling function changes the color of the geograftal edges to match the color of the object. This creates the illusion that the geograftal has disappeared near the interior but smoothly appears as it approaches an area where it needs to be drawn. We have found that constraining F_G between 0 and 0.7 works well. This enforces color scaling only in the interior of the object and draws geograftal silhouettes near model silhouettes in black. An example of how geograftals scale their width, height and line color is shown in Figure 6.

When an object is at a greater distance from the viewer, many artists draw a smaller number of disproportionately large strokes [13]. To avoid removing and adding geograftals as they pass a distance threshold, we apply scaling functions to the geograftals with distance. We can define a smooth scaling factor for distance:

$$F_D = \textit{Actual Distance}/\textit{Maximum Distance} \tag{2}$$

where *Actual Distance* is the actual geograftal distance from the viewer and *Maximum Distance* is the maximum distance from the viewer possible in the viewing frustum. Each geograftal is assigned a random scaling function of a predetermined type that allows for some of them to grow smaller and some to grow larger with distance. To minimize the geograftal's features with distance, we multiply their feature values by F_D. To make geograftal features larger with distance, their feature values are multiplied by $1 + (1 - F_D)$. These are just two of the many scaling functions we have tested that yield pleasing results.

Scaling by F_D has the desired effect as shown in Figure 7. We found, however, that leaving the size and number of the geograftals the same while using a scaling function to decrease the width of the silhouette edges with distance may have a more pleasing effect. This interactively simulates the effect of line weight depth cueing. Line weight depth cueing establishes the distance from the viewer to the model based on the boldness of the silhouette. Artists use this method to portray three dimensions in a two-dimensional medium [6].

Quite different effects are created when non-linear functions are used to scale the geograftals. We have experimented with these types of scaling functions but none seem to be as visually pleasing as the linear examples we have presented.

3.4 Editing Geograftals

Because we specify geograftals as individual objects and geograftal textures as the sum of geograftals placed on the surfaces of the models, we are able to edit the geograftal's attributes individually or all at once and save the results. To edit an individual geograftal, the user chooses which geograftal to affect and then changes that geograftal's parameters. The edits update interactively, so the changes can be seen as they are made. This allows individual geograftals to be "sculpted" on an object as in Figure 3. With this method, distinguishing features can be designed. Since the attributes of every object are fully modifiable, this gives the user the ability to precisely model an entire scene.

Global edits are also possible with our system. By changing global attribute values, we can change the attributes of all of the geograftals or just a subset of them. Since we may want to build a geograftal texture with more than a single type of geograftal, we allow for multilayer editing. Each geograftal texture applied to a model is represented as a layer. A layer is a set of geograftals composed of a single geograftal type which are associated as a group. By selecting a layer on a multilayer texture, we can apply changes just to the geograftals that belong to that specific texture layer. This allows us to composite multiple texture layers on a single model, achieving intricate multilayer textures as shown in Figure 4(b).

4 Shading

Artists often use lighting effects to provide more information about the shape of an object. Geograftals can be used to convey the same shape information that is conveyed by traditional lighting and shading as described in Section 5.4. It is convenient to calculate and save lighting information to scale and color geograftals rather than use an automated lighting system, such as the one provided by OpenGL. The particular lighting we present is occlusion free and simulates shadows with sources of negative light. We calculate the light values for every point on the mesh and then apply smooth shading to the surfaces to obtain the appearance of a lit object. The value of the light at a surface point is only sensitive to the direction of the normal at that point relative to the light source. This describes the shape of the object uniformly over the surface without regard to shadows. We use the following non-standard equation to describe the light at any point:

$$C = O_c + \Sigma \left(L_c N \cdot L_D \right) \tag{3}$$

where O_c is object color, L_c is light color, N is the point normal, L_D is the light direction and C is the resultant color. This is a simple additive light shader that maintains the color of the object.

After clamping the value to a valid color range, we apply smooth shading to the surfaces using the light values for the points to obtain a smoothly lit model. We can obtain interactive rates for this calculation in complex scenes as reported in Table 1. Geograftals are shaded by taking their color value from the nearest point on the surface. See Figures 10(a) and 12 for examples of these shading techniques.

5 Line Drawing

A number of line drawing effects can be created by using geograftals to represent strokes on a surface. In Section 5.1 we describe a method of calculating silhouettes and internal cusps. Next, in Section 5.2 an algorithm for interactive line art is presented. Finally, in Section 5.5 this algorithm is extended to include other painting techniques.

5.1 Interactive silhouette and cusp calculation

Silhouettes are lines that describe the shape of the object near outside edges along the polyhedral mesh. Here they are defined be an edge which joins both a front and a back facing surface. We define a cusp as an edge that joins any surfaces whose normals have an angle of greater than 90 degrees between them.

Cusps can be identified in a preprocess, since they are view-independent. The cusp edges can be put in a list and displayed quickly as a set of GL_LINES.

Silhouettes, however, are defined only in relation to the view vector. One of the goals of this work is to provide scene invariance. Markosian *et al.* [5] showed how to calculate silhouettes quickly,

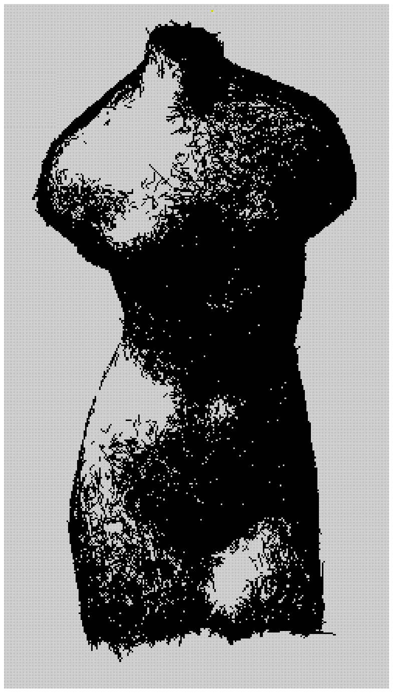


Figure 8: Venus model shaded with randomly placed geograftal strokes.

Model	Points	Polygons	0 g/p	1 g/p	100 g/p
Sphere	98	96	66	65.1	33
3 Spheres	294	288	66	61	16.5
Refined Sphere	386	384	65.8	58.1	13.2
Venus De Milo	4257	4254	62	11.9	1.44

Table 1: Model information and timings, in frames per second, on a 195Mhz MIPS R10K SGI Indigo 2 with an Extreme graphics card, for the number of geograftals per polygon. 100 g/p is presented as a pathological case.

but did not guarantee that every silhouette edge would be found. Gooch *et al.* [3] showed how silhouettes could be rendered using environment maps. The shape of such silhouettes is intriguing but does not provide predictable effects. For our purposes, a brute force search of all front-facing polygons proved fast enough. As an example, we were able to achieve 62 frames per second using a model with 4257 surfaces, displaying both surfaces and silhouettes as shown in Table 1. After calculating the silhouettes for the new viewpoint, they are put in a list and can be displayed quickly in the same fashion as the cusps. Silhouettes calculated by this method are demonstrated in Figure 9(a). The method presented by Kowalski *et al.* obtained a hand drawn look by perturbing the silhouette edges. It is clear that this method is viable here since points in the edge list can be perturbed to obtain this effect, but this may not be practical for preserving scene invariance since random perturbations of silhouettes edges would lead to inconsistent renderings of a scene.

Artists often draw silhouettes with varying widths to indicate the lighting, curvature, or distance from the viewer of the object [13]. We emulate these effects by changing silhouette line widths with a scaling function associated with the light on the surface along silhouette edges.

$$S_w = \left(1 - \left(RGB_{\text{Total}}/RGB_{\text{Max}}\right)\right) MSW \qquad (4)$$

Here, S_w is the silhouette width, RGB_{Total} is the sum of the RGB color values at the corner points of the surface, RGB_{Max} is the maximum RGB color values for those points and MSW is the maximum silhouette width. The inner term, $1 - \left(RGB_{\text{Total}}/RGB_{\text{Max}}\right)$ indicates the relative amount of light on an edge. We next multiply this by the largest pixel width value for the given edge line to obtain the actual line width for a silhouette edge S_w. This results in a silhouette width that changes smoothly with the lighting on an object as in Figure 9(b).

5.2 Automatic Line Shading

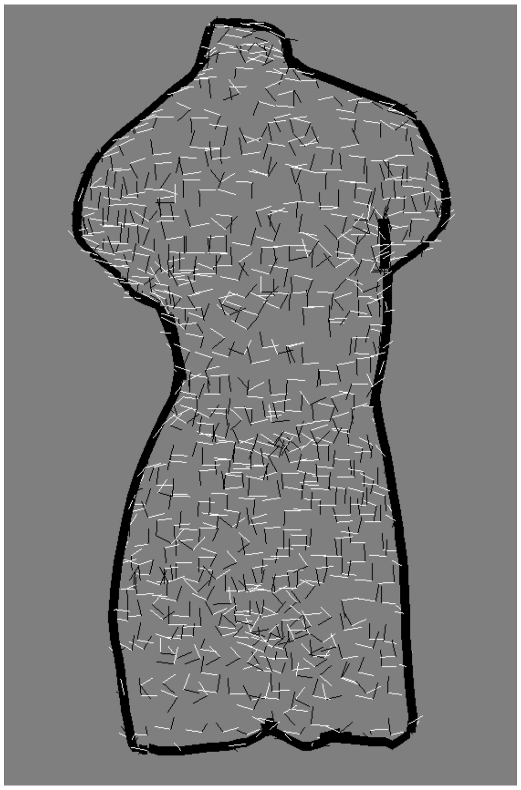
Artists often use loose and sketchy cross-hatching techniques to illustrate lighting effects on an object with black and white pen and ink strokes [13] as in Figure 8. We use graftals as stroke objects to define a texture that responds to light rather than view position. Stroke computation methods, then interactive display methods are presented.

5.3 Determining Stroke Attributes

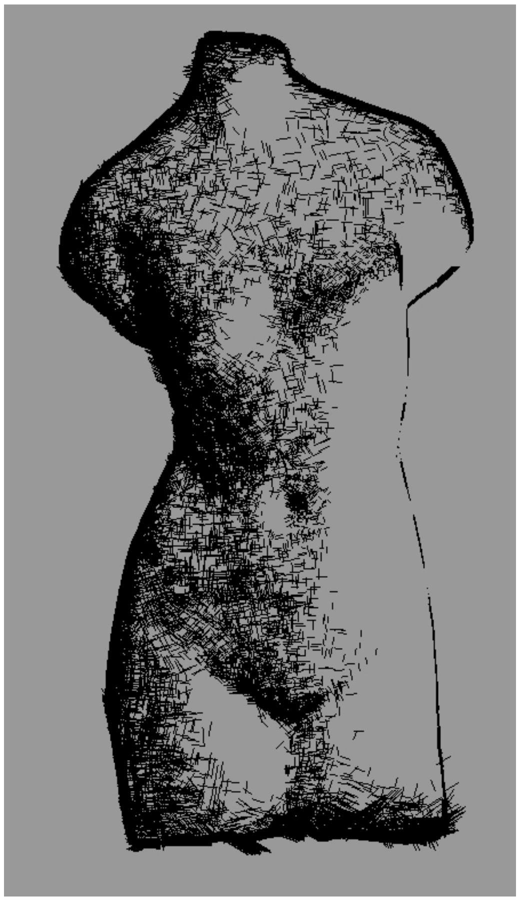
We would like strokes to appear to be randomly placed lines on the surface of the object. Represented as geograftals, strokes are randomly placed on the surface using the method described in Section 3.2. The geograftal position is used to locate the first point that defines the stroke line. There are several strategies which may be employed to find the second point that defines the stroke line segment. To give the effect of random crosshatching, we choose a random unit vector on the surface plane. Then this vector is multiplied by the stroke length and the result is added to the original position yielding a second point which defines the end of the stroke. We define the length of the stroke to equal the average of the lengths of the u, v basis vectors for the local surface quadrilateral. This length is convenient because it prevents the stroke from protruding past adjacent surfaces and affecting the perceived light of surfaces that are not immediate neighbors. This constrains the stroke to provide information about local light only. Perceived light on the model is blurred when strokes from a large number of non-local surfaces overlap.

Winkenbach *et al.* [16] previously showed that allowing a user to choose vector directions in which to draw strokes for a two dimensional scene yields pleasing results. (While this may be effective in an image-based rendering approach, we found it impractical to have the user specify directions in our system, since after rotation and translation, the vectors may not describe relevant strokes because the objects they were associated with may be in a different screen position.) It is noticeable that users of image-based approaches often choose vectors that seem to closely correspond to the principal curvatures of the surface. We automatically generate this effect by deriving an estimate of the two principal curvatures at each point in the polyhedral mesh as presented by Taubin [15]. These principal curvatures can be used to calculate the second point to represent the stroke. This method is similar in spirit to a method presented by Elber [1] and yields strokes which appear consistent with model's curvature. The principal curvature vectors for the Venus de Milo mesh are shown in Figure 9(a). The line art result of such strokes is shown in Figure 9(b).

Since strokes are represented in this system as single lines, they may overlap the edge of the mesh surface with which they are associated. Indeed, it is desirable for them to do so. If strokes were drawn entirely within each polygon, the viewer would have a clear sense of the mesh structure since no strokes would cross any edges. However, strokes rendered in this manner hang off the edges of silhouettes, leading to a loose and sketchy feel, which is often a natural side effect of hand drawn art.



(a)



(b)

Figure 9: (a) shows the principal curvatures of the mesh at each surface point. (b) shows the model shaded with curvature oriented geograftal strokes.

5.4 Stroke Shading Model

The stroke shading algorithm requires a potentially large number of geograftals for each surface. Calculating these strokes at run-time might be too costly to achieve interactive rates. In order to achieve interactive rates, we need to precompute a list of the strokes that might be needed to render tone for each object. Tone (as defined in Section 2) on a surface is represented by the number of strokes drawn on each surface quadrilateral.

We define an array of N stroke geograftals for each surface. When objects are initialized, we precompute N strokes and enter them in this array. We use the lighting values that we calculated in Section 4 to determine the number of strokes to draw to define relative tones.

$$T = \left(1 - \left(\mathrm{RGB}_{Total}/\mathrm{RGB}_{Max}\right)\right) N \tag{5}$$

Here, T is the number of strokes to draw to define the tone, RGB_{Total} is the sum of the RGB color values at the corner points of the surface, RGB_{Max} is the maximum RGB color values for those points that make up the surface. The inner term $\mathrm{RGB}_{Total}/\mathrm{RGB}_{Max}$ defines a smooth scale S, between 0 and 1 that represents the relative amount of light at a surface. T is an index into the array of the strokes. When we display the strokes, we use this index to determine how many strokes to draw. We draw each stroke whose array index is less than our index bound T. This calculation is quick to update since we never change the strokes.

This algorithm maintains interframe coherence because the same strokes are drawn under identical lighting conditions. Since only the lighting model affects the stroke index, viewpoint changes have no effect on the strokes drawn. When the strokes are drawn there is a recurrence of the popping effect. We have found that for sufficiently large N (100 seems to work well in practice), this effect is minimal since the popping occurs as a part of a smooth transition between line drawn tones within a larger texture.

Salisbury pointed out that the tone of a line drawing is changed when the drawing is scaled [11]. We use a scaling function that increases the stroke index as distance from the viewer increases. This effectively compensates for the relative tone of an object as its distance from the viewer is changed. Our simulations have shown that the results of the scaling function depends on factors such as the surface area of the polygons in the system, and the size of the viewing frustum relative to the stroke widths and the objects. We achieve our best results using scaling functions that are tailored by the user to the specific object and viewing volume. An example of this is shown in Figure 10(f). Since a precise solution to the tone matching problem may require significant run time computation for every viewpoint change, and because we attempt to maintain interactive rates, it is currently beyond the scope of this work.

5.5 Colored Line Art

A variety of colored pencil or oil painting effects are achieved by coloring the stroke lines. Since there is no inherent light or color in the black strokes applied in the previous section, the strokes were sufficient to imply the concept of light. When the strokes are colored, they no longer imply lighting. They seem to reveal holes in the model because lighting is now shown by the color of the strokes rather than by stroke placement. We present two solutions to this problem. The first solution is to shade the underlying surfaces. This is analogous to a painter laying down an underpainting to establish tone before applying detail. In this case, the strokes are used conceptually as with the geograftal objects described in Section 3. The strokes are a “hint” of a texture that allow the user to complete the rest of the texture by inference. The second solution is to increase the coverage of the strokes on each surface in an equal way. Here we allow the user to set the line index into the stroke array for every

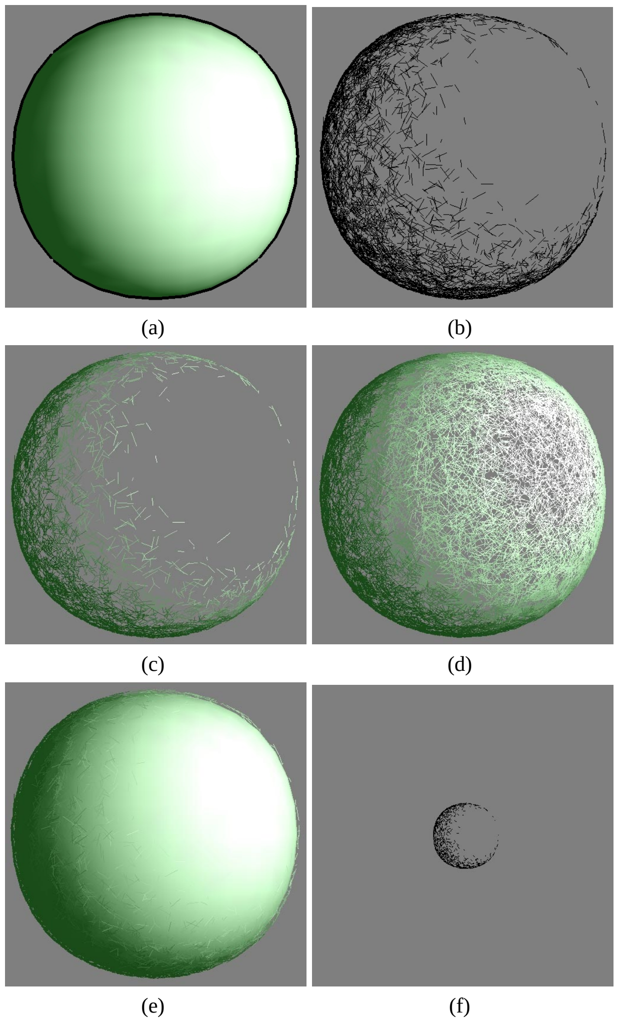

(a) (b) (c) (d) (e) (f)

Figure 10: The line art strokes that imply lighting in picture (b) imply holes in the sphere in (c). To correct this we either draw enough strokes to obtain an adequate surface coverage as in (d), or shade the surfaces on which the strokes lie as in (e). In (f), the tone of (b) has been maintained under translation by applying a scaling function.

surface to a value that is large enough to obtain a sufficient surface coverage. We can simulate different painting effects by changing global attributes such as brush type, size and orientation. Using this technique we produce effects similar to Meier's painterly rendering [7]. These techniques are illustrated in Figure 10. Other artistic techniques produced with this system such as impressionist and pointillist are shown in Figure 11.

6 A Graftal Painting System

A geograftal system can be used to paint strokes onto the surface of an object. After choosing a brush shape and size, it is possible to paint on the object by choosing the position for the brush stroke to be applied. We seed a geograftal object of the appropriate brush type and attributes at that position and apply a stroke. Unlike painting systems based on standard texture mapping, geograftals sidestep the problems of surface distortion and parametric overlapping because only the position of the geograftal is tied to the object. Because of this, geograftal features such as size and shape are not vulnerable to distortion or cropping where the shape of the model is irregular or where the parametric bounds of the texture map lie. Moreover, the strokes of the geograftal paint system are fully editable. An example of this is shown in Figure 3 where the mouth and chin of the character have been painted on by a user.

7 Future Work and Conclusion

We have presented a new technique for rendering artistic effects within a coherent framework. We have shown a stable modeling approach that encompasses and extends the artistic effects previously obtainable by various non-photorealistic rendering methods. Our system provides an automated method for rendering complex scenes with expressive artistic techniques from simple models. We have proposed a method that maintains interframe coherence by introducing the idea of geometric graftal objects to represent graftal textures and have shown that such a system can generalize to many other artistic effects. Moreover, the user of our system has complete control over each geograftal object and we provide multilayer editing techniques. The system renders at rates from 2 to 60 frames per second on a low end workstation, depending on the complexity of the scene and the effects desired. Our test scenes had meshes containing between 512 to 16,000 polygons.

Extending the range of styles currently available for geograftal rendering is one of the immediate goals of future work. It would also be valuable to formalize notions of view dependent object behavior in ways that allow for easy geograftal design by users. Since each model is represented as a subdivision surface mesh, it might be easy to achieve higher frame rates by using lower resolution meshes for rendering objects at a distance. It might also be useful to examine other intrinsic properties of the subdivision mesh to generate line strokes. Our silhouettes currently only vary line width based on lighting. Other types of data such as curvature might be useful metrics for silhouette line width variation. Furthermore, our system currently only handles straight line strokes. Strokes represented by user defined curves or bitmaps would be a valuable addition.

References

[1] Gershon Elber. Interactive line art rendering of freeform surfaces. *Computer Graphics Forum*, 18(3):1–12, September 1999. ISSN 1067-7055.

[2] Kurt Fleischer, David Laidlaw, Bena Currin, and Alan Barr. Cellular texture generation. *Proceedings of SIGGRAPH 95*, pages 239–248, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

[3] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Rich Riesenfeld. Interactive technical illustration. *1999 ACM Symposium on Interactive 3D Graphics*, pages 31–38, April 1999. ISBN 1-58113-082-1.

[4] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John Hughes. Art-based rendering of fur, grass, and trees. *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[5] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time nonphotorealistic rendering. *Proceedings of SIGGRAPH 97*, pages 415–420, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[6] Judy Martin. *Technical Illustration: Materials, Methods, and Techniques*, volume 1. Macdonald and Co. Publishers, 1989.

[7] Barbara J. Meier. Painterly rendering for animation. *Proceedings of SIGGRAPH 96*, pages 477–484, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[8] V.S. Ramachandran and William Hirstein. The science of art a neurological theory of aesthetic experience. *Journal of Consciousness Studies*, 6(6-7):15–51, 1999.

[9] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):313–322, July 1985. Held in San Francisco, California.

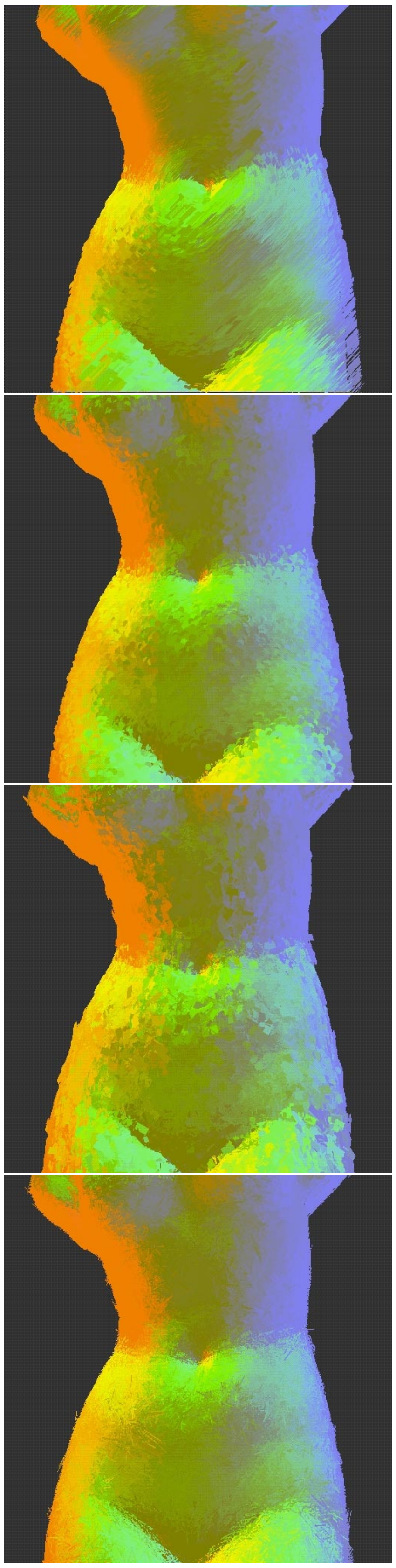


Figure 11: By interactively varying input parameters, we achieve different painterly effects. Examples shown here are, from top to bottom, oriented impressionist, pointillist, and on the bottom two pictures, random impressionist with different input parameters.

[10] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. *Proceedings of SIGGRAPH 97*, pages 401–406, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[11] Mike Salisbury, Corin Anderson, Dani Lischinski, and David H. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. *Proceedings of SIGGRAPH 96*, pages 461–468, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[12] Alvy Ray Smith. Plants, fractals and formal languages. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):1–10, July 1984. Held in Minneapolis, Minnesota.

[13] Alvy Ray Smith, Michael Wright, and James Horton. *An Introduction to Art Techniques*. DK Publishing Inc., 1995.

[14] Robert L. Solso. *Cognition and the Visual Arts*. MIT Press/Bradford Books Series in Cognitive Psychology, 1999.

[15] Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. *ICCV95*, pages 902–907, 1995.

[16] Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. *Proceedings of SIGGRAPH 96*, pages 469–476, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

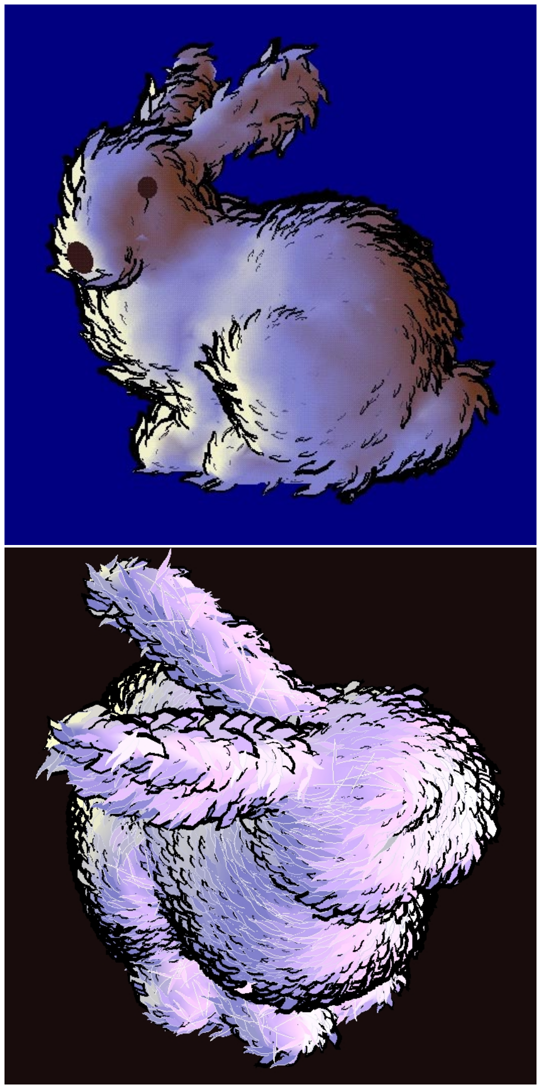


Figure 12: Bunnies with varied graftal fur textures applied. Model courtesy of Stanford University.