Rendering Parametric Surfaces in Pen and Ink

Georges Winkenbach *David H. Salesin*

Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

Abstract

This paper presents new algorithms and techniques for rendering parametric free-form surfaces in pen and ink. In particular, we introduce the idea of "controlled-density hatching" for conveying tone, texture, and shape. The fine control over tone this method provides allows the use of traditional texture mapping techniques for specifying the tone of pen-and-ink illustrations. We also show how a planar map, a data structure central to our rendering algorithm, can be constructed from parametric surfaces, and used for clipping strokes and generating outlines. Finally, we show how curved shadows can be cast onto curved objects for this style of illustration.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.6 [Computer Graphics]: Methodology and Techniques.

Additional Key Words: non-photorealistic rendering, comprehensible rendering, pen-and-ink rendering, resolution-dependent rendering, stroke textures, controlled-density hatching, outlining, shadow algorithms.

1 Introduction

In many applications—from architectural design, to medical texts, to industrial maintenance and repair manuals—a stylized illustration is often more effective than photorealism. Illustrations convey information better, consume less storage, are more easily reproduced, are more capable of conveying information at various levels of detail, and are in many respects more attractive than photorealistic images.

In a previous paper [22], we introduced a system for automatically generating pen-and-ink illustrations of three-dimensional architectural models. In that paper, we showed how many of the principles of traditional pen-and-ink rendering, such as achieving tones through texture, could be simulated algorithmically. In particular, we introduced the concept of a "prioritized stroke texture", which is used to reproduce arbitrary tones and convey textures simultaneously.

However, this earlier work was limited to polyhedral models. With curved surfaces, a number of the fundamental assumptions we used break down. Most notably, in this earlier work we assumed flat-shaded surfaces, and we used BSP trees both for creating a planar map data structure and for clipping individual strokes quickly.

In this paper, we generalize our previous work to handle curved surfaces formulated parametrically, such as B-splines surfaces, NURBS, and surfaces of revolution. We introduce a mechanism for creating "controlled-density hatching," which allows strokes to gradually disappear in light areas of a surface or in areas where too many strokes converge together, and allows new strokes to gradually come into existence in dark areas or areas in which the existing strokes begin to diverge too much. With controlled-density hatching, we are able to exert fine-grain control over the tone depicted in various areas of a pen-and-ink image. We use this newly acquired ability, together with traditional (image-based) texture mapping techniques, to extend considerably the repertoire of effects that can be achieved with stroke textures. We demonstrate these effects with texture maps used for surface detail, bump mapping, and environment ("reflection") mapping. In addition, we show how a planar map can be created and strokes efficiently clipped without the use of BSP trees. Finally, we describe a simple method to handle the casting of curved shadows onto curved objects.

1.1 Related work

A few authors have addressed similar goals in their published work.

Dooley and Cohen proposed a system to enhance a traditional shaded images with illustration techniques [3, 4]. They showed how line and surface qualities could be customized by the user to create more effective images.

Saito and Takahashi [18] used "G-buffers" and image processing techniques to enhance ray-traced images with illustration features. Their system handles outlining, hatching, and shadows. However, the use of strokes that we propose allows perhaps more expressiveness and extends the range of illustrations that can be created automatically.

Leister presented a technique to emulate copper-plate rendering [12], an engraving technique used for old styles of printing. A ray-tracing approach is used to render curves on free-form objects. These curves lie at the intersection of parallel planes with the 3D object being rendered. An advantage of this approach is that it easily handles reflections and shadowing.

The Piranesi system proposed by Lansdown and Schofield [11] also uses non-photorealistic techniques to create illustrations from 3D models. Piranesi uses a standard graphics pipeline to create a 2D reference image akin to a G-buffer. The user is then allowed to select specific regions of the image and apply textures that emulate natural media interactively or automatically.

Elber [5] described an algorithm to cover NURBS surfaces with isoparametric curves, thus emulating a form of line-art rendering. However, his approach does not address a number of the issues in pen-and-ink illustration considered in this paper, such as building tone with stroke textures, outlining objects only when necessary, and rendering shadows.

1.2 Overview

The rest of this paper is organized as follows. Section 2 gives a brief review of some of the key principles of pen-and-ink illustration, and summarizes the system architecture used for creating illustrations of polyhedral models. Section 3 describes the various algorithms

that lie at the heart of our system. Section 4 describes the particular stroke textures we used for the figures in this paper, and gives statistics for these results. Finally, Section 5 suggests some areas for future work.

2 Background

In this section, we briefly review some of the principles of pen-and-ink illustration; much more detailed studies can be found in a number of texts [9, 13, 17]. We then describe some of the key architectural features of the pen-and-ink illustration system we introduced in our previous work, upon which the results in this paper are based.

2.1 Principles of pen-and-ink illustration

Some of the key principles of pen-and-ink illustration include:

- *Strokes.* Strokes are the fundamental building-blocks of pen-and-ink illustration. The thickness and density of the strokes is varied to achieve subtle shading effects. In addition, strokes should also have some variation in thickness and waviness so as not to appear too "mechanical."
- *Texture.* Texture results from a large number of pen strokes placed in juxtaposition. The character of the strokes is important for conveying texture—for example, crisp, straight lines are good for "glass," whereas rough, sketchy lines are good for "old" materials.
- *Tone.* The perceived grey level or "tone" is a function of the density of the strokes in a particular region of the illustration. The same strokes that are used to convey texture must also be used to achieve the desired tone.
- *Outline.* Outlining play an essential role in illustration; indeed, outlining is one of the key features that differentiates illustration from photorealistic imagery. Outlines are generally introduced only where they are required to disambiguate regions of similar tone. The quality of the outline stroke must also be varied to convey texture.

2.2 System architecture for polyhedral models

The system for automatically producing pen-and-ink illustrations of polyhedral models, upon which the results in this paper are based, is not very different from a traditional photorealistic renderer. The input to the system consists of a 3D polyhedral model, one or more light sources, and a camera specification. The output is an illustration in the style of pen and ink.

To render a scene, the polyhedral renderer begins by computing the visible surfaces and the shadow polygons, using 3D BSP trees for both operations [1, 7]. The outcome is a set of convex polygons that can be ordered in depth with respect to the view point. The renderer uses these polygons to build both a 2D BSP tree and a planar map representations of the visible surfaces in the scene. It then renders each region in the planar map using a procedural stroke texture. The collection of strokes required to render each flat-shaded surface is generated without considering occlusions. Each stroke is then clipped against the visible portions of the surface using the 2D BSP tree. Finally, the outline strokes are drawn by considering all the edges of the planar map, and rendering only those edges necessary for the illustration, according to the outlining principles.

3 Algorithms

The pen-and-ink rendering system that we describe in this paper utilizes the same basic architecture as the polygonal renderer we introduced in our earlier work. It also uses the same procedural stroke texture idea, and relies on a planar map for generating outlines and clipping strokes. However, in the presence of free-form surfaces, many of the techniques used in the polygonal renderer no longer work. In this section, we present our solutions to these problems.

3.1 Controlled-density hatching

To produce pen-and-ink illustrations from parametric surfaces, the most fundamental change from a polyhedral renderer is in the generation of the stroke textures.

Curved surfaces require a much more sophisticated approach than flat-shaded polygonal surfaces, which can be hatched in a uniform fashion. First, we will need a way of orienting the hatching strokes along a surface. Second, we will need some mechanism for allowing strokes to gradually disappear in light areas of a surface or in areas where too many strokes converge together. Conversely, we will also need to allow new strokes to gradually appear in dark areas or areas in which the existing strokes begin to diverge too much. We will call such a mechanism *controlled-density hatching*.[1]

To solve the first problem, that of choosing an orientation for the hatching strokes, we simply use a grid of lines in the *parameter domain* (u, v). The grid consists of parallel lines running in one or more user-specified directions. For most illustrations, we simply use isoparametric curves, which run parallel to u and/or v.

We now turn to the second problem, that of achieving controlled-density hatching. Achieving a given tone by hatching an arbitrary parametric surface is a non-trivial problem. Figure 1 illustrates the difficulty, even for the case of a simple two-dimensional transformation. In this case, rendering isoparametric curves with constant thickness results in an image with varying tones. Our solution is to adjust the thickness of the strokes in order to keep the "apparent tone" constant. Figure 2 illustrates the same concept, but in this case, for a perspective view of a sphere.

In order to solve this problem formally, we begin by defining a *stroke* γ as a pair of functions $(\lambda(t),\ \theta(t))$, where $\lambda(t)$ is a line in the parameter domain (u, v), and $\theta(t)$ is a *thickness function*, which describes the *thickness* used in rendering the stroke at every parameter value t. Furthermore, we define the *apparent tone* of an image in the neighborhood of a given point in *image space* (x, y) to be the ratio of the amount of ink deposited in that neighborhood to its area. If the point (x, y) happens to lie on a stroke, the apparent tone can also be expressed as the ratio θ/d, where θ is the thickness of the stroke and d is its image-space separation from adjacent strokes.

With these definitions, the controlled-density hatching problem can be formally stated as follows.

Given:

- *A parametric surface* $\sigma : (u, v) \mapsto (x_w, y_w, z_w)$*, which maps points in the parameter domain* (u, v) *to points in* world space (x_w, y_w, z_w)*;*
- *a perspective viewing transformation* $\mathcal{V} : (x_w, y_w, z_w) \mapsto (x, y)$*, which maps (visible) points in world space to points in image space* (x, y)*;*
- *a hatching direction* $\boldsymbol{h} = (h_u, h_v)$ *in the parameter domain; and*
- *a target tone function* $T(x, y)$*.*

Find: *A set of strokes* $\gamma_i = (\lambda_i, \theta_i)$*, with lines* λ_i *in the parameter domain running parallel to the hatching direction* $\boldsymbol{h}$*, such that the apparent tone of mapping the strokes through* $\mathcal{M} = \mathcal{V} \circ \sigma$ *is* $T(x, y)$*.*

[1] For this work, we consider only surfaces with a global parameterization, such as B-spline surfaces, NURBS, and surfaces of revolution. Ideas for generalizing to a broader class of surfaces, such as patch-based surfaces and smoothly-shaded polygonal meshes, are discussed in Section 5.

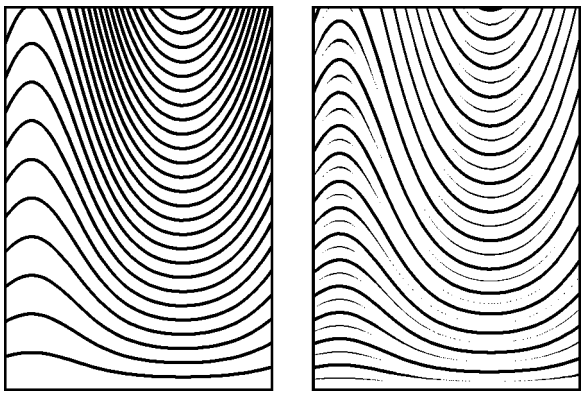

Figure 1 Controlled-density hatching for a simple 2-dimensional transformation $\mathcal{M} : (u, v) \mapsto (u,\ v + v \cdot \exp(\sin(u)))$. Rendering isoparametric curves with constant thickness results in an image with varying tones (left). We adjust the thickness of the strokes in order to keep the "apparent tone" constant (right).

The key step in solving this problem will be to determine exactly how the images of two parallel lines in the parameter domain converge and diverge when seen in image space.[2] In particular, let $\lambda_i(t)$ and $\lambda_{i+1}(s)$ be two parallel lines that are d units apart in the parameter domain, and let $\lambda_i'(t)$ and $\lambda_{i+1}'(s)$ be their images, under $\mathcal{M}$, in image space. We would like to know the distance d' between the two image-space curves as a function of t. Once we have this distance function $d'(t)$, we can use it to adjust the thickness and spacing of the strokes to compensate for any spreading or compression.

A simple closed-form expression for the distance between the two curves $\lambda_i'(t)$ and $\lambda_{i+1}'(s)$ does not exist in general, so we seek to approximate it. We begin by writing the mapping $\mathcal{M}$ as two scalar-valued functions

$$\mathcal{M}(u, v) \ \equiv \ (X(u, v), Y(u, v))$$

To approximate the behavior of $\mathcal{M}$ in a small neighborhood about (u, v) we consider the Jacobian matrix

$$J(u, v) = \begin{bmatrix} X_u & X_v \\ Y_u & Y_v \end{bmatrix}$$

where X_u, X_v and Y_u, Y_v are the partial derivatives of X and Y with respect to u and v respectively. The Jacobian matrix J is a linear transformation, which can be thought of as a Taylor expansion of $\mathcal{M}$ in the neighborhood of (u, v) truncated to the first-order terms. Under J, the two parallel lines λ_i and λ_{i+1} in parameter space map to parallel lines, denoted by $J(\lambda_i)$ and $J(\lambda_{i+1})$, in image space.

To estimate how much the curves λ_i' and λ_{i+1}' diverge, we look at the ratio ρ_i of the distance d_i' between the lines $J(\lambda_i)$ and $J(\lambda_{i+1})$ in image space, and the distance d_i between the lines λ_i and λ_{i+1} in the parameter domain. If the line λ_i is given by the implicit-form coefficients $\langle a, b, c\rangle$, then the ratio ρ_i is given by (see Appendix A):

$$\rho_i \ = \ \frac{d_i'}{d_i} \ = \ \sqrt{\frac{(X_u Y_v - X_v Y_u)^2\ (a^2 + b^2)}{(aY_v - bY_u)^2 + (bX_u - aX_v)^2}} \qquad (1)$$

The ratio $\rho_i(t)$ is a scalar-valued function that approximates how the distance between strokes is altered by the mapping $\mathcal{M}$. We call ρ_i the *stretching factor* of $\mathcal{M}$. When ρ_i is large, the lines spread apart; when ρ_i is small, they compress together. The *maximum stretching factor* $\overline{\rho}_i = \sup_t(\rho_i(t))$ taken along the line λ_i plays a special role: given two lines λ_i and λ_{i+1} offset by the distance d, it allows us to evaluate the maximum spacing $\overline{d}_i{}' = \overline{\rho}_i d$ between the two corresponding strokes.

[2] Note that the degree to which strokes in image space converge or diverge is dictated not only by the parametric surface, but also by the final projection to image space.

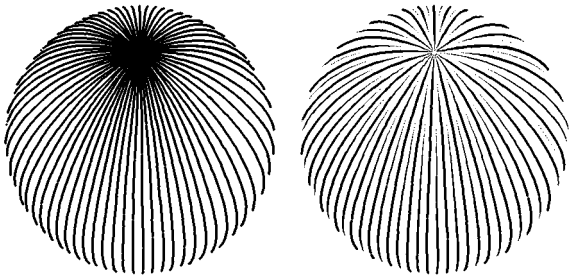

Figure 2 Controlled-density hatching for a perspective view of a sphere. Again, rendering isoparametric curves with constant thickness results in an image with varying tones (left). Using varying stroke thicknesses keeps the "apparent tone" constant (right).

We are now ready to generate strokes so as to achieve the target tone $T(x, y)$. We will do this in four steps.

First, we must decide what the maximum distance $\overline{d}'$ between two strokes in the image should be. This value is dictated by the maximum (or darkest) tone $\overline{T}$ that must be achieved anywhere on the surface, and the *maximum stroke thickness* $\overline{\theta}$ set by the user. To guarantee that $\overline{T}$ can be achieved, given $\overline{\theta}$, the strokes need to be spaced by no more than $\overline{d}' = \overline{\theta}/\overline{T}$ on the image plane.

Second, we note that because the stretching factor ρ_i is derived from a first-order approximation of $\mathcal{M}$, it is accurate only for very small steps in parameter space. In practice, however, the strokes must be spread apart by a comparatively large distance. To work around this problem, we use a stepping technique. To spread two strokes by a distance $\overline{d}'$, we take a series of small step of size δ in parameter space, updating the stretching factor after each step. Stepping starts from the line λ_i, and proceeds until the accumulated image-space distance, given by $\sum_j \overline{\rho}_j \delta$, equals or exceeds $\overline{d}'$. In our implementation, δ is set to $0.01\overline{d}'/\overline{\rho}_i$.

Third, we must modulate the thickness of the strokes to accurately render the tone $T(x, y)$. Two factors influence the thickness of the stroke $\lambda_i'(t)$: the actual image-space distance $d'(t)$ between the strokes, and the tone to be achieved. The stepping algorithm devised in the first step guarantees that two adjacent strokes are spread by at most $\overline{d}'$. However, the actual spacing $d'(t)$ can be smaller. To compensate for this variation, the thickness of stroke λ_i' must be scaled by the ratio $d'(t)/\overline{d}' \ \approx \ \rho_i(t)/\overline{\rho}_i$. Finally, to take into account the varying tone, we also scale the stroke thickness by the ratio $T(t)/\overline{T}$. In summary, the thickness of $\lambda_i'(t)$ is given by

$$\theta_i(t) = \frac{T(t)}{\overline{T}} \frac{\rho_i(t)}{\overline{\rho}_i}\, \overline{\theta}$$

Finally, we introduce an additional feature to create more interesting hatching. Although the strokes generated by the method above accurately render the target tone, the thicknesses of all the strokes vary simultaneously. A more appealing effect is achieved when short and long strokes are interspersed, as depicted on the right side of Figure 1. This effect is created by introducing an additional *spreading*

factor σ, set by the user. The initial strokes are spread by the distance $\sigma\overline{d}'$ instead of $\overline{d}'$. The extra gaps created are then recursively filled with additional *filler* strokes.

The expression for the thickness $\theta_i(t)$ of a filler stroke γ_i at level ℓ of recursion is slightly more complicated. To derive it, we first note that the image-space distance between two adjacent strokes at level ℓ of recursion is given by

$$d'_\ell(t) \;=\; \frac{\sigma\,\overline{d}}{2^\ell}\frac{\rho_i(t)}{\overline{\rho}_i} \;=\; \frac{\sigma\overline{\theta}}{2^\ell\overline{T}}\frac{\rho_i(t)}{\overline{\rho}_i} \qquad (2)$$

With this style of recursive hatching, we would like to achieve the target tone by using the thickest possible strokes, before introducing a filler stroke at level ℓ. Consequently, if $\theta_i(t) > 0$ for some t, then the thickness of the neighboring strokes at recursion level $\ell - 1$ is $\overline{\theta}$. The contribution to the tone from these strokes is $T_{\ell-1} = \overline{\theta}/d'_{\ell-1}$, while stroke γ_i contributes $T_\ell = \theta_i(t)/d'_\ell$. Finally, the overall target tone to achieve is $T(t) = T_{\ell-1} + T_\ell = \overline{\theta}/d'_{\ell-1} + \theta/d_\ell$. Substituting for $d'_{\ell-1}$ and d'_ℓ using equation (2), and noting that $\theta_i(t)$ cannot exceed $\overline{\theta}$, yields

$$\theta_i(t) = \min\left\{\overline{\theta},\, \left(\frac{\sigma T(t)}{2^\ell\overline{T}}\,\frac{\rho_i(t)}{\overline{\rho}_i} - \frac{1}{2}\right)\,\overline{\theta}\right\}$$

The recursion stops when $\theta_i(t) \leq 0$ everywhere along the stroke. The minimum thickness of a stroke is dictated either by the pixel size, or by a constant set by the user. However, "visually thinner" sizes can still be obtained by using dashed strokes.

The hatching textures of all the figures shown in this paper were generated using this recursive algorithm. Typically, σ ranges between 2 and 8.

3.2 The planar map

As discussed in Section 1.2, a key data structure of the pen-and-ink illustration system for polyhedral models was a planar map of all the visible surfaces and shadow polygons. This planar map was constructed with the help of 2D and 3D BSP trees [22]. Recent results introduced by Naylor and Rogers [15] show how to build 2D BSP trees with Bézier curves. However, it is not clear how this work can be generalized to handle scenes containing parametrically defined curved surfaces. It is also not clear how Chin and Feiner's BSP-tree-based shadow algorithm can be generalized in the presence of curved surfaces. For these reasons, we devised a method for computing the planar map and the shadows that does not rely on BSP trees.

3.2.1 Constructing the planar map

The planar map data structure partitions the image plane into homogeneous regions so that each region corresponds to a single visible object in the scene. In our new algorithm, the planar map is constructed in three main phases.

In the first phase, we tessellate every object in the scene into a polygonal mesh. The resolution of the tessellation is chosen so as to yield a reasonably-accurate approximation to the object. Our implementation uses a fixed resolution set by the user, although a flatness criterion could also be used.

In the second phase, we compute higher-resolution piecewise-linear approximations for all the silhouette curves of the meshed objects. This step is required to obtain smooth and accurate silhouettes without requiring an unduly fine tessellation. Our technique is very similar to the one developed for hidden-curve removal by Elber and Cohen [6], only it operates on a polygonal mesh, rather than on a parametric surface directly. To find the silhouette curves, we first identify the mesh edges that span the silhouette. To do this, we examine the normal vectors at the two endpoints of every edge. If the projections of the normals on the viewing direction are in opposite directions, then the edge spans the silhouette. In this case, the two mesh faces adjoining the edge are subdivided, and the process is repeated. Our implementation performs a fixed number of subdivisions in this manner. Finally, the silhouette curve is further refined, using a root-finding method to evaluate a more precise silhouette point along each remaining mesh edge that spans the silhouette. A piecewise-linear silhouette curve is then constructed by connecting all of the silhouette points in the mesh.

In the third and final phase, we construct the planar map itself. Initially, the planar map consists of a single region corresponding to the entire display area. Each mesh face is then inserted into the planar map in turn. First, the face is projected onto the view plane, and its edges are merged with those of the planar map. Next, we resolve occlusions between the new face and the existing faces in the planar map that are covered by the new face. Our implementation currently assumes non-intersecting objects; thus, for each existing face in the planar map, we merely need to determine whether the existing face or the face being inserted is closer to the viewer. We do this by summing up the distance from the view point to the 3D point corresponding to each edge midpoint. Whichever face yields the smallest sum is considered to be closer to the viewer. If the face being inserted is closer, the planar map region is updated to reflect the new information.

Once all objects have been inserted into the planar map, each resulting region corresponds to a single visible 3D face in the mesh decomposition. In our implementation, we maintain a link from each region to its 3D face. In turn, each 3D mesh maintains a link to its original object. These links are used by the procedural stroke textures to compute a variety of information, as described in Section 3.3.

3.2.2 Robustness issues

A common problem in geometric algorithms, and one to which our planar map construction algorithm is certainly not immune, is that it is not always easy to maintain consistency between the topological and geometric information in the data structure when imprecise computations like floating-point arithmetic are used [10, 19]. To build the planar map robustly, we use a method inspired by the work of Gangnet *et al.* [8]. Notably, we restrict all the line endpoints to

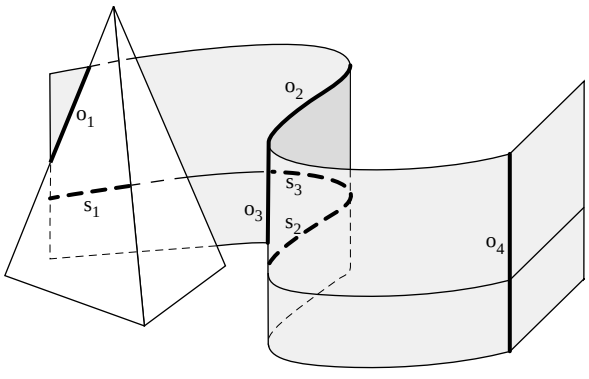

Figure 3 Several cases must be considered when tracing outlines (edges labeled o_1 to o_4), and clipping strokes (edges labeled s_1 to s_3).

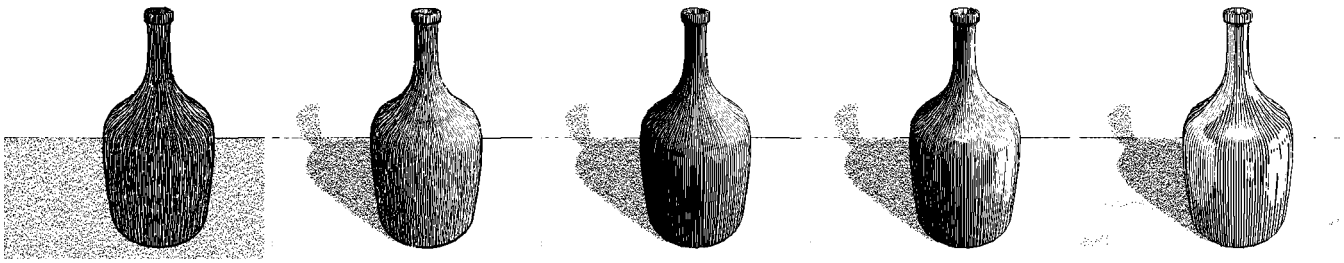

Figure 4 Creating a pen-and-ink illustration. The steps involved are not so different from those required to create an attractive photorealistic rendering. From left to right: constant-density hatching; smooth shading with rough strokes, using a single light source; smooth shading with straighter, longer strokes adjusted to depict glass; introducing environment mapping; and, finally, the same image after adjusting the reflection coefficients, shown at full size in Figure 5.

lie on an integer lattice, and we use infinite-precision rational arithmetic to compute all intersections exactly. Because the planar map stores only line segments, the number of bits required for the intermediate computations is bounded. In particular, we use 14-bit integers to represent the lattice points, which allows all intersections to be stored using 32-bit rational integer numbers. This choice limits us to a resolution of about 800 dots per inch over a 10×10-inch image-space area. (See Winkenbach [21] for more details.)

3.3 Using the planar map

As in the original polyhedral renderer, the planar map is used for rendering outline edges. In this work, the planar map is additionally used for clipping individual strokes to visible regions. Here we consider how these two processes can be implemented for curved surfaces.

3.3.1 Outlining

Object outlines are constructed from edges of the planar map. With curved surfaces, four types of planar map edges can give rise to an outline edge (see Figure 3):

- Case o_1: an edge that bounds two regions belonging to different objects. The texture for such an outline edge is taken from the object closest to the view point. In the case of two abutting surfaces, this choice is arbitrary.
- Case o_2: an edge that bounds two regions belonging to the same object, but whose corresponding 3D mesh faces have opposite orientations.
- Case o_3: an edge that bounds two regions belonging to the same object and having the same orientation, but at different depths.
- Case o_4: an edge that arises from a C^1 discontinuity on the surface.

An outline path is assembled by appending as many adjacent outline edges as possible. The darkness of the stroke along an outline edge is affected by two factors:

- *The tone value on the surface* — letting an outline edge fade away in regions of highlight reinforces the quality of the shading.
- *The contrast between two adjoining surfaces* — if the tone difference between the two adjacent surfaces is small, a darker outline is required to mark the boundary.

The degree to which these criteria affect the outline is selectable by the user.

3.3.2 Generating stroke paths

Both outline and texture strokes are initially constructed as 3D polyline curves, called *stroke paths*. Each stroke path vertex stores several items of information, including the parametric coordinate, the corresponding 3D position, and the tone evaluated on the surface at that point. The number of vertices in the path is adjusted using a subdivision algorithm, with the goal of accurately capturing not only the shape of the stroke, but also any variation in tone. The latter is particularly important when texture, bump, or environment maps with small features are present.

3.3.3 Clipping stroke paths

Using the planar map, we generate strokes for all surfaces that are at least partially visible. However, these strokes can potentially extend into invisible regions. To clip the strokes in the presence of curved surfaces, we break each stroke at the silhouette points, yielding segments that either face toward or away from the view point. Each of these segments is then projected onto the planar map and clipped. (If the object being rendered is a solid, then back-facing stroke segments can be rejected immediately.) The clipping process starts by locating the planar map region in which the stroke's first vertex lies. The visibility of the stroke is then tested for every planar map edge that the stroke crosses. When the visibility changes, a root-finding method is used to find the intersection between the path and the planar map edge. The intersection point is then added to the path, breaking it into smaller segments.

Several tests are used to determine the visibility of a stroke segment within a face of the planar map. They are, in order of application (see Figure 3):

1. *Same object* — the planar map face must be linked to the same 3D object that the stroke is covering (eliminates s_1).
2. *Same orientation* — the planar map face must be linked to a 3D mesh face that has the same orientation (back-facing or front-facing) with respect to the view point as the 3D surface point along the stroke path (eliminates s_2).
3. *Same depth* — the 3D location of the path vertex must be close to the corresponding 3D location of the 3D mesh face. This last test is required since, with free-form surfaces, several different points on the same surface can project to the same 2D point (eliminates s_3).

If all three tests succeed, the segment is visible and marked as such. Only the visible segments of each stroke are drawn.

Figure 5 Glass bottle. An environment map is used to give the illusion of a reflected surrounding.

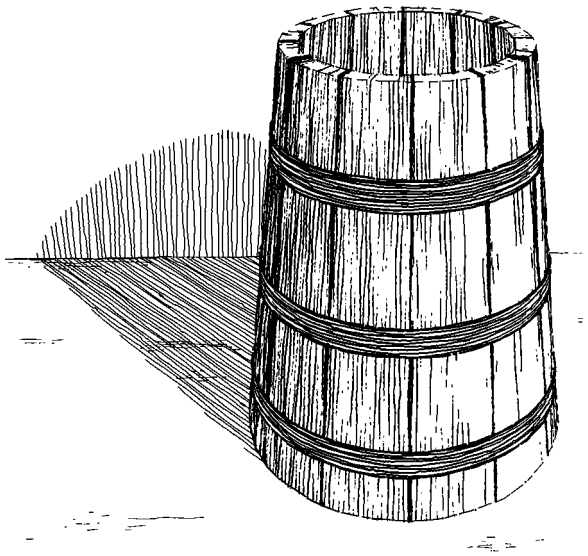

Figure 6 Wooden bucket. The bucket is modeled as a single surface of revolution. The planks are created by a prioritized stroke texture.

3.4 Shadows

In the polygonal version of the renderer, polygons were split along shadow boundaries before being inserted into the 2D BSP tree. Hence, the different partitions in the BSP-tree would distinguish between regions that were in and out of shadow. The shadows were then rendered with strokes clipped to the shadow regions. Unfortunately, with curved surfaces, shadow boundaries are much more difficult to generate. Thus, we decided to use a simpler two-pass clipping approach inspired by Williams [20] instead.

Shadow strokes are generated for all the visible surfaces. To clip these strokes, we build an additional *shadow planar map* with respect to the light source, in addition to the *view planar map*. Each shadow stroke is first clipped against the view planar map, just like all other strokes. In a second pass, the remaining visible shadow strokes are clipped against the shadow planar map; this time, however, just the portions of the strokes that are *not* visible from the light source (and therefore in shadow) are preserved and rendered.

Note that the view planar map and the shadow planar map lie in distinct 2D spaces. Therefore, each visible stroke left after the first clipping pass must be re-mapped to the shadow planar map space before the second clipping pass can take place.

4 Results

In this section, we demonstrate the rendering algorithms with several examples. The rendering times for these figures can be found in Table 1.

All of the examples in this paper were created using an iterative design process, not unlike the procedure typically used in creating an attractive photorealistic rendering. First, we generally set up one or more light sources, then adjust the quality of the strokes, then add any texture maps, and finally adjust the various reflection-model parameters until an appealing illustration is achieved. Figure 4 illustrates this process for the glass bottle, shown at full size in Figure 5.

4.1 Texture mapping

Controlled-density hatching allows "fine grain" control over the tone of a pen-and-ink illustration. With this new capability, we can use traditional texture mapping techniques to vary the tone on the surface of an object. For example, Figure 5 uses an environment map to enhance the illusion of the glass material. Figure 6 uses a bump map to perturb the shading on the wooden planks. Figure 7 uses an ordinary texture map to create the geometric pattern on the bowl; it also uses a bump map to emboss the word "MILK" and create a slightly irregular surface on the jug.

4.2 Other texture styles

The basic hatching algorithm described in this paper can be used to generate many other texture styles:

- *Wood*. The wood texture shown in Figure 6 uses a variety of strokes, much like the prioritized stroke texture for wood used in the polygonal version of the renderer. Thin wavy strokes are used to convey wood grain, while longer strokes of varying thickness delineate the gaps between the wood boards.
- *Stippling*. Figures 7 and 8 show the use of stippling to build tone values. To create the stipples, we generate hatching strokes as described previously. However, the resulting stroke paths are not rendered directly; instead, they serve as curves along which the stipples are drawn. The spacing between the stipple marks along the stroke path is randomized. In addition, the stipple marks are also offset from the path by a small random distance.
- *Crosshatching*. Figure 7 also shows the use of crosshatching — using more than one hatching direction — to create dark shadow tones. Crosshatching is also used on the cane in Figure 8 .

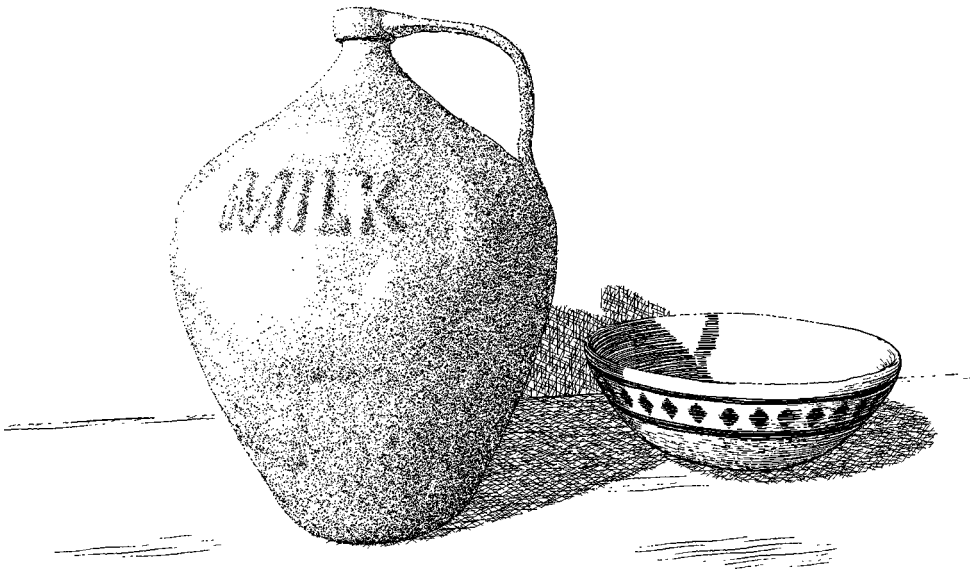

Figure 7 Ceramic jug and bowl. A traditional (image-based) texture map is used to model the details on the bowl as well as the stains on the table. A bump map is used to emboss the word "MILK" on the jug, and to give some irregular variation to its surface.

Fig	*Model*	*Planar map*	*Rendering*
5	*Glass bottle*	*74*	*46*
6	*Wooden bucket*	*21*	*44*
7	*Jug and bowl*	*126*	*128*
8	*Hat and cane*	*230*	*120*

Table 1 Rendering times for various illustrations presented in this paper. All times are in seconds, and were measured on a Power Mac 7100/80.

5 Conclusion and future work

In this paper, we have introduced the concept of controlled-density hatching, which allows strokes to be generated so as to simultaneously convey tone, texture, and shape for parametric surfaces. Because controlled-density hatching provides "fine grain" control of the tone of an illustration, we were also able to use traditional texture mapping techniques to extend the range of effects that can be achieved with pen-and-ink rendering. We have also described an algorithm to construct a planar map from parametric surfaces, and we have shown how this planar map can be used for outlining and stroke clipping, in addition to resolving occlusions. Finally, we have described a simple method to render shadows with strokes.

Perhaps the biggest limitation of this work is that it deals only with surfaces possessing a global parameterization. Unfortunately, many commonly used surface representations, such as patch-based surfaces, implicit surfaces, subdivision surfaces, and smoothly-shaded polygonal meshes, do not have this property. One possible solution is to parameterize such surfaces, using for example the methods of Maillot *et al*. [14] or Pedersen [16]. Another alternative is to do away with the parameterization altogether, and to instead generate strokes along directions that are more intrinsic to the geometry of the surface — for example, along directions of principal curvature [2]. This approach may also be suitable for mapping stroke textures on polygonal meshes, since surface curvature can still be approximated in this case [14].

Acknowledgments

We would like to thank Jorge Stolfi for many useful discussions during the early phase of this project.

This work was supported by an Alfred P. Sloan Research Fellowship (BR-3495), an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), and industrial gifts from Interval, Microsoft, and Xerox.

References

[1] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. *Computer Graphics*, 23(3):99–106, 1989.

[2] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[3] Debra Dooley and Michael F. Cohen. Automatic illustration of 3D geometric models: Lines. *Computer Graphics*, 24(2):77–82, March 1990.

[4] Debra Dooley and Michael F. Cohen. Automatic illustration of 3D geometric models: Surfaces. In *Proceedings of Visualization '90*, pages 307–314, October 1990.

[5] Gershon Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transaction on Visualization and Computer Graphics*, 1(3):231–239, September 1995.

[6] Gershon Elber and Elaine Cohen. Hidden curve removal for free form surfaces. *Computer Graphics*, 24(4):95–104, August 1990.

[7] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, July 1980.

[8] Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. Incremental computation of planar maps. *Computer Graphics*, 23(3):345–354, July 1989.

[9] Arthur Leighton Guptill. *Rendering in Pen and Ink*. Watson-Guptill Publications, New York, 1976.

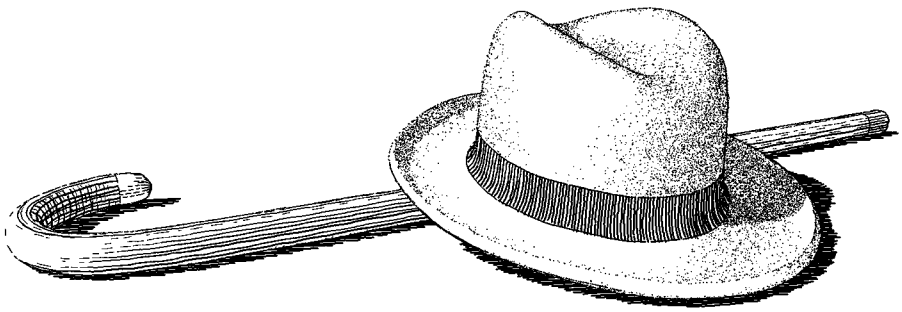

Figure 8 Hat and cane. Both the hat and the cane are modeled with B-spline surfaces. The ribbon is modeled as a separate B-spline surface. Note the curved shadow that the hat projects on its rim, and the use of crosshatching on the curved portion of the cane.

[10] Christoph Hoffman. The problems of accuracy and robustness in geometric computation. *Computer*, 22:31–42, 1989.

[11] John Lansdown and Simon Schofield. Expressive rendering: A review of nonphotorealistic techniques. *IEEE Computer Graphics and Applications*, 15(3):29–37, May 1995.

[12] Wolfgang Leister. Computer generated copper plates. *Computer Graphics Forum*, 13(1):69–77, 1994.

[13] Frank Lohan. *Pen and Ink Techniques*. Contemporary Books, Inc., Chicago, 1978.

[14] Jérôme Maillot, Hussein Yahia, and Anne Verrout. Interactive texture mapping. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics*, Annual Conference Series, 1993.

[15] Bruce Naylor and Lois Rogers. Constructing partitioning trees from Bézier-curves for efficient intersection and visibility. In *Proceedings of Graphics Interface '95*, pages 44–55, 1995.

[16] Hans Køhling Pedersen. Decorating implicit surfaces. Proceedings of SIGGRAPH 95 (Los Angeles, California, July 6-11, 1995). In *Computer Graphics*, Annual Conference Series, 1995.

[17] Tom Porter and Sue Goodman. *Manual of Graphic Techniques 4*. Charles Scribner's Sons, New York, 1985.

[18] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3D shapes. *Computer Graphics*, 24(4):197–206, August 1990.

[19] David H. Salesin. *Epsilon Geometry: Building Robust Algorithms from Imprecise Computations*. PhD thesis, Stanford University, March 1991. Available as Stanford Report number STAN-CS-91-1398.

[20] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3):270–274, August 1978.

[21] Georges Winkenbach. *Computer-Generated Pen-and-Ink Illustration*. PhD thesis, University of Washington, May 1996.

[22] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics*, Annual Conference Series, 1994.

A Deriving the stretching factor

To derive the expression for the stretching factor ρ_i, given in equation (1), we first note that the linear transformation J maps points (u, v) in parameter space to points (x, y) in image space by

$$J \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3}$$

Next, we write the implicit equations for line λ_i and its image $J(\lambda_i)$, using the implicit-form coefficients $\langle a, b, c\rangle$ for λ_i, and $\langle a', b', c'\rangle$ for $J(\lambda_i)$:

$$[a \;\; b] \begin{bmatrix} u \\ v \end{bmatrix} + c = [a' \;\; b'] \begin{bmatrix} x \\ y \end{bmatrix} + c' = 0 \tag{4}$$

Combining equations (3) and (4), we readily establish that

$$\begin{aligned} [a' \;\; b'] &= [a \;\; b] J^{-1} \\ c' &= c \end{aligned} \tag{5}$$

The distance d between two parallel lines with implicit-form coefficients $\langle a, b, c\rangle$ and $\langle a, b, c + \epsilon\rangle$ is $d = \epsilon/\sqrt{a^2 + b^2}$. The stretching factor ρ_i is given by the inverse ratio of the distance d_i between the lines λ_i and λ_{i+1}, and the distance d_i' between their images $J(\lambda_i)$ and $J(\lambda_{i+1})$:

$$\begin{aligned} \rho_i &= \frac{d_i'}{d_i} = \frac{\epsilon/\sqrt{(a')^2 + (b')^2}}{\epsilon/\sqrt{a^2 + b^2}} \\ &= \sqrt{\frac{(X_u Y_v - X_v Y_u)^2 \, (a^2 + b^2)}{(aY_v - bY_u)^2 + (bX_u - aX_v)^2}} \end{aligned}$$