# Orientable Textures for Image-Based Pen-and-Ink Illustration

*Michael P. Salisbury*   *Michael T. Wong*   *John F. Hughes**   *David H. Salesin*

University of Washington   *GVSTC

## Abstract

We present an interactive system for creating pen-and-ink-style line drawings from greyscale images in which the strokes of the rendered illustration follow the features of the original image. The user, via new interaction techniques for editing a direction field, specifies an orientation for each region of the image; the computer draws oriented strokes, based on a user-specified set of example strokes, that achieve the same tone as the image via a new algorithm that compares an adaptively-blurred version of the current illustration to the target tone image. By aligning the direction field with surface orientations of the objects in the image, the user can create textures that appear attached to those objects instead of merely conveying their darkness. The result is a more compelling pen-and-ink illustration than was previously possible from 2D reference imagery.

**CR Categories and Subject Descriptors:** I.3.3 [Computer Graphics]: Picture/Image Generation — Display algorithms. I.4.3 [Image Processing] Enhancement — Filtering

**Additional Key Words:** Controlled-density hatching, direction field, image-based rendering, non-photorealistic rendering, scale-dependent rendering, stroke textures.

## 1 Introduction

Illustrations offer many advantages over photorealism, including their ability to abstract away detail, clarify shapes, and focus attention. In recent years, a number of systems have been built to produce illustrations in a pen-and-ink style. These systems can be classified into two broad categories, depending on their input: *geometry-based systems* [1, 2, 7, 12, 16, 17, 18], which take 3D scene descriptions as input; and *image-based systems* [10, 13], which produce their illustrations directly from greyscale images. The main advantage of geometry-based systems is that—because they have full access to the 3D geometry and viewing information—they can produce illustrations whose strokes not only convey the tone and texture of the surfaces in the scene, but—by placing strokes along the natural contours of surfaces—they can also convey the 3D forms of the surfaces. Existing image-based systems, on the other hand, have no knowledge of the underlying geometry or viewing transformations behind the images they are rendering, and until now have been able to convey 3D information only by having a user draw individual strokes or specify directions for orienting particular collections of strokes across the image.

University of Washington, Box 352350, Seattle, WA 98195-2350
{ salisbur | mtwong | salesin }@cs.washington.edu
*NSF STC for Computer Graphics and Scientific Visualization, Brown University Site, PO Box 1910, Providence, RI 02912
jfh@cs.brown.edu

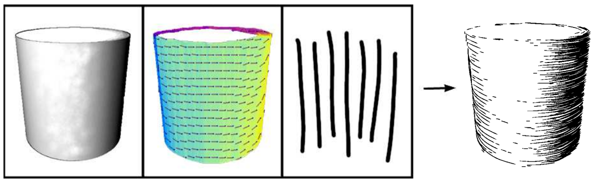


**Figure 1** The three components of a layer are from left to right tone, direction, and a stroke example set. An illustration (far right) is rendered based upon one or more such layers.

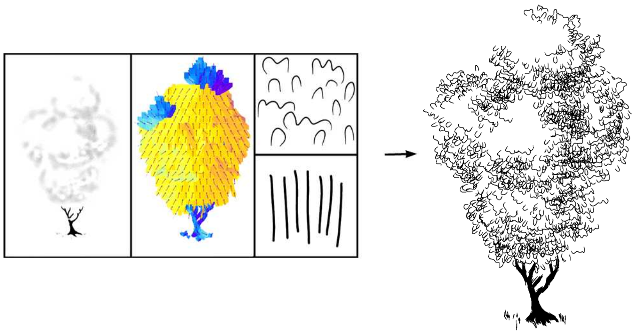


**Figure 2** A tree with curved strokes for leaves and straight strokes for branches and trunk.

In this paper, we introduce the notion of "orientable textures" and show how they can be used to readily convey 3D information in an image-based system for pen-and-ink illustration. In our interactive system, a user creates an illustration from a reference image by specifying three components: a greyscale *target image* that defines the desired tone at every point in the illustration, a *direction field* that defines the desired orientation of texture at every point, and a *stroke example set*, or set of strokes, to fill in the tone areas (see figures 1 and 2). Given these three components and a scale for the final illustration, the system creates an *orientable texture*—generated procedurally—that conveys the tone, texture, and forms of the surfaces in the scene. An illustration is composed of one or more such layers of orientable textures, allowing an illustration to be rendered with several, potentially overlapping, types of strokes.

The ability to generate comparable illustrations with an image-based system rather than a geometry-based system offers several advantages. First, using an image-based system greatly reduces the tasks of geometric modeling and of specifying surface reflectance properties, allowing much more complicated models (such as furry creatures and human faces) to be illustrated. Second, an image-based system provides the flexibility of using *any* type of physical photograph, computer-generated image, or arbitrary scalar, vector, or tensor field as input, allowing visualization of data that is not necessarily even physical in nature. Finally, image-based systems offer more direct user control: the ability to much more easily modify tone, texture, or stroke orientation with an interactive digital-paint-style interface.

Although this paper is, to our knowledge, the first to use orientable textures for image-based pen-and-ink illustration (in which the strokes must convey not only orientation, but texture and tone), the idea of orienting strokes for illustration dates back at least as far as the seminal papers by Saito and Takahashi [11] and Haeberli [6] in SIGGRAPH 90. Winkenbach and Salesin [17] and Meier [9] also make use of oriented strokes for geometry-based illustration.

Supporting orientable textures for image-based pen-and-ink illustration requires solutions to several new subproblems, which we discuss in this paper. These problems include: creating interactive techniques that facilitate the specification of the kind of piecewise-continuous vector fields required for illustration; rendering strokes and stroke textures according to a vector field in such a way that they also produce the proper texture and tone; and efficiently estimating tone as new oriented strokes are progressively applied.

The next section describes the user interface for specifying the components of an illustration. Section 3 discusses the rendering of illustrations with oriented textures. Section 4 presents our results.

## 2 The interactive system

We provide an editor, similar to a conventional paint program, that allows the user to interactively alter the tone and direction components of a layer.[1] The user can view and edit arbitrary portions of a component at varying levels of zoom, superimpose multiple components, and paint directions directly on top of the target image. For an example of the high-level control afforded by our system, refer to figure 3.

***Editing tone.*** Our tone editor is similar to existing paint programs. It supports lightening, darkening, and other image-processing operations, as well as painting. The user can load a reference image and designate it as a "cloning source." Selected portions of this reference may then be painted into a given layer's tone component. Tone may also be transferred between layers by painting. A negative cloning brush allows the user to freely and creatively reverse tonal relationships in a reference image.

***Editing direction.*** Since we represent a direction field as a grid of direction values, much like an image of pixels, the direction-field editor is similar to the tone editor.[2]

The user "paints" directions on the image with a collection of tools, a few of which we describe here. The basic tool is the *comb*, which changes the directions of pixels beneath the cursor to match the direction of motion of the cursor. If a user wishes to smooth out discontinuities in the direction field, there is a *blending tool* that smooths a region of directions by convolving each point under the brush with a $3 \times 3$ filter.[3] There are also various region-filling tools. One tool lets the user fill a region with a constant direction. Another provides *interpolated fill*: the user draws two curves, after which the region between them is filled with directions that are tangents of linear interpolants of the curves. A third provides *source fill*, which orients directions away from a selected point.

The current state of the direction field is shown in two ways: first, a grid of line segment indicators covers the image and everywhere points in the direction of the field; second, a color-coded direction image is superimposed on the tone image

***Applying the stroke example set.*** A *stroke* is a mark to be placed on the page. Each stroke is *oriented*, in the sense that it can be rotated to any angle to follow the direction field where it is placed. The *stroke example set* is a collection of strokes, all drawn with respect to the vertical orientation, that serve as prototypes for the strokes in the final image. Each such stroke is represented as a cubic

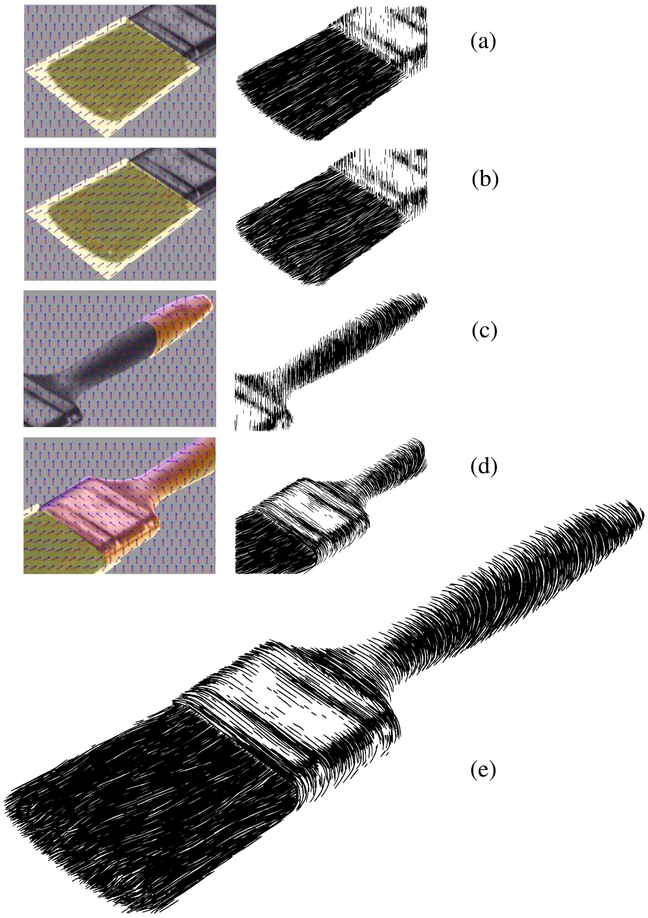



**Figure 3** The steps in specifying the direction field for a paintbrush illustration. Shown in inset at various stages during the development of the illustration are, on the left, the user interface, and on the right, the corresponding rendered illustration. By default, the direction field is oriented downward. In (a) we see the effect of an interpolated fill between two lines on either side of the brush bristles. Panel (b) shows the state of the direction field and illustration after some irregularities were introduced to the bristles by nine coarse strokes of the direction comb along the length of the bristles, and thirty fine strokes at the bristle tips. Panel (c) shows the state of the brush handle after interpolating fills between four curves drawn to reflect its surface orientation. In (d), the last section of the direction field covering the metal ferrule has been defined with three interpolating fills. Panel (e) shows the completed brush illustration.

B-spline with knot sequence $(0, 0, 0, 1, 2, \ldots, n-1, n, n, n)$, making it endpoint-interpolating. Thus a stroke example set for "parallel hatching" would contain many nearly vertical line segments, as shown in the third panel of figure 1, while for the leaves in figure 2, the strokes are wavy to suggest the edges of masses of foliage. When a stroke is drawn at a point in the illustration, it is rotated so that the vertical vector in the stroke texture aligns with the direction vector at that point; it is further warped so that this relation is true all along the stroke (see Section 3.1).

The repeated use of strokes from the example set to achieve tone with a specified orientation is a kind of procedural stroke texture. Non-procedural stroke textures were used by Salisbury *et al.* [13, 14]. In this previous work, the textures tiled the plane, and the stroke selected for drawing at a point was the one that happened to pass through that point. By contrast, in this new system the placement of strokes on the final illustration is independent of their relative position in the texture. Spacing between strokes is instead maintained indirectly by the rendering system (see Section 3). Dynamic placement of strokes is an important feature, for if we have

[1] The stroke example set is created in a separate program and can be loaded by name.

[2] We represent directions as values from 0 to 255, with 0 down, 128 up, and values increasing counter-clockwise. The resolution of the direction grid is the same as that of the tone image.

[3] We filter directions by first converting them into unit vectors, then performing a weighted sum of those vectors with the weights $(1, 2, 1; 2, 4, 2; 1, 2, 1)$, and then converting the resulting vector back into a direction.

**Figure 4** Magnifying a low-resolution direction field using (left) a standard symmetric resampling kernel, and (right) the modified kernel used by Salisbury *et al.* [14]. The same sharp tone component was used for both illustrations.

a direction field that diverges (say, for drawing the water spraying outwards from a fountain) and a stroke texture of parallel straight-line strokes that we wish to have follow the diverging field, a simple plane-tiling will not follow the field, and an embedding of the stroke texture that *does* follow the field will be stretched at the divergent end, necessarily causing the strokes to become more sparse. By contrast, our new method will insert additional strokes as the field widens, thus maintaining the density. In trade for this, we lose the texture-wide coherence that was available in our previous work.

## 3 Rendering

Once the user has specified the three components of a layer (tone, direction, and texture) our pen-and-ink renderer combines all of the components of each layer to generate the pen strokes of the final illustration. The user need only be concerned with the overall high-level aspects of the illustration such as tone and stroke direction; the system does the tedious work of placing all the strokes. Besides providing easy control over essential elements of an illustration, this separation of components until rendering allows us to produce illustrations at any size by first rescaling the components and then rendering, as described by Salisbury *et al.* [14]. Figure 4 demonstrates magnification of the direction field that respects edge discontinuities.

The rendering process is driven by a notion of "importance." We define the *importance* of a point as the fraction of its intended darkness that has not yet been accumulated at that point. By drawing in order of importance, we make all areas approach their target darkness at the same rate. Rendering therefore consists, roughly, of looking for the location with greatest importance, placing a stroke there, updating an image that records the importance, and repeating, until the importance everywhere is below a termination threshold. Each step of the process has subtleties, which are discussed below.

***Matching the illustration to the target.*** We aim to place strokes in the illustration so that the tone of the illustration "matches" that of the tone image. Matching is necessarily approximate, because the illustration is purely black and white, whereas the tone image is greyscale. To facilitate this approximate matching, we think of each stroke as adding darkness to a *region* of the illustration. Moreover, since strokes in dark areas will be closely spaced and those in light areas will be sparse, the size of each region must be inversely proportional to the darkness. One way of spreading the darkness of a stroke over a region is to blur the image of the stroke when considering the effect of its darkness. To measure the progress of our illustration towards the target image, we therefore compare a blurred version of the illustration with the tone image, where the blurring consists of applying averaging filters of variable size across the illustration, with the size increasing with the target lightness in a region. The diameter of the blurring filter is the same as the average inter-stroke distance required to achieve the target lightness.

We record our success at matching the illustration to the tone image by maintaining a *difference image*, updated after each stroke is drawn, whose value at each pixel is the difference between the tone image and a blurred version of the illustration. The *importance image* is derived from the difference image; its value at each point is

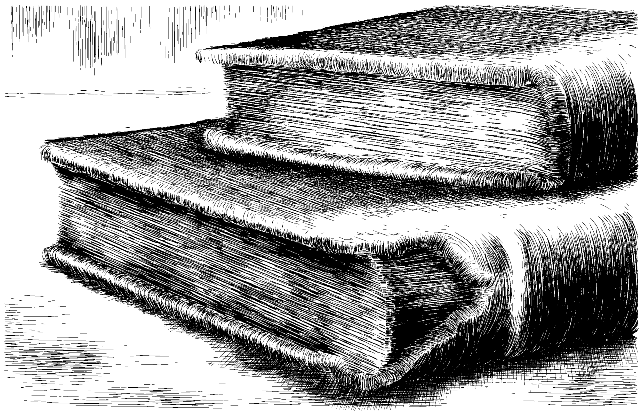


**Figure 5** Stacked books (after illustration by Frank Lohan [8].)

the current difference divided by the initial value of the difference.[4]

***Drawing strokes in the right place.*** One of the basic rules of pen-and-ink illustration is that strokes should be placed evenly: close together in dark areas, widely spaced in light areas [8]. In the computation of the difference image, the importance-image values at points within some distance of a stroke are lowered when the stroke is drawn, with points near the stroke being lowered most; the size of the region affected is determined by the target tone (see Section 3.2). This algorithm tends to maintain stroke separation.

To help determine where to draw the next stroke, i.e., the location with greatest importance, we maintain a quadtree on the importance image, updated locally whenever a stroke is drawn.

***Deciding when to stop.*** We do not actually try to drive the importance image to zero: even our filtered version of the strokes cannot hope to match the values in the tone image exactly. Instead, we try to drive the importance image to within a narrow tolerance around zero.[5] When the maximum value in the importance image is below a termination threshold, the renderer declares the illustration complete and stops drawing strokes.

### 3.1 Drawing a Stroke

The lowest-level activity is the actual drawing of a stroke, in itself a complex task. Once the algorithm knows where to place it, the stroke must be oriented, bent, and drawn. It must also be clipped if extending it further would make the illustration too dark. We discuss these processes in turn.

***Orienting and bending.*** To start, the algorithm randomly selects a prototype stroke from the stroke example set. We would like to map this stroke into the direction field so that, at every point along its length, the stroke's new angle relative to the direction field is the same as the prototype stroke's angle with respect to the vertical direction. Since this mapped stroke is not easy to find, we approximate it by mapping the control hull of the prototype stroke into the direction field in an angle-preserving way, as described below. This process produces a mapped stroke that is close to our ideal stroke and is easy to compute, although it is the *control hull* of the stroke that passes through the target point rather than the stroke itself. The errors thus introduced are small as long as the control hull fits the stroke closely and the direction field does not change too fast.

To map the control hull into the direction field, we first pin a random control point $P_i$ of the stroke onto the target location $X$ in the

[4] If the initial difference is zero (i.e., if the target tone is white), the importance is set to zero.

[5] The storage values 0 to 255 correspond to importance values of $-0.14$ to 1.0. This range is a compromise between providing enough resolution in the positive values to distinguish differences in importance, and allowing negative values so that slightly overdarkened areas can be accommodated.

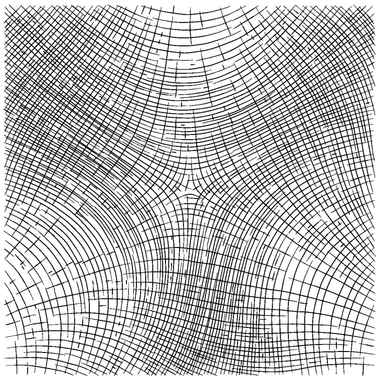


**Figure 6** A visualization of four quantities from a symmetric tensor field. The integral curves of the principle-direction field are shown by strokes; the density of the strokes in each direction is related to the magnitude of the principle value associated with that direction.

**Figure 7** Hair and face (after untitled photograph by Ralph Gibson [3].)

illustration. To find the location of $P_{i+1}$, we need to map the points along the segment $P_iP_{i+1}$ to locations $\gamma_i(s)$ in the illustration, for $0 \leq s \leq 1$. To define $\gamma_i$, let $\theta_i$ denote the angle between the vector $v_i = P_{i+1} - P_i$ and the vertical; for each $s$, we want the angle between the tangent $\gamma_i'(s)$ and the direction field at $\gamma_i(s)$, called $d(\gamma_i(s))$, to be $\theta_i$ as well. In addition, we want the arclength of $\gamma_i(s)$ between $s = 0$ and $s = 1$ to be the length of $v_i$. In summary, we want

$$\begin{aligned} \gamma_i(0) &= X \\ \text{angle}(\gamma_i'(s),\, d(\gamma_i(s))) &= \theta_i \\ \|\, \gamma_i'(s) \,\| &= \|\, v_i \,\| \end{aligned}$$

We solve this set of differential equations numerically, using Euler integration, and record $\gamma_i(1)$ as the place to map $P_{i+1}$. We repeat this process to place the remaining points of the hull. Because our strokes have many control points, this approach effectively warps the stroke so that at every point its angle to the direction field in the illustration is very similar to its angle to the vertical in the stroke example set.

***Clipping.*** Pen-and-ink artists have various rules for clipping strokes. One widely-accepted convention is that strokes do not cross object boundaries or boundaries between semantically different portions of objects, such as the edges of hard shadows [15]. We adhere to this convention by clipping strokes when they reach places where the direction field turns rapidly.[6] Strokes are also clipped when continuing to draw them would over-darken some region of the image. If a stroke is sufficiently short and has been clipped for this latter reason, it is removed altogether—pen-and-ink artists do not generally use short strokes to fill in every little bit of a dark area—and the importance value there is set to "below threshold" so that no further strokes will be draw into that area.

After the stroke is followed as far as possible in each direction from the pinned location, it is added to the illustration, and the difference and importance images are updated.

### 3.2 Updating the difference image

To quickly update the difference image with each added stroke, we sacrifice accuracy for efficiency through two approximations that seem to work well in practice.

The first approximation is that instead of blurring the current illustration after adding each stroke and subtracting the result from the tone image, we subtract a blurred version of the stroke from the difference image. This assumption amounts to presuming that the blurred version of multiple strokes will be the same as the sum of blurred versions of the individual strokes, which is fine when strokes do not overlap; when they do, we lighten the blurred version of the stroke as described below.

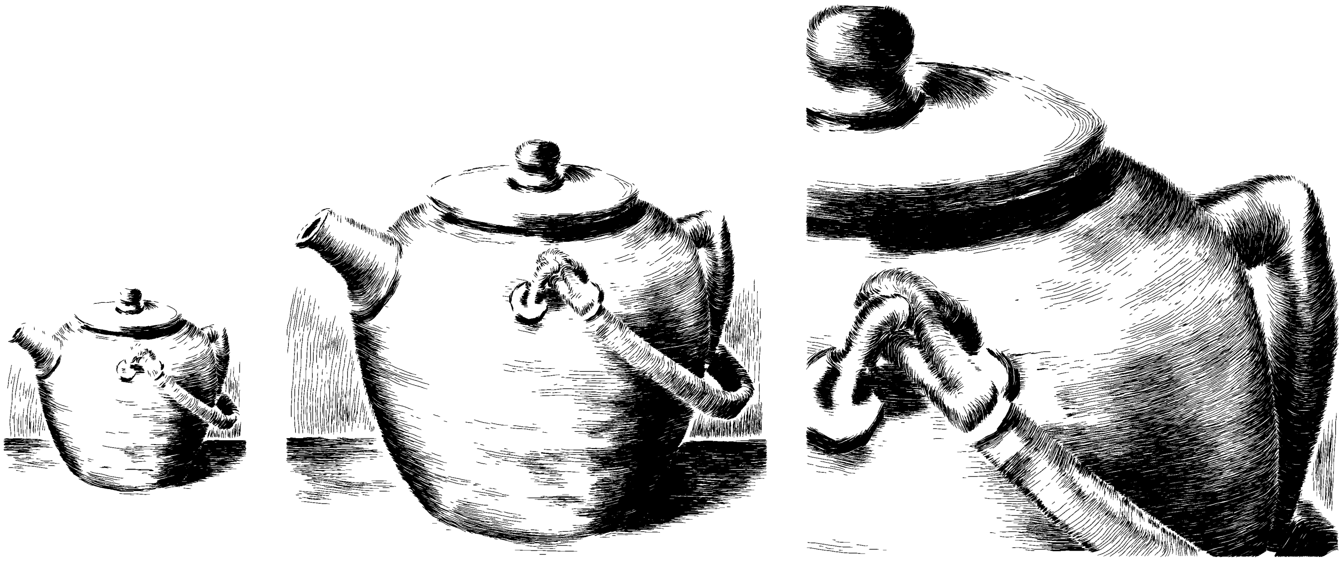
The second approximation is in our computation of the filtered image of a stroke. Instead of rendering the stroke itself, we render its control hull as a wide blurry line. The width $w$ is computed as $2h/t$ mm, where $h$ is the stroke thickness (in mm) and $t$ is the desired tone value between 0.0 (white) and 1.0 (black), and then clamped to the range 1–10 mm. We use Gupta-Sproull antialiased line drawing [4], but we supply the algorithm with a modified "darkness look-up table," whose width is as specified above, and whose height is twice the reciprocal of the width.[7] If the strokes are drawn with even spacing $w$, a nearly-constant blurred tone of average value $t$ results. In our Gupta-Sproull computation, we treat neither the endpoints nor major-axis-direction changes as exceptional cases. In practice, these simplifications seem to have had no discernible effect.

***Overlapping strokes and darkness adjustment.*** For light areas in the final illustration, strokes rarely overlap, whereas in dark areas they will often overlap. If each stroke in a dark region is counted as contributing as much darkness as a comparable stroke in a light area, the dark-area strokes will be overcounted: points where strokes cross will count as having been darkened twice or more. We therefore compute a *lightening factor*, which is a function of tone and the stroke example set. These lightening factors are computed in a preprocessing step: we draw many strokes into a buffer and record the buffer's darkness after each stroke. When we finish, we will know that, for instance, in an area of 50% grey, only 90% of the pixels drawn end up being visible; the rest overlap with other black pixels. In that case, when filling a region with a target tone of 50% grey, we would reduce the darkness of the filtered strokes to 90% before adding them to the blurred image, assuming that on average only 90% of their area does not overlap with other strokes in that region and will therefore actually contribute darkness to the illustration.

This approximation is not only faster than drawing-then-blurring, it also allows us to render a new stroke directly into the difference image without using a separate buffer. The lightening factor described above is incorporated into the "darkness look-up table" so that each stroke is drawn by looking at the underlying target tones. These tones determine which portion of the darkness look-up table

[6] Some automated assistance in detecting object boundaries would be valuable. We also intend to let the user draw into an "outline image," which would be used for both drawing outlines and truncating hatching strokes.

[7] For width $w$, height $h$ and distance from stroke center $x$, the look-up value is $(0.884/h)e^{-2.3(x/w)^2}$, which is simply a bump function that tapers to nearly zero.



(a) (b) (c)

**Figure 8** A teapot at three different scales (after illustration by Arthur Guptill [5].)

to use, and the values found there are directly incorporated into the difference image.

### 3.3 Output enhancements

The strokes to be drawn are deposited in a PostScript file, along with an interpreter that converts B-splines into drawable PostScript Bézier segments. We can also add two "stroke character" enhancements to the B-splines before printing (see the stroke detail inset of Figure 9).

The first enhancement is to render strokes with variable width.[8] Each stroke has three widths associated with it—one at each end and one in the middle. These widths are adjustable on a per-layer basis from the editing interface, and impart subtle expressive effects. Tapering the ends of strokes is ideal for rendering hair, but inappropriate for rendering hard shadows, for example.

The second enhancement is the addition of small "wiggles" to strokes more than 5mm long, to simulate a hand-drawn appearance. This effect is achieved by first resampling the control hull (except for the endpoints, which we copy), placing points with random spacing of about 4mm $\pm$1mm. We then randomly perturb each interior control point slightly along the angle bisector of its two adjacent sides, and perturb the two end control points both along and orthogonal to the control hull segments that they terminate. In the current system, the perturbations are uniformly distributed between $-0.15$mm and 0.15mm.

## 4 Results

The pen-and-ink illustration system was written in two linked parts: the user interface was written in C++, and the rendering engine was written in Modula-3. The interface runs at interactive speed, and the pen-and-ink renderer takes a few minutes to render the illustrations presented here (see Table 1).

We have produced several illustrations to test the capabilities of our system. Figures 5 and 8 are attempts to closely follow examples of real pen-and-ink drawings from illustration texts. Figure 8 also shows that our system can rescale illustrations while maintaining the character of their texture.

[8]The adjustments that are made are ignored in the computation of darkness—they are to be thought of as merely embellishments.

| Fig | Content | % Reduction | # Strokes | Time (sec) |
|---|---|---|---|---|
| 5 | Books | 58 | 16722 | 258 |
| 6 | Vectors | 35 | 665 | 25 |
| 7 | Hair/Face | 79 | 37618 | 788 |
| 8a | Teapot small | 65 | 2924 | 50 |
| 8b | Teapot | 65 | 8361 | 77 |
| 8c | Teapot closeup | 65 | 13617 | 200 |
| 9 | Raccoon | 62 | 55893 | 960 |

**Table 1** Illustration statistics and rendering timings measured on a Silicon Graphics workstation with a 180MHz R5000 processor.

Figure 6 shows a way of visualizing measured or computed vector fields using our system. It was created by bypassing the interactive stage of the system and feeding directions and tones directly into the renderer. Figures 7 and 9 show our ability to render non-smooth, difficult-to-model surfaces such as hair and fur. Our stroke lengths are approximately 1–10cm in the original PostScript rendering. This scale is similar to that at which pen-and-ink artists typically work. These artists often reduce their work for final presentation to achieve a finer, more delicate feel. We have done the same with our illustrations; the reductions are reported in Table 1.

## 5 Future work

Our current system suggests two principle areas for future research.

***Interactive illustrations.*** Currently the user interacts with the components of the underlying representation of the illustration. It would be nice for the user to have the option of interacting instead with the pen-and-ink illustration itself. Modifications to the illustration would be immediately reflected by corresponding changes in the tone or direction. While previous interactive systems [13] have allowed the user to directly manipulate the illustration, they do not—as does our system—allow the user to specify abstract high-level attributes of the illustration, and thus are not required to make a large number of changes as the result of a simple user action. With our system, changing the directions underneath the cursor can easily require removing and reapplying hundreds of strokes. Much of the incremental update mechanism needed for such behavior is already supported by our system, but we currently would require a considerable increase in rendering speed to make such an interface responsive enough to be usable.

**Figure 9** Raccoon with detail inset showing stroke character.

***Coherent textures.*** Many pen-and-ink drawings make use of textures such as bricks or shingles or fabrics that require strokes to appear in locally coherent patterns. Many artists also draw small groups of parallel hatches together in coherent clusters when filling in large areas of tone. We would like to support these kinds of coherent textures in our illustrations. The biggest difficulty is in dealing with diverging direction fields, since it is not obvious how to maintain local coherence and scale while following such a field without tearing the texture at some point.


### Acknowledgments

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 Augmentation award (N00014-90-J-P00002), and an industrial gift from Microsoft.


## References


[1] Debra Dooley and Michael Cohen. Automatic illustration of 3D geometric models: Lines. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, pp. 77–82, March 1990.

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

[3] Ralph Gibson. *Tropism: photographs*. Aperture, New York, 1987.

[4] S. Gupta and R. F. Sproull. Filtering edges for gray-scale displays. *Computer Graphics (SIGGRAPH '81 Proceedings)*, 15(3):1–5, August 1981.

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

[6] Paul Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics*, 24(4):207–214, August 1990.

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

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

[9] Barbara J. Meier. Painterly rendering for animation. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pp. 477–484. Addison Wesley, August 1996.

[10] Yachin Pnueli and Alfred M. Bruckstein. **Dig**$^{\mathbf{i}}{}_{D}$*ürer* — a digital engraving system. *The Visual Computer*, 10(5):277–292, 1994.

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

[12] Takafumi Saito and Tokiichiro Takahashi. NC machining with G-buffer method. *Computer Graphics*, 25(4):207–216, July 1991.

[13] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pp. 101–108. ACM Press, July 1994.

[14] Mike Salisbury, Corin Anderson, Dani Lischinski, and David H. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pp. 461–468. Addison Wesley, August 1996.

[15] Gary Simmons. *The Technical Pen*. Watson-Guptill Publications, New York, 1992.

[16] Thomas Strothotte, Bernhard Preim, Andreas Raab, Jutta Schumann, and David R. Forsey. How to render frames and influence people. *Computer Graphics Forum*, 13(3):455–466, 1994. Eurographics '94 Conference issue.

[17] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pp. 91–100. ACM Press, July 1994.

[18] Georges Winkenbach and David H. Salesin. Rendering free-form surfaces in pen and ink. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pp. 469–476. Addison Wesley, August 1996.