Computer-Generated Pen-and-Ink Illustration

Georges Winkenbach and David H. Salesin
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195

23 June 1994

Technical Report 94-01-08b

Abstract

This paper describes the principles of traditional pen-and-ink illustration, and shows how a great number of them can be implemented as part of an automated rendering system. It introduces "stroke textures," which can be used for achieving both texture and tone with line drawing. Stroke textures also allow resolution-dependent rendering, in which the choice of strokes used in an illustration is appropriately tied to the resolution of the target medium. We demonstrate these techniques using complex architectural models, including Frank Lloyd Wright's "Robie House."

Computer-Generated Pen-and-Ink Illustration

Georges Winkenbach *David H. Salesin*

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

Abstract

This paper describes the principles of traditional pen-and-ink illustration, and shows how a great number of them can be implemented as part of an automated rendering system. It introduces "stroke textures," which can be used for achieving both texture and tone with line drawing. Stroke textures also allow resolution-dependent rendering, in which the choice of strokes used in an illustration is appropriately tied to the resolution of the target medium. We demonstrate these techniques using complex architectural models, including Frank Lloyd Wright's "Robie House."

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism — Color, Shading, Shadowing, and Texture.

Additional Key Words: architectural rendering, comprehensible rendering, non-photorealistic rendering, prioritized stroke textures, resolution-dependent rendering, texture indication.

1 Introduction

Most of the research in computer graphics rendering over the last twenty years has been devoted to the problem of creating images of physical scenes with ever-increasing complexity and realism. The success of this research has been a well-heralded achievement in graphics.

However, the computer's ability to display images of ever-increasing complexity gives rise to a new problem: communicating this complex information in a comprehensible and effective manner. In order to communicate truly complex information effectively, some form of visual abstraction is required. This type of abstraction has been studied most comprehensively in the fields of graphic design and traditional illustration.

In this paper, we therefore examine algorithms for the "non-photorealistic" rendering of complex forms. While photorealistic images certainly have their place, in many applications, such as architectural and industrial design, a stylized illustration is often more effective.

The advantages of illustration are numerous. Illustrations can convey information better by omitting extraneous detail, by focusing attention on relevant features, by clarifying and simplifying shapes, or by exposing parts that are hidden. In addition, illustrations often consume less storage than realistic images, and are more easily reproduced and transmitted. Illustrations also provide a more natural vehicle for conveying information at different levels of detail. Finally, in many applications, illustrations can add a sense of vitality difficult to capture with photorealism.

The benefits of illustrations over photographs are well-recognized in many practical contexts. For example, medical texts almost always employ hand-drawn illustrations in place of (or in addition to) photographs, since they allow tiny and hidden structures to be much better described. In addition, most assembly, maintenance, and repair manuals of mechanical hardware employ illustrations rather than photographs because of their clarity. For example, at Boeing, even when CAD databases of airplane parts exist, all high-quality manuals are still illustrated by hand in order to provide more effective diagrams than can be achieved with either photorealistic rendering or simple hidden line drawings [16].

To explore the use of abstraction as a means for conveying information effectively, it makes sense to begin with an area with well established conventions. For this reason, we are beginning our investigation using the domain of pen-and-ink illustrations of architectural forms, for which a great number of well-documented conventions already exist [5, 11, 13, 14, 17, 20]. Restricting the domain to "pen and ink" also has the advantage that no exotic display technology is required to view the algorithms' output: conventional laser printers, even the inexpensive 300 dots-per-inch variety, give quite reasonable results.

In the rest of this paper, we describe a number of principles of traditional pen-and-ink illustration, and we show how a great number of them can be implemented as part of an automated rendering system.

1.1 Related work

The area of "non-photorealistic rendering" has received relatively little attention in the computer graphics community. We survey most of the related work here.

Seligmann and Feiner have described methods for automatically constructing illustrations to achieve a particular communicative goal [24]. Their system is primarily concerned with the high-level goal of composing the best model for communicating a particular intent, whereas the system we describe is more concerned with the low-level details of rendering the model once it is built. Thus, our system could serve as a "back-end" for theirs.

With respect to the rendering of architectural forms, Yessios described a prototype "computer drafting" system for common materials in architectural designs, including stones, wood, plant, and ground materials [26], which, like our work, attempts to provide a warmer, hand-drawn appearance as opposed to a mechanical one. Miyata also gave a nice algorithm for automatically generating stone wall patterns [19]; these patterns would make a good starting point for some of the pen-and-ink techniques described in this paper.

With respect to line-drawing techniques, Appel et al. were the first to discuss how a line could be "haloed" automatically to give the appearance of one line passing behind another [2]. Kamada and Kawai generalized this work by showing how different line attributes, such as dashed and dotted line, could be used to give a more informative treatment of hidden lines [12]. Dooley and Cohen later introduced more line qualities, such as thickness, and discussed how the treatment of outline and surface shading could be customized by a user to create more effective illustrations [6, 7]. In the commercial realm, the Premisys Corporation markets a product called "Squiggle" that adds waviness and irregularities to CAD output as a post-process, lending a hand-drawn appearance to the drawings [21]. The Adobe Dimensions program allows PostScript stroke textures to be mapped onto surfaces in three dimensions [1].

The research described in this paper was most directly inspired by the work of Saito and Takahashi, who introduced the concept of a "G-buffer" for creating comprehensible renderings of 3D scenes [22]. Our work takes a somewhat different approach, in that it integrates aspects of 2D and 3D rendering, whereas their method essentially uses image processing techniques once the set of G-buffers are created. In addition, by introducing methods for texturing surfaces with strokes, the work in this paper extends the repertoire of the types of renderings that can be produced in a purely automated way.

In related works, our group is exploring several different aspects of the pen-and-ink illustration problem. This paper describes the overall vision of computer-generated illustration, surveys principles from traditional illustration, and shows how they can be incorporated into an automated system for rendering 3D models. A second paper discusses the issues of creating pen-and-ink illustrations interactively, with an emphasis on using 2D greyscale images as a starting point [23]; in this interactive work, the responsibility of producing an effective illustration is primarily the artist's. A third paper examines the issues involved in representing, editing, and rendering the individual strokes that are the building blocks of any line illustration system [8].

1.2 Overview

The rest of this paper is organized as follows. Section 2 surveys the principles of traditional pen-and-ink illustration. Section 3 discusses how these principles can be used to guide the design of an automated system for producing this type of imagery. Section 4 introduces "strokes" and "stroke textures," the building blocks of our system, and describes how they can be used to implement many of the traditional illustration principles. Section 5 discusses some of our results, and Section 6 lays out an agenda for future research in the area. Finally, the appendix gives details about the implementation.

2 Principles of pen-and-ink illustration

While pen-and-ink drawing has a long history, dating back to the illuminated manuscripts of the Middle Ages, it is only relatively "recently" — that is, since the end of the 19th century — that pen-and-ink illustration has been developed as an art form in and of itself.

Pen-and-ink illustration is a limiting medium. The pen gives off no color or tone, so both color and shading must be suggested by combinations of individual strokes. Furthermore, when rendered manually, it is very difficult and time-consuming with pen and ink to cover a large area with tone, and it is practically impossible to lighten a tone once it is drawn.

However, pen-and-ink illustrations have some particular qualities that make them especially attractive. First, they are ideal for outlines: each individual pen-and-ink stroke can be made expressive by employing small irregularities in its path and pressure. Second, pen and ink provide a real economy of expression in representing tones and texture: the character of a few small strokes can clearly indicate the difference between textures like smooth glass and old knotted wood.

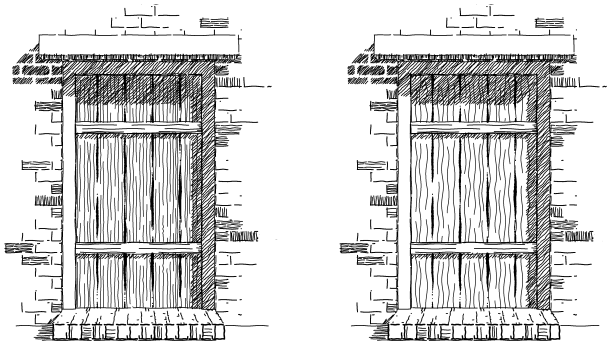

Figure 1: *Two doors. The lines of wood grain are drawn with an even pressure, while the lines between the planks use varying pressure. For wood grain, we typically use the waviness function of the left door, while that of the right door has been exaggerated.*

In addition to these concrete advantages, pen-and-ink drawings by their very nature possess some special qualities that are difficult to capture in other media. Their simplicity provides an appealing crispness and directness. Finally, pen-and-ink illustrations blend nicely with text, due to their linear quality and their use of the same ink on the same paper, making them ideal for printed publications.

In the rest of this section, we survey some of the fundamental principles of illustrating in pen and ink. These principles are distilled primarily from Guptill's classic text, *Rendering in Pen and Ink* [11], and also from Lohan's *Pen&Ink Techniques* [17] and several other sources [5, 13, 14, 20]. While the field of pen-and-ink is too vast to allow a comprehensive treatment within the scope of this paper, the principles described here should be sufficient to motivate many of the design choices for a computer-graphics system. We organize our treatment into three parts: Strokes, Tone and texture, and Outline.

2.1 Strokes

In classical pen-and-ink illustration, a "stroke" is produced by placing the point, or "nib," of a pen in contact with the paper, and allowing the nib to trace out a path. The thickness of the stroke can be varied by varying the pressure on the nib.

Some principles of stroke-drawing are summarized below:

- *Too thin a stroke can give a washed-out appearance; too coarse can detract from the delicate details.*
- *It is frequently necessary to vary the pen position, with the nib sometimes turning as the stroke is drawn.*
- *Strokes must look natural, not mechanical. Even-weight line drawings appear lifeless; instead, the thickness of a line should vary along its length.*
- *Wavy lines are a good way to indicate that a drawing is schematic and not yet completely resolved.*

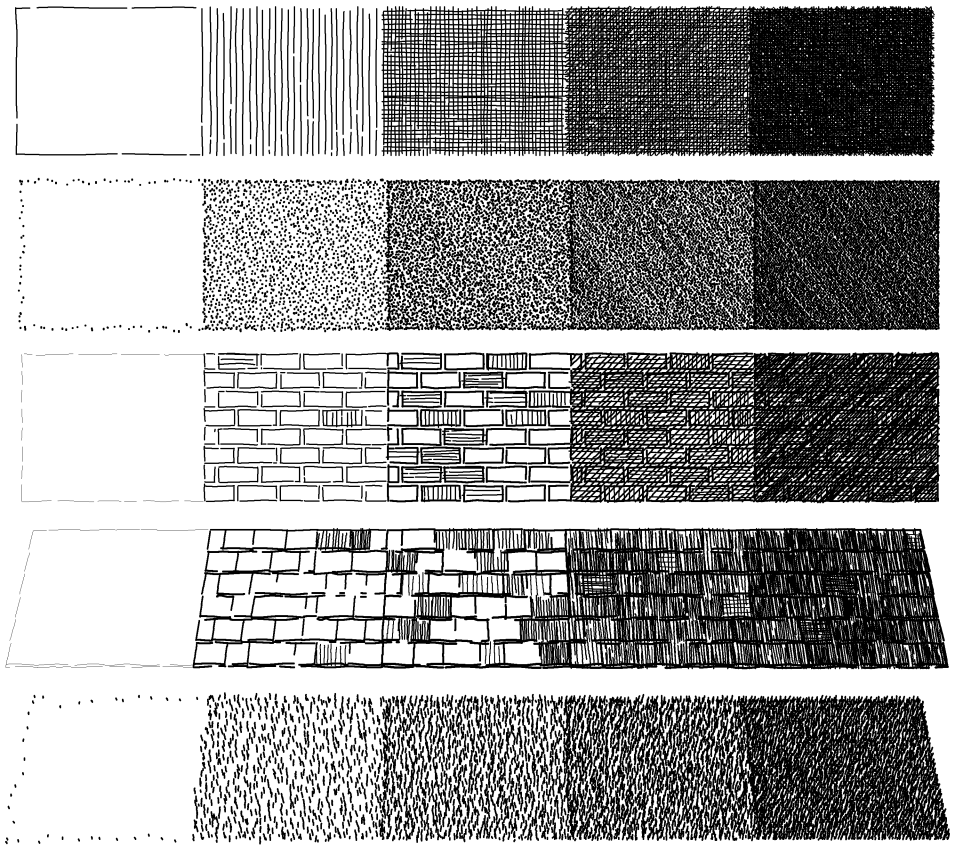

Figure 2: *Using strokes to indicate both texture and tone. The stroke textures used, from top to bottom, are: "cross-hatching," "stippling," "bricks," "shingles," and "grass." Notice how the outline style of the white areas is also particular to each texture.*

2.2 Tones and texture

The terms "value" and "tone" are used interchangeably to refer to the amount of visible light reflected toward the observer from a point on a surface. In traditional pen-and-ink illustration, it is impossible to portray the value of each surface precisely; instead, combinations of strokes are used to create an overall impression of the desired tone.

The tone achieved by a combination of strokes is a function of the ratio of black ink to white paper over a given region of the illustration. If the character of the strokes is varied, then the same strokes that are used to achieve a particular tone can also be used simultaneously to indicate the "texture" of the subject being rendered. This dual role of individual strokes to convey both tone and texture is part of the economy of pen-and-ink illustration.

Here are some of the principles of drawing tones and textures with pen-and-ink strokes:

- *Tones should be created from lines of roughly equal weight and spacing.*
- *It is not necessary to depict each individual tone accurately; however, presenting the correct arrangement of tones among adjacent regions is essential.*
- *To disambiguate objects, it is sometimes important to "force tone" by enhancing contrast or inventing shadows.*
- *The character of strokes is important for conveying texture, as well as geometry and lighting. For example:*
  - *Crisp, straight lines are good for "glass."*
  - *Horizontal surfaces should be hatched with predominantly horizontal lines.*
  - *Absence of detail altogether indicates glare.*
  - *A sketchy kind of line is good for "old" materials, while careful stippling is good for "new" materials.*
- *To lend economy to the illustration, it is important to utilize some form of "indication" for conveying the impression of a texture without drawing every single stroke. The method of indication should also be varied across the drawing to avoid monotony.*

2.3 Outlines

Realistic scenes contain no real outlines; instead, forms are defined by variations in texture and tone. However, outline is nevertheless a very natural means for portraying objects — for example, most children's drawings utilize outline almost exclusively.

The medium of pen and ink is ideal for creating outlines with an incredible range of expressiveness. The pen allows for outlines that change thickness, sometimes disappearing altogether. In addition, the character of the outline stroke can be a very powerful indicator of texture.

Outline strokes are used not only for the contours of an object, but also for delineating the essentials of its interior. For example, in an illustration of a leaf, the veins are typically rendered in outline, in addition to the contour.

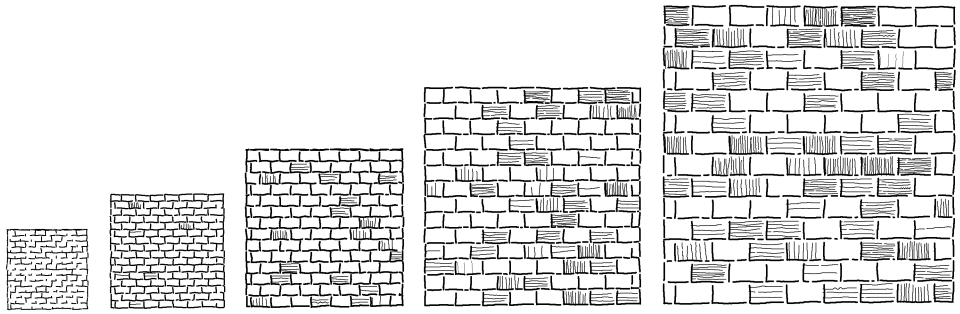

Figure 3: *Creating the same texture and tone at different scales. At the smallest scale, the brick outline strokes are sufficient to build the tone. As the scale increases, the prioritized stroke texture automatically introduces shading inside the bricks to maintain the tone. The same technique applies to generating illustrations at the same scale but for different printer resolutions.*

Different styles of rendering use various combinations of outline and tone; all combinations are possible. Moreover, there exists an entire spectrum between outline and tone illustrations: as outline drawings become increasingly complex, they begin to take on more and more aspects of an illustration with tone.

Here are some of the important classical principles for drawing expressive outlines:

- *The quality of the outline stroke is important for conveying texture. For example, crisp straight lines are good for hard objects, while a greater variety of line quality is better for soft objects.*
- *Thick outlines are used to suggest shadows, or to bring one object or part of an object forward in the scene. Thick line junctions are used to suggest darkness where objects overlap and to add "snappiness" to the illustration.*
- *Outlines should become "haloed" and fade away where one object passes behind another object.*
- *Outlines must be introduced where tones are omitted to convey shape.*
- *Using "indication" for drawing outlines is just as important as for drawing tones.*

3 Computer-generated pen-and-ink illustration

Implementing these principles of pen-and-ink illustration as part of an automated system presents an interesting challenge. A reasonable starting point is to take the traditional "graphics rendering pipeline" for photorealistic imagery and see which parts, if any, need to be altered in order to support this style of non-photorealism.

We identified two fundamental differences:

1. *The dual nature of strokes.* In the traditional graphics pipeline, the renderings of texture and tone are completely independent. A texture is typically defined as a set of images assigned to each surface, which affect the shading parameters. Tone is produced by dimming or brightening the rendered shades, while leaving the texture invariant. However, for pen-and-ink illustration, the very same strokes that produce tone must also be used to convey texture. Thus, tone and texture must become more tightly linked in a system for producing this type of imagery.

2. *The need to combine 2D and 3D information.* In the traditional graphics pipeline, the information used for rendering is entirely three-dimensional, with the final projection to two dimensions largely a matter of sampling the rendered shades. For pen-and-ink illustration, the 2D aspects of the particular projection used are every bit as essential as the 3D information for creating a proper rendering. The necessary 2D information takes a number of forms. First, the size of the projected areas must be used to compute the proper stroke density, in order to accommodate the dual nature of strokes described above. In addition, the 2D adjacencies of the projected geometry must also be used, since outlining depends on such issues as the type of junction between 2D boundaries (whether two adjacent regions in 2D are adjoining in 3D or passing one behind the other), and the level of contrast between tones of adjacent 2D regions.

Thus, our rendering system is a basic graphics pipeline with a few notable changes. The standard aspects of the pipeline include:

- *The model.* Any standard polygonal 3D model will do.
- *The assignment of texture.* Textures are assigned to 3D surfaces in the usual way. However, the textures are no longer described by images, but by "stroke textures," as discussed in the next section.
- *The lighting model.* Any standard illumination model can be employed to compute a "reference solution," which is then used as a target for tone production with strokes. We use the Phong model, which, although not physically-based, appears to be quite adequate for most non-photorealistic rendering.
- *The visible surface algorithm.* Any object-space or list-priority visible surface algorithm will do; we use BSP trees in our implementation.
- *Shadow algorithm.* The shadow algorithm must also use an object-space or list-priority method; we use Chin and Feiner's BSP tree shadow volumes [4].

Here are the notable differences from the standard pipeline:

- *Maintaining a 2D spatial subdivision.* The need to consider 2D adjacency information in rendering suggests the use of some form of spatial subdivision of the visible surfaces. We use a half-edge data structure for maintaining this planar map [18].
- *The rendering of texture and tone.* Polygons are no longer scan converted; instead, both texture and tone must be conveyed with some form of hatching. The stroke textures we define in the next section achieve this effect.
- *Clipping.* The strokes must be clipped to the regions they are texturing. Since so many strokes are drawn, the clipping must be extremely fast. In addition, in order to simulate a hand-drawn

effect, the clipping should not be pixel-based — that is, it should not remove just those pixels of the stroke that are outside the clipping region — since this gives an unnatural, mechanical appearance. Instead, the clipping should be stroke-based, allowing a wavy stroke to sometimes stray slightly outside of the clipping region. To achieve this effect, we clip the straight-line paths of our strokes prior to adding in the function for waviness (see Section 4.1). For fast clipping, we use set operations on a 2D BSP tree representation of the planar map [25].

- *Outlining.* Outlines play a significant role in pen-and-ink illustration. Outlines come in two varieties. The "boundary outlines," which surround visible regions, must be drawn in a way that takes into account both the textures of the surrounded regions, and the adjacency information stored in the planar map. In addition, "interior outlines" are used within polygons to suggest shadow directions or give view-dependent accents to the stroke texture.

A brief description of the rendering process follows; more details about the rendering algorithm can be found in the appendix. To render a scene, the system begins by computing the visible surfaces and the shadow polygons. It then uses these polygons, projected to Normalized Device Coordinate (NDC) space, to build the 2D BSP tree and the planar map. Each visible surface is then rendered. The procedural texture attached to each surface is invoked to generate the strokes that convey the correct texture and tone for the surface. All the strokes are clipped to the visible portions of the surface using set operations on the 2D BSP tree. Finally, the outline strokes are drawn by extracting from the planar map all of the outline edges necessary for the illustration, as described in Section 4.3.

4 Strokes and stroke textures

In this section we discuss strokes and stroke textures, the essential building blocks of our system.

4.1 Strokes

In our system, all strokes are generated by moving a nib along a basic straight path. Character is added to the stroke by perturbing the path with a *waviness function* and by varying the pressure on the nib with a *pressure function*. Figure 1 demonstrates some of the effects that can be achieved with different waviness and pressure functions. A more detailed explanation of our strokes can be found in Appendix A.3.1.

4.2 Stroke textures

A *stroke texture* is a collection of strokes used to produce both texture and tone. We define a *prioritized stroke texture* as a set of strokes each with an associated priority. When rendering a prioritized stroke texture, all of the strokes of highest priority are drawn first; if the rendered tone is still too light, the next highest priority strokes are added, and so on, until the proper tone is achieved.

For our stroke textures, we assign different aspects of the texture different priority. For example, for a "brick" texture, the outlines of the individual brick elements have highest priority, the strokes for shading individual bricks have medium priority, and the hatching strokes that go over the entire surface have lowest priority. In the cross-hatching texture, vertical strokes have priority over horizontal strokes, which have priority over the various diagonal stroke directions. Figure 2 demonstrates several greyscales of tone produced using different procedural prioritized stroke textures, including "cross-hatching," "stipple," "brick," "shingle," and "grass." For each texture, the relative priorities of the strokes can be seen from the collection of strokes used to achieve a particular value of grey. More details about the procedural methods for our stroke textures are given in Appendices A.3.2 and A.3.3.

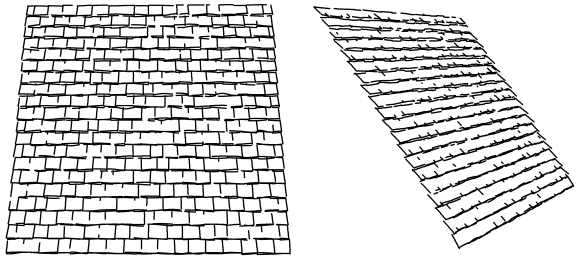

Figure 4: *The effect of changing view direction on outline strokes of a shingle texture. Notice how the vertical edges begin to disappear as the texture is viewed from a more edge-on direction.*

Although not explored in this paper, the idea of prioritized stroke textures is general enough to support many kinds of non-procedurally generated textures as well, such as textures drawn directly by an artist, or strokes produced through edge extraction from a greyscale image. These kinds of non-procedural stroke textures are explored in more detail by Salisbury et al. [23].

4.2.1 Resolution dependence

A common problem with the figures created by existing computer drawing programs is that they do not scale well when printed at different sizes or resolutions. Enlargement is typically performed either by pixel replication, which yields ugly aliasing artifacts, or by drawing the same strokes at higher resolution, which yields thinner strokes and an overall lighter illustration. Reduction is almost always performed by scan-converting the same curves at a lower resolution, often yielding a large black mass of overlapping strokes. Printing speed is also a common problem with illustration reduction, since the same number of strokes needs to be transmitted to and rendered by the printer, even when a smaller number of strokes would have sufficed (and actually have been preferable from an aesthetic standpoint, as well).

The prioritized stroke textures described here do not suffer from these problems. Strokes are chosen to provide the proper texture and tone for a given illustration size and printer resolution, as demonstrated in Figure 3. Note that for smaller images or coarser resolutions, fewer strokes are required, improving printing efficiency. Efficiency can be improved still further by rendering a simplified approximate version of each stroke, accurate to within one printer pixel [8].

4.2.2 Indication

As discussed in the principles of texture generation, it is important to suggest texture without drawing every last stroke. This principle of "indication" lends economy to an illustration. It also makes an illustration more powerful by engaging the imagination of the viewer rather than revealing everything.

Indication is one of the most notoriously difficult techniques for the pen-and-ink student to master. It requires putting just enough detail in just the right places, and also fading the detail out into the unornamented parts of the surface in a subtle and unobtrusive way.

Clearly, a purely automated method for artistically placing indication is a challenging research project.

We therefore decided to compromise and implement a semi-automated method, whereby the user specifies at a very high level where detail should appear in the drawing, and indication is used everywhere else. For easy specification of the areas of detail, we borrowed the idea of using "fields" generated by line segments from the morphing paper of Beier and Neely [3]. The user interactively places "detail segments" on the image to indicate where detail should appear. Each segment is projected and attached to the texture of the 3D surface for which indication is being designed.

A field $w(x, y)$ is generated by the detail segment ℓ at a point (x, y) in texture space according to

$$w(x, y) \quad = \quad (a + b * \mathit{distance}((x, y), \ell))^{-c}$$

where a, b, and c are non-negative constants that can be used to change the effect of the field. When several detail segments are present, we define the field at a point (x, y) to be that of the closest segment. So as not to create patterns that are too regular, the field $w(x, y)$ is perturbed by a small random value. Textures such as "bricks" and "shingles" evaluate the strength of the field of indication at the center of each brick or shingle element. The set of strokes for that element is generated only if the indication field is above some preset threshold.

This approach seems to give reasonable results, as demonstrated in Figures 6 and 9. Figure 5 shows the detail segments that were used to generate Figure 6.

4.3 Outline

As described in Section 3, outlines come in two varieties: *boundary* and *interior* outlines. The *boundary outlines* surround the visible polygons of the image, and must be drawn in a way that takes into account both the textures of the surrounded regions, and the adjacency information stored in the planar map. The *interior outlines* are used within polygons to suggest shadow directions or to give view-dependent accents to the stroke texture.

In our implementation we have tried to address many of the principles for the effective use of these two types of outline, as described below.

Expressing texture with outline. Each stroke texture $\mathcal{T}$ has associated with it a *boundary outline texture*, which is used whenever the outline of a polygon textured with $\mathcal{T}$ is rendered. The boundary outline textures for some of our procedural textures are demonstrated in the white squares of Figure 2. These boundary outline textures are also displayed with and without their accompanying stroke textures in the illustrations of Figure 7.

Minimizing outline. Let E be an edge that is shared by two faces F and G of a planar subdivision. Our rendering algorithm draws E only if the tones of face F and G are not sufficiently different for the two faces to be easily disambiguated by their shading alone. In this sense, we minimize the use of boundary outline strokes. When a boundary outline stroke *is* drawn, it must be rendered according to the boundary outline texture for one of the two faces F or G. We choose the texture of the face of the planar subdivision that represents a polygon closer to the viewer. Figure 8 demonstrates how outline is omitted in the presence of sharp changes in tone, and added in the absence of tone changes.

Accented outlines for shadowing and relief. "Accenting," or thickening, outline edges is a technique for providing subtle but important cues about the three-dimensional aspects of an illustrated scene. In our implementation, the interior outlines of each brick in the "brick" stroke texture are drawn according to their relationship with the direction of the light source: brick edges that cast shadows are rendered with thickened edges, while illuminated brick edges are not drawn at all. Figure 10 demonstrates this effect.

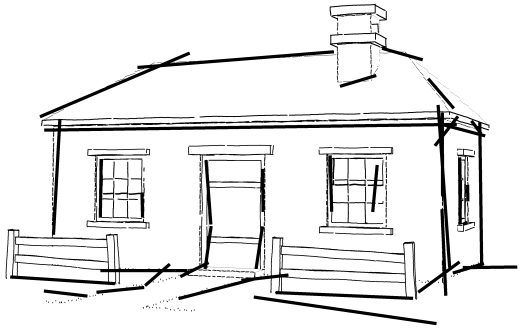

Figure 5: *Detail segments. The user interactively attaches "detail segments" to the surfaces to indicate roughly where details should appear.*

Dependence of viewing direction. In addition to the light source direction, the viewing direction is another important parameter that should be taken into account when drawing outline strokes. For example, consider a roof of shingles. Viewed from above, all edges between individual shingles are clearly visible; viewed from more to the side, however, the shingles tend to blend together, and vertical edges begin to disappear, leaving the horizontal edges predominant. This effect is demonstrated in Figure 4. To implement this effect, each stroke texture is outfitted with a very simplified "anisotropic bidirectional reflectance distribution function" (BRDF), to borrow a term from radiometry, which describes its outline features in terms of both the lighting and viewing directions.

5 Results

Our computer-generated pen-and-ink illustration system was used to create all the figures in this paper. The system was developed on a Macintosh Quadra 700 using ThinkC.

The only input to the program is the scene geometry, including texture assignments for each surface, and some field lines for specifying the "indication."

We also used the system to generate an image of the top two floors of Frank Lloyd Wright's "Robie House," as shown in Figure 9. The model consist of 1043 polygons. It took 30 minutes to compute and print the image. Of this time, 22 minutes were devoted to computing the planar map from the input geometry, and 8 minutes were required for actually rendering the image at 600 dots per inch.

6 Summary and future work

This paper does not propose any radically new algorithms or present any complex mathematics. However, we feel it nevertheless provides a number of contributions to the computer graphics community. These contributions include:

- Surveying established principles from traditional illustration that can be used for communicating visual information effectively.
- Showing that a large number of these principles can be incorporated as part of an automated rendering system, and that the

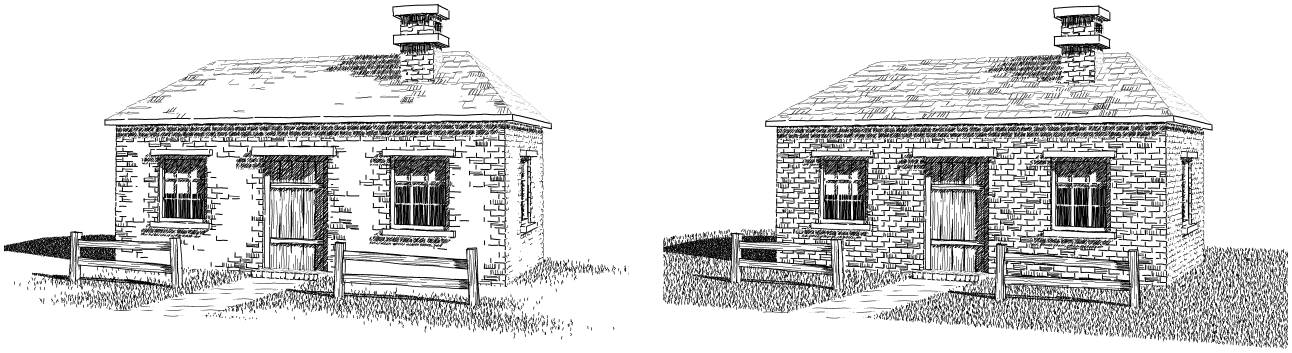

Figure 6: *Indicating texture. The left house is drawn using "indication"; the right house is not.*

information present for driving the ordinary graphics pipeline is in many respects also sufficient for achieving important non-photorealistic effects.

- Introducing the concept of a "prioritized stroke texture," a general framework for creating textures from strokes, and providing a methodology for building procedural versions of these textures.
- Allowing a form of resolution-dependent rendering, in which the choice of strokes used in an illustration is appropriately tied to the resolution of the target medium.

However, the work described in this paper is just one early step in the exploration of automated non-photorealistic rendering algorithms. There are many ways to extend this work, including:

- Improving the procedural stroke textures, and automating further our methods for creating them.
- Incorporating other illustration effects, such as exploded, cut-away, and peel-back views, for showing parts that are hidden.
- Adding more interactive controls to help in designing 3D illustrations. Also, experimenting with very high-level controls—for example, a control to add emphasis to parts of an illustration, which would work by automatically accentuating and suppressing detail over different parts of the image.
- Rendering other natural forms that appear in architectural drawings (and for which established conventions also exist), such as trees, grass, water, human figures, etc.
- Rendering other types of databases besides architectural models, such as databases of mechanical parts. Also, applying traditional illustration techniques and principles to databases that are not inherently visual in nature, such as flow simulations or higher-dimensional datasets. (This variety of rendering could be thought of as a form of scientific visualization.)
- Creating animations. Because our system uses randomness profusely, issues in frame-to-frame coherence arise. For instance, large features that are random, such as the selection of bricks that are shaded, should not vary from frame to frame. However, more subtle features, such as the waviness of strokes used to give the hand-drawn appearance, should be allowed to waver [15].
- Exploring other forms of illustration besides pen-and-ink, including traditional forms like watercolor and air brushing, as well as new methods of conveying information visually that may not necessarily mimic traditional forms.

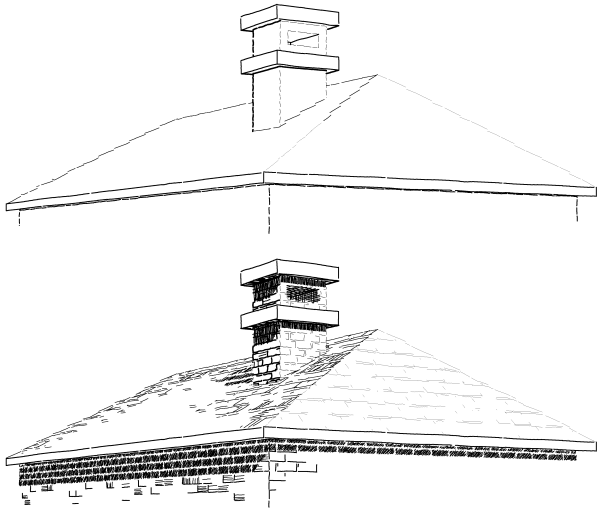

Figure 7: *Indicating texture through outline. Notice how different textures are delineated with different styles of boundary outlines. The upper and lower illustrations are the same, except that all but the boundary outline textures have been removed in the upper illustration to present the different styles more clearly.*

Acknowledgments

We wish to thank Tony DeRose for many helpful ideas along the way. Thanks also to Dan Ambrosi who advised us on illustration techniques and who also suggested the "Robie House" as a model. We are also grateful to Taweewan Winkenbach who spent many hours building the model for the "Robie House".

This work was supported by NSF Presidential and National Young Investigator awards (CCR-8957323 and CCR-9357790), by the University of Washington Graduate Research and Royalty Research Funds (75-1721 and 65-9731), and by industrial gifts from Adobe, Aldus, and Xerox.

References

[1] Adobe Systems Incorporated, Mountain View. *Adobe Dimensions*, 1992.

[2] Arthur Appel, F. James Rohlf, and Arthur J. Stein. The haloed line effect for hidden line elimination. Proceedings of SIGGRAPH '79 (Chicago, Illinois, August 8-10, 1979. In *Computer Graphics* 13, 2 (August 1979), 151–157.

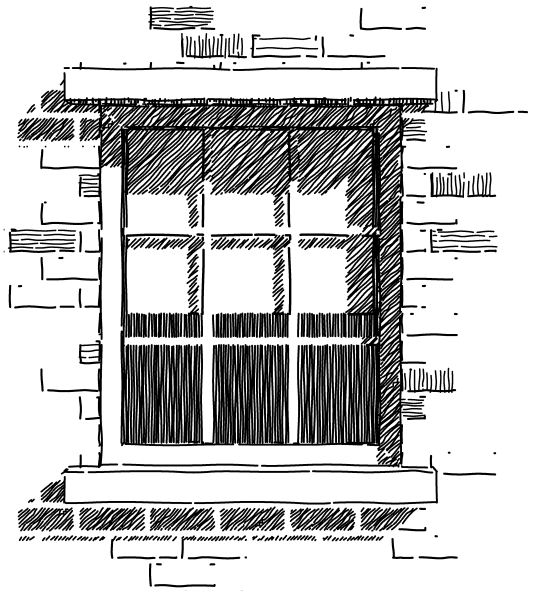

Figure 8: *Outline minimization. Notice how the boundary edges on the vertical and horizontal dividers between the panes appear only where contrast with the adjacent surface is low. (These boundary outlines are also omitted when they face the light source.)*

[3] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 35–42.

[4] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31 - August 4, 1989). In *Computer Graphics* 23, 3 (July 1989), 99–106.

[5] Frank Ching. *Architectural Graphics*. Van Nostrand Reinhold Company, New York, 1975.

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

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

[8] Adam Finkelstein and David H. Salesin. Multiresolution curves. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics*, Annual Conference Series, 1994.

[9] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. Proceedings of SIGGRAPH '80 (Seattle, Washington, July 14-18, 1980). In *Computer Graphics* 14, 3 (July 1980), 124–133.

[10] Leonidas Guibas, Lyle Ramshaw, and Jorge Stolfi. A kinetic framework for computational geometry. In *Proceedings of the 24th IEEE Annual Symposium on Foundations of Computer Science*, pages 100–111, 1983.

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

[12] Tomihisa Kamada and Saturo Kawai. An enhanced treatment of hidden lines. *ACM Transaction on Graphics*, 6(4):308–323, October 1987.

[13] Stephen Klitment. *Architectural Sketching and Rendering: Techniques for Designers and Artists*. Whitney Library of Design, New York, 1984.

[14] Paul Laseau. *Architectural Drawing: Options for Design*. Design Press, New York, 1991.

[15] John Lasseter. Personal communication, January 1994. Pixar, Richmond, California.

[16] John Lewis. Personal communication, November 1993. Boeing Computer Services, Seattle, Washington.

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

[18] Martti Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland 20850, 1988.

[19] Kazunori Miyata. A method of generating stone wall patterns. Proceedings of SIGGRAPH '90 (Dallas, Texas, August 6-10, 1990). In *Computer Graphics* 24, 4 (August 1990), 387–394.

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

[21] The Premisys Corporation, Chicago. *Squiggle*, 1993.

[22] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3D shapes. Proceedings of SIGGRAPH '90 (Dallas, Texas, August 6-10, 1990). In *Computer Graphics* 24, 4 (August 1990), 197–206.

[23] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics*, Annual Conference Series, 1994.

[24] Dorée Duncan Seligmann and Steven Feiner. Automated generation of intent-based 3D illustration. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *Computer Graphics* 25, 4 (July 1991), 123–132.

[25] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. Proceedings of SIGGRAPH '87 (Anaheim, California, July 27-31, 1987). In *Computer Graphics* 21, 4 (July 1987), 153–162.

[26] Chris I. Yessios. Computer drafting of stones, wood, plant, and ground materials. Proceedings of SIGGRAPH '79 (Chicago, Illinois, August 8-10, 1979. In *Computer Graphics* 13, 2 (August 1979), 190–198.

A Implementation Details

A.1 Overview

Three main global data structures are used by our system:

- *The model M*. The model is stored as a collection of polygons in three-space. For convenience, concave polygons and polygons with holes are decomposed into convex polygons.
- BspTree. The 2D BSP tree [25] is a representation of the visible polygons projected to Normalized Device Coordinates (NDC) space. It is used for fast clipping of strokes.
- PlanarMap. The planar map [18] is a partition of the NDC plane into *vertices*, *edges*, and *faces*, according to the NDC projections of the visible polygons. It is used to generate the outline strokes of the surfaces.

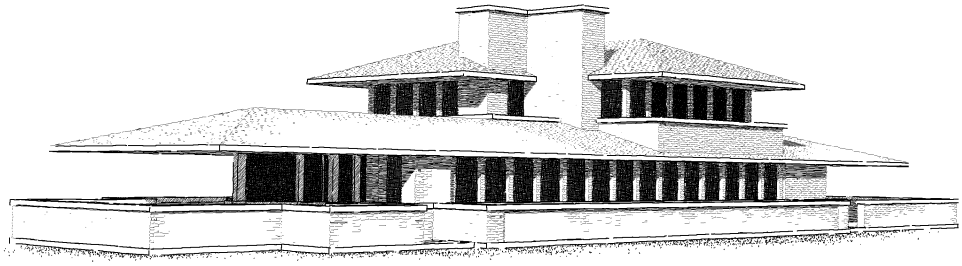

Figure 9: *Frank Lloyd Wright's "Robie House."*

The rendering process is structured as follows:

procedure *RenderScene*(M)*:*
  ($BspTree$, $PlanarMap$) $\leftarrow$ *VisibleSurfaces*(M)
  for each visible surface $S \in M$ **do**
    Strokes $\leftarrow$ *Texture*(S, *Tone*(S))
    for each stroke $s \in$ *Strokes* **do**
      Render(*ClippedStroke*(s, $BspTree$))
    end for
  Render(*ConstructMinimalOutline*(S, $PlanarMap$))
end procedure

The following sections describe the individual stages of the rendering process in more detail.

A.2 Computing the visible surfaces

We use a 3D BSP tree to compute visibilities [9], and Chin and Feiner's shadow volumes [4] to compute the shadow polygons. The result is a set of convex polygons that can easily be ordered in depth with respect to the view point. To build the 2D BSP tree, the visible polygons are examined in front-to-back order. Each polygon is first projected to NDC space, and then inserted into the 2D BSP tree. The insertion into the 2D BSP tree is equivalent to the set union operation described by Thibault and Naylor [25], except that "in" leaf nodes carry an additional pointer back to the 3D polygon from which they originate. As such, the 2D BSP tree forms a partition of NDC space, with each cell in the partition corresponding either to a unique frontmost polygon in the 3D scene, or to the background.

The planar map data structure is computed with the help of the 2D BSP tree. We begin by inserting a single rectangular region, representing the entire drawing surface in NDC space, into the tree. As each node of the tree is traversed, the original region is partitioned into smaller and smaller faces in each branch of the tree. Faces reaching an "out" leaf node are tagged as background faces. Faces reaching an "in" leaf node receive a pointer to the corresponding 3D polygon in M. The BSP tree node also receives a pointer to the planar map face. Because of numerical inaccuracies, it is possible that some leaf nodes in the BSP tree never receive a matching face in the planar map. During clipping, a segment that falls in a leaf node having no planar map pointer is simply discarded. Because such nodes correspond to extremely thin regions, no visible artifacts result.

Geometrically, the planar map and the BSP tree are redundant: they encode the same 2D partition. However, the two data structures are amenable to different tasks. The BSP tree is efficient for clipping strokes through set operations, but does not readily allow searching among neighboring polygons. By contrast, the planar map encodes polygon adjacencies, but does not lend itself as well to clipping.

A.3 Rendering the textures

A.3.1 Individual strokes

A *stroke* $\mathcal{S}$ consists of three parts:

- a *path* $P(u) : [0,1] \rightarrow \mathbb{R}^2$, giving the overall "sweep" of the stroke, as a function of the parameter u.
- a *nib* $\mathcal{N}(p)$, defining the cross-sectional "footprint" of the stroke, as a function of the pressure p on the nib.
- a *character function* $C(u) = (C_w(u), C_p(u))$, describing the *waviness* of the curve $C_w(u)$ (how the curve departs from its path) and the pressure $C_p(u)$ on the nib.

The stroke $\mathcal{S}$ is defined as all pixels in the region

$$\mathcal{S} \quad = \quad (P(u) + C_w(u)) * \mathcal{N}(C_p(u))$$

where $*$ denotes the convolution of two parameterized point sets $A(u)$ and $B(u)$ of the Cartesian plane $\mathbb{R}^2$. This convolution is defined as [10]:

$$A(u) * B(u) \quad = \quad \bigcup_{u \in [0,1]} \{a + b \,|\, a \in A(u) \,\wedge\, b \in B(u)\}.$$

A stroke $\mathcal{S}$ is rendered by scan-converting the path (after waviness is added) and stamping a copy of the nib, scaled by the pressure value, in place of drawing each pixel. Note that more efficient scan-conversion methods undoubtedly exist. Indeed, the investigation of a good representation for individual strokes, including their overall sweep and character functions, is a sizable research topic in and of itself [8].

All strokes are drawn by a C++ object named *InkPen*. An *InkPen* is in turn composed of three objects: a *Nib*, a *WavinessFunction*, and a *PressureFunction*. Different pens can be created by assembling various combinations of these components. So far, we have only used circular nibs of variable radius, and a sine-wave waviness function with randomly perturbed amplitude and wavelength. Two kinds of pressure functions are used throughout the images in this paper: a simple "broken-line" function that lifts the pen off the paper with some randomness, and a random sine wave function that creates strokes of varying thickness. Although our implementation does not allow for all the generality of real pen and ink as described in Section 2.1, the limited set of functions we have implemented still allows for a fairly wide range of expressiveness.

An *InkPen* supports methods to: scale the nib size; query the amount of ink deposited between two points when using a particular nib size; and draw a stroke between two points to achieve a particular darkness, in which case the darkness of the stroke will be appropriately modulated by the *PressureFunction* of the pen.

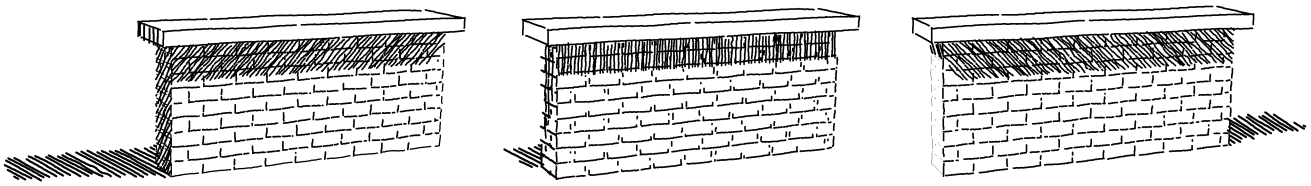

Figure 10: *Illuminated bricks. Notice how the thickened edges change to follow the shadow direction.*

A.3.2 Building tone from strokes

The first step in building tone is to compute a reference shade $v \in [0 = white, 1 = black]$; we use a simple Phong model in our implementation. A procedural stroke texture is then used to produce a set of strokes that together achieve the target grey value v. As a simple example, to achieve a tone v over an area A using a nib of width w with non-overlapping strokes requires drawing strokes of total length vA/w.

To compute tone accurately, it is also important to take into account the overlap between strokes that cross each other. Assuming uncorrelated overlap between the strokes of each non-overlapping set seems to work well. For example, for bidirectional hatching, suppose that the strokes in each hatching direction deposit a quantity of ink x on the paper. Then the total ink in the cross-hatched area is given by the sum of the ink from the strokes in each direction $2x$, minus the quantity of ink that is deposited in the same place twice, which, assuming uncorrelated strokes, can be approximated by x^2. Thus, in order to achieve a value v, we use enough strokes in each direction to achieve a value x that satisfies the quadratic equation $2x - x^2 = v$. The correct level for each set is therefore given by $x = 1 - \sqrt{1 - v}$. This expression is used, for example, by the "cross-hatching" texture to achieve the right tone when two different directions of hatching are crossed (Figure 2).

A.3.3 Stroke textures

To draw more complex textures, we use procedural prioritized stroke textures, as described in Section 4. In this section, we examine the "brick" texture in some detail. The description in this section can readily be extended to other types of stroke textures.

The "brick" texture builds tone out of three sets of strokes: the brick outlines; shading strokes within the bricks; and hatching strokes layered over the whole surface. Each set of strokes is associated with a different *InkPen*.

The rendering process for "brick" textures is summarized below:

procedure *RenderBrickTexture*(*TargetTone*, *Polygon3D*)
  Layout ← *GenerateBricks*(*Polygon3D*, *BrickSizes*)
  for each brick $B \in$ *Layout* **do**
    DrawBrickOutline(B, *TargetTone*, *ViewPoint*, *Lights*)
    if the tone of B is too light **then**
      ShadeWithinBrick(B, *TargetTone*)
    end if
  end for
  if the overall tone is still too light **then**
    HatchOver(*Layout*, *TargetTone*)
  end if
end procedure

The brick outlines are generated from a simple layout that is computed on the fly in texture space and then projected into device space before drawing takes place.

The width of the nib used to draw the outline strokes is scaled according to the tone being rendered: darker tones use the default nib size; for lighter tones, the nib size is scaled down. The same nib size is used for all the bricks in order to achieve a consistent result over the entire surface. The darkness of each stroke is then adjusted so as to take the BRDF and shadow edge effects into account, as described in Section 4.3. The darkness of the strokes may be further reduced, for example, when the area of the bricks becomes smaller due to perspective forshortening.

Let T_o be the tone created by the outline strokes for a given brick B. We estimate T_o by taking the sum of the amount of ink deposited by each outline stroke and dividing it by the area covered by the brick on the paper, as discussed in Appendix A.3.2. If T_o is not dark enough to achieve the desired tone, then the interior of the brick is shaded. The darkness of each brick is limited by a constant T_s. Rather than shading every brick with the same tone $T = \textit{TargetTone} - T_o$, we instead shade each brick with probability $\min\{1, T/T_s\}$. If the brick is shaded, enough strokes are used to achieve a tone T_s, with some randomness added.

If the shaded bricks still do not achieve the target tone, that is, if $T_o + T_s < \textit{TargetTone}$, then additional hatching is used over the top of all the bricks. For these hatch lines, we use the method described in Appendix A.3.2 to take into account the overlap between strokes.

A.3.4 Clipping strokes

The strokes must be clipped to the visible regions they texture. The 2D BSP tree data structure is used for this purpose. The path of each stroke is "pushed" down the BSP tree until it reaches one or more leaf nodes. Only the portions of the path that reach nodes belonging to the region being textured are rendered. Other clipping conditions can also be used. For instance, hatching shadow areas is handled by generating strokes over the entire surface and clipping them to the regions in shadow.

A.3.5 Constructing the outline strokes

The outlines of visible surfaces are extracted by traversing the set of edges stored in the planar map. As described in Section 4.3, these outline edges are rendered only when the tones on either side of the edge are very similar, and when the edges themselves are not directly illuminated by the light source. Testing the tones of adjacent faces is easily accomplished by searching in the planar map.