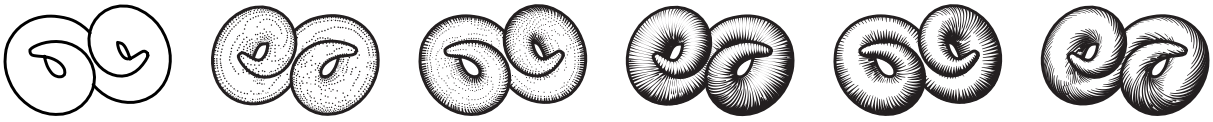
EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

High Quality Hatching

Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte

Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg, Germany
jzander@cs.uni-magdeburg.de, {isenberg|stefans|tstr}@isg.cs.uni-magdeburg.de



Abstract

Hatching lines are often used in line illustrations to convey tone and texture of a surface. In this paper we present methods to generate hatching lines from polygonal meshes and render them in high quality either at interactive rates for on-screen display or for reproduction in print. Our approach is based on local curvature information that is integrated to form streamlines on the surface of the mesh. We use a new algorithm that provides an even distribution of these lines. A special processing of these streamlines ensures high quality line rendering for both intended output media later on. While the streamlines are generated in a preprocessing stage, hatching lines are rendered either for vector-based printer output or on-screen display, the latter allowing for interaction in terms of changing the view parameters or manipulating the entire line shading model at run-time using a virtual machine.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

Keywords: non-photorealistic rendering, high quality hatching, line rendering, line shading

1. Introduction

The area of non-photorealistic rendering (NPR) has become a rapidly growing field in computer graphics over the last two decades. The main goal behind NPR is to enrich the expressiveness of computer graphics techniques by generating synthetic images that embody qualities of hand-drawn imagery. One of the techniques receiving high interest is the creation of line drawings. Here, two classes of lines need to be generated and, thus, distinguished. First, there is the outline or silhouette that closes an object and segregates it from the surrounding. Second, there are hatching lines that collectively convey tone as well as texture of an object's surface.

In this paper we present a way to generate and render hatching lines based on a three-dimensional polygonal mesh (in particular, we use triangle meshes). Most approaches for line rendering today aim for a fast generation and rendering of lines possibly exploiting capabilities of graphics hardware. These methods compute the lines on the surface of the model in order to avoid artifacts such as incoherence and shower-door-effect but keep the speed. However, the lines are finally output in pixel images yielding sampling artifacts and reducing the quality of the images. Other techniques generate the lines after the model has been projected into 2D which resembles the traditional way of generating line drawings, for example, in engravings, copper plates, or pen-and-ink drawings.

Our method aims for the generation of vector oriented hatching in order to yield line renditions with a higher quality. Since we compute the lines directly in 3D, additional procedures are needed to achieve high quality, rendering speed, and aesthetic appeal of the resulting images. Our approach not only allows us to reproduce images in an appropriate quality for printing. We also give the designer of the drawings the opportunity to interactively work with the rendition when creating it. This comprises being able to manipulate the view on the model as well as adapting parameters of the hatching process including the shading model.

Our method does therefore not achieve the frame-rates of the aforementioned real-time hardware shading techniques but does, nonetheless, allow for interactive rendering. At the same time we also have the possibility to create versions of the rendition adapted to the desired reproduction quality. We use OpenGL lines (or similar technologies) for rendering on screen but change to a vector based description for high quality output.

The main contributions of this paper are therefore:

- object-space generation of streamlines to allow for generation of hatching illustrations with interactive frame-rates as well as off-line,
- a new algorithm to achieve an even distribution of the object-space streamlines computed from a polygonal mesh,
- the use of a virtual machine to replace the formulas of the line shading model at run-time allowing for changes that go beyond a mere parameter adjustment, and
- techniques to render hatching lines in high quality by processing the streamlines specifically with respect to print media, in particular to requirements resulting from using monochrome ink.

The paper is organized as follows. Section 2 describes related work in the field of computer generated hatching lines. In Section 3 we give an outline of our algorithm to create the lines and describe the main steps in detail. Section 4 is devoted to the actual rendering, i. e., the adaptive visualization of the lines. We discuss a number of examples in Section 5 and finally conclude the paper with some ideas for future work.

2. Related Work

Traditional line renditions that employ hatching are quite commonly used in arts and for illustration purposes. Hatching lines fill an area of an image by collectively conveying texture and tone. While these lines may vary in their length, they typically follow some geometric features of the object being depicted and they may be layered to produce cross-hatching. Considering hand-made drawings, hatching is not a drawing style or technique in itself, instead it is used and achieved with several different techniques. Therefore, a wealth of methods have been developed also in computer graphics to achieve the same or similar effects.

In the first of a series of papers, SALISBURY et al. show how to use *stroke textures* to convey a certain darkness for shading with pen-and-ink lines [SABS94]. When interactively drawing an illustration, the algorithms selects strokes from the stroke texture until a desired shading has been achieved. WINKENBACH and SALESIN introduce the concept of *prioritized stroke textures* [WS94]. This allows the resolution dependent placement of pre-recorded strokes to achieve the same perceived grey value. If the resolution changes, their method places more or fewer strokes until the desired tone is achieved yielding a rendition that is appropriate for the specific resolution. SALISBURY et al. further discuss the scale-dependence of pen-and-ink drawings in terms of perceived darkness of the hatched areas [SALS96]. In particular, they demonstrate how to maintain sharp discontinuities of the textures across various resolution levels.

Being the first to create hatching renditions from 3D scenes, LEISTER bases his approach on modified ray tracing [Lei94]. He uses an additional direction that is defined for each object's surface and a parameter to determine the distance between two hatches, similar to the *u*-*v* parameter field defined for texturing. Being a ray tracing adaption, his method can emulate reflections and refractions. The algorithms produces image-space results which means that they cannot easily be processed any further using stroke manipulations. In addition, the appearance of the created images is rather clean and artificial.

PNUELI and BRUCKSTEIN present their **Dig$^{\mathbf{i}}$***Dürer* system that uses greyscale images as input and creates a halftoned output image that resembles the style of engravings [PB94]. It computes level contours of a potential field and is based on a curve evolution algorithm that controls the density of line elements.

WINKENBACH and SALESIN introduce a technique for generating hatching renditions from parametric surfaces by employing isoparametric curves [WS96]. They use prioritized stroke textures and align the strokes according to these curves. They achieve a quite natural look by using long and short strokes as well as adding small alterations to the strokes. Additionally they used randomized dots on lines to stipple an area with a desired tone. SALISBURY et al. use a 2D greyscale image as input and require the user to specify a direction field as well as example strokes [SWHS97]. Their system then generates hatch line textures that reproduce the shading of the original image while conveying the impression to be attached to the surface of the object and following its features.

In a completely image-based approach, OSTROMOUKHOV presents an algorithm that uses a 2D source image and so called engraving layers—basic dither screens that are combined to form hatching specific dither patterns [Ost99]. He requires user-interaction in order to specify how these layers have to be deformed by image warping in order to follow certain features of the image. A screening process computes the final rendition, that possesses a very clean and artificial appearance.

In an object-space approach, DEUSSEN et al. use internal skeletons created from triangle meshes using progressive meshes [DHR*99]. The skeletons are used to determine a direction perpendicular to which the object is sliced. The slice curves are then used as hatching lines for the objects again producing a very clean and artificial appearance. By adding line styles and thus changing the appearance of the hatch-

ing lines based on shading or other geometric information, a more natural look can be achieved.

The technique presented by RÖSSL and KOBBELT works in image-space and also uses triangle meshes [RK00]. They first compute an approximation of curvature directions and normals for each vertex and do a linear interpolation for the values on the faces. They then render G-buffers for both normal and curvature direction vectors. Afterwards, they use streamlines for following the hatching lines in 2D. Interaction is required for specifying homogeneous parts of curvature directions as well as reference lines in the projection. Although the shading they used in the given examples could be improved, they achieve a fairly natural look of the images.

HERTZMANN and ZORIN also work in image-space and base their method on smooth surfaces given by a polygonal control mesh [HZ00]. Similar to the previously discussed approach, they also use approximated principal curvature lines in 3D that are projected into image-space. They add some preprocessing of the direction field in order to avoid artifacts in the hatching lines and also create a fairly natural look of the renditions.

There are a number of approaches that generate lines on 3D shapes for rendering based on some features of the model. For example, in [ARS79], [Elb95b], [Elb95a], and [Elb98] the generation of isoparametric lines on freeform surfaces is discussed and it is demonstrated how to enhance them with, e. g., line haloes. ELBER [Elb99] and RÖSSL et al. [RKS00] discuss how to use lines on the surface of objects to visualize vector or curvature direction fields.

INTERRANTE uses principal curvature directions for visualizations of volumetric data using lines on the surface [Int97]. In a subsequent paper [GIHL00], GIRSHICK et al. discuss the use of principal curvature directions for 3D line drawings in general. They state that there are, for example, psychological reasons for employing principal curvature lines in order to enhance shape recognition where, e. g., silhouette lines are not enough. However, the example renditions they present, both generated from volumetric and polygonal data, do not resemble what is traditionally considered to be hatching style. Also using principal curvatures for line orientation, DONG et al. apply the generation of hatching lines that are computed for volumetric data to the area of medical illustration [DCLK03]. By taking the local characteristics of the volume data into account they are able to improve the quality of the rendition.

Summarizing these findings, we come to the conclusion that we can successfully use principal directions for the generation of hatching lines. Principal directions are defined everywhere on a surface except in isolated singularities, they are not dependent on a parameterization of the surface and they are known to convey the form of an object to the viewer. Our observations have also shown that ray-tracing, semi-automatic methods with manually fitted parametric curves, and skeleton approaches produce very sterile lines. The use of principal curvatures gives the possibility to create a more hand-crafted appearance if we compare the results of the aforementioned techniques to hand-made engravings. However, a post processing of the direction field obtained from principal direction vectors is necessary in order to avoid too many distracting details and to generate a more homogeneous field.

3. Algorithm Overview

Our algorithm to produce high quality hatching uses triangular meshes as input. The whole process comprises two stages: a preprocessing phase where lines are computed in 3D and a rendering phase where these lines are visualized. In the first stage, we start with the generation of a direction field based on curvature information. We then process the curvature field in order to enhance its quality before 3D streamlines are generated. These streamlines are the input to the second stage where they are rendered according to the desired output device. In the following we will describe each of these steps in more detail.

3.1. Generation of a Curvature Field

Hatching lines, as already stated above, follow some geometric feature of an object. To achieve this, the first step in the preprocessing phase is to establish direction information for the lines. We therefore generate a direction field, consisting of a unit direction vector for each vertex of the model, laying in the appropriate tangent plane. As has been argued before, e. g., by INTERRANTE [Int97], GIRSHICK et al. [GIHL00], and HERTZMANN and ZORIN [HZ00], principal curvature values are well suited. Indeed, it has been found that in traditional illustrations hatching lines are frequently used to emphasize curvature. Therefore, we approximate the principal curvature directions using a method introduced by RÖSSL AND KOBBELT [RK99] and store these values in the direction field. For each vertex, there are two possible vectors following the maximal and minimal curvature direction κ_1 and κ_2 as can be seen in Figure 1. The curvature κ_i with the higher absolute value (indicated by the sign of the mean curvature $\frac{1}{2}(\kappa_1 + \kappa_2)$) determines the more curved direction which will then be used. In most cases this is a good heuristic to emphasis cylindrical structures.

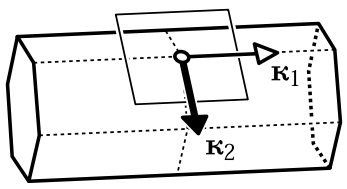


Figure 1: *Approximated principal curvature directions for a vertex of the polygonal model. In this case the direction of κ_2 is chosen in order to hatch around the cylinder's circumference.*

3.2. Processing of the Curvature Field

The quality of the resulting direction field directly depends on the underlying algorithm for curvature computation (see GOLDFEATHER [Gol01]) and on the properties of the mesh. Noise and high levels of detail easily introduce unwanted artifacts. To avoid these, a simplification algorithm can be applied to the mesh (see PRAUN et al. [PHWF01]) or the mesh can be smoothed before curvature is computed. However, no reliable curvature information can be extracted from flat or spherical surfaces. Therefore, too smooth surfaces tend to result in poorly aligned direction vectors.

Since the resulting direction field relies solely on local curvature information, is not very homogeneous. To improve this, we have gone a similar route as HERTZMANN and ZORIN [HZ00]: an energy term is defined that measures the deviation of a direction vector to the ones located on its incident edges. In contrast to HERTZMANN and ZORIN [HZ00] this is done for 180° and not 90° symmetries. Our algorithm only wants to wrap up the surface with evenly distributed parallel lines. Cross-hatching is then achieved by repeating the process with rotated lines (see Figure 10 for an example). This allows to generate cross-hatchings with arbitrary angles whereas Hertzmann et al. only generate orthogonally crossing lines. To achieve almost parallel lines, directions within a homogeneous neighborhood are used as a basis for fitting the other directions using a global non-linear optimization technique (cf. [HZ00]), which is applied on all regions which do not satisfy a user selectable level of homogeneity.

3.3. Generation of 3D Streamlines

After a smooth direction field has been computed, streamlines are generated by integrating the direction vector field on the surface of the model. In contrast to the technique suggested in [ACSD*03] that solves the problem in 2D, we employ a version of the original algorithm presented in [JL97] and adapt it for the third dimension. Therefor it is necessary to determine direction vectors not only at the individual vertices, but for every point on the surface. This is done by a spherical-linear interpolation of the respective direction vectors using the *barycentric coordinates* of a position as weights. The seeding strategy was also modified from the original paper. Since it is not always possible to reach all parts of a model from a single initial seed point, each face centroid is used as a possible seeding point. Starting from there, the algorithm tries to reach as much area as possible. While a streamline is growing through the direction vector field, new possible seed points are generated alongside. Only if none of these can be used to create a new streamline, the next face centroid is used as a new possible seeding point.

To prevent line crossings, streamlines are only integrated until they meet each other. Instead of the grid-based streamline-proximity scheme, cylinders are used to compute streamline distances and to end a streamline if it closes in on another one (see Figure 2). The cylinders are generated along the growing directions of the streamlines and are stored in all the faces they intersect. This helps to prevent streamlines from influencing each other on opposite sides of thin regions in the mesh.

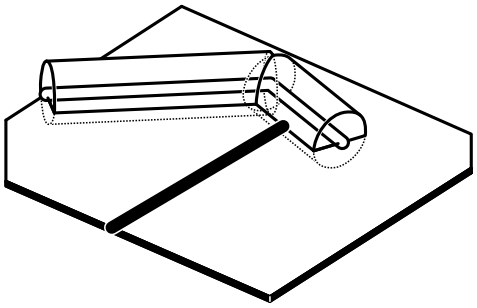


Figure 2: *A new streamline (black) is terminated at the cylinder surrounding a previously computed streamline (white). The endpoint of the new streamline is kept on a distance which equals the cylinder's radius.*

In order to find the distance, a lookup is done for all cylinders that are stored in the according face and the nearest intersection in the growing direction of a streamline is computed. As long as the size of the faces is less or roughly equal to the intended distance between two streamlines, this is very fast because, in average, there is only one cylinder per face. The computational overhead only grows for large faces. Using cylinders has another advantage. Streamlines are allowed to come very close with their tips, removing wide gaps that otherwise tend to occur (see Figure 3).

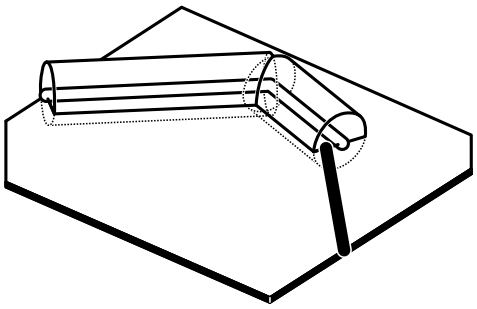


Figure 3: *The use of cylinders allows a new streamline (black) to come closer to an existing streamline (white) than the cylinder's radius, if it approaches its tip.*

A property that is needed later is that the streamlines have to follow the surface of the mesh closely. So while integrating the direction field, new points are inserted each time an edge is crossed, moving from one face to the next. This ensures that even with a wide step size, streamlines will not poke through the surface. The quality of the integrator also needs further observation. First, we used a very simplistic

Euler integrator, but changing to a fourth order Runge-Kutta allowed a wider step-size. This drops the number of segments that are needed to represent the streamlines. Nonetheless, the quality continues to be very high, resulting in decreased time needed for the preprocessing stage.

3.4. Line Tapering

In the final phase of the preprocessing stage data for line tapering is gathered. The process differs slightly from [JL97] since we use a different distance metric for streamline collisions. In our implementation it is possible that streamlines can get very close with their tips. This effect should not be impaired by line tapering. So closeness of nearby streamlines is not measured omni-directional, but parallel to the direction of the streamline (see Figure 4). This aligns with the streamline generation process using cylinders as described above. For each point on a streamline, first the local average direction is calculated. This is then used in combination with the normal of the face below the point to construct a plane orthogonal to that direction. To measure the distance of nearby streamlines the intersection points of all nearby streamlines with this plane are computed and the minimal distance is stored. Finally, this minimal distance is employed to calculate the tapering value that is needed in the rendering stage as a multiplicative modulator for the line width.

Figure 4: *Computing streamline distances omni-directionally results in gaps between the tips of nearby streamlines as can be seen on the left. Using our approach these gaps are minimized resulting in the image on the right.*

4. Rendering in High Quality

After the preprocessing stage has been completed, the second phase begins—visualizing the streamlines as hatching lines. The major goal in this step is to render the streamlines with high quality in order to be able to use the generated images, for example, for reproduction in print media. Therefore, three goals have to be accomplished:

- a fast and effective line processing including hidden line removal (HLR) that removes the silhouette strokes and streamlines not visible from the given viewpoint while still offering interactive frame-rates,
- an appropriate NPR line shading for the hatching strokes in order to produce a smooth transition between fully drawn and not drawn lines according to the specific lighting condition, and
- a high quality line output that produces vector-oriented data for reproduction including, for example, the transformation of a shaded line into a monochrome representation.

4.1. HLR and Line Processing

The generated streamlines are inserted into a stroke rendering pipeline that first uses a hybrid hidden line removal algorithm (we employ the z-buffer based method presented by ISENBERG et al. [IHS02]) in order to remove the occluded parts. The mentioned HLR algorithm was originally conceived to clip object silhouettes, but it works very well under these new requirements. There are only minor artifacts at the object's silhouette due to z-buffer imprecisions as shown in Figure 5. This can be resolved by simply removing all strokes on backfaces. A welcome side-effect of this procedure was a noticeable frame-rate improvement because the backfacing test is cheaper than feeding segments through the HLR algorithm.

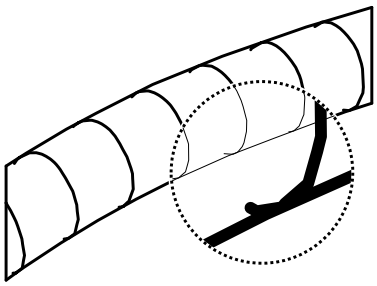


Figure 5: *If only the z-buffer based HLR scheme is used, small tickmarks will appear near the silhouette. These have to be removed.*

Using a hybrid HLR scheme also enables us to achieve interactive frame-rates when rendering the images. This results from computing the streamlines in the preprocessing stage off-line and only performing the HLR test at run-time. Being able to render the images at interactive rates is very important for illustration designers to directly see the effects of changes they made.

In addition, we do not use advanced line stylization with the generated hatching and silhouette lines. In general, for producing hatching renditions it is not necessary to follow a style based approach as discussed by various authors (see, e. g., NORTHRUP and MARKOSIAN [NM00] or ISENBERG et al. [IHS02]). In contrast, when examining cross-hatched images created by artists one finds that they only consist of monochrome lines. Thus, advanced stylization that introduces structure to the lines (as, e. g., by using textures or

applying path variations) is not necessary in this case. This has many advantages, for example, faster rendering and less artifacts. In particular, we achieve frame-coherency since the hatching lines are attached to the surface of the objects. In addition, there are no artifacts created by the stylization when it comes to generating high quality output.

4.2. Line Shading

In order to limit the number of lines that are drawn and to convey a specific illumination condition it is now necessary to apply line shading. We opted for a rich NPR centric shading model that features effects like rim shading or curvature lighting (see, e. g., STROTHOTTE and SCHLECHTWEG [SS02]). To provide more flexibility to the user for adapting this shading model and to allow easier experiments with different effects, it was not implemented as a "hardwired" formula. Instead, it is generated by a virtual machine (bytecode interpreter). This is a very elegant way of allowing the user to make changes to the expressions and by that to control the two main parameters: stroke width and stroke density. These changes are possible at run-time and avoid tedious recompiles. At run-time, the formulas are being compiled into byte code that is executed by a small stack-based interpreter in order to obtain the shading values. This is achieved by parsing the expression and breaking it down into constants, symbols, and operators. The whole expression is converted into *Reverse Polish Notation* in order to simplify the evaluation of the term because in this representation brackets and operator precedence are of no importance. Later, the term is evaluated for each vertex of a streamline and symbols are substituted with their associated value which changes depending on view direction or local surface orientation. The speed impact of the interpretation is negligible.

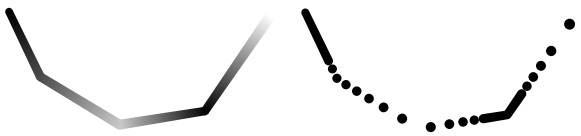
4.3. Line Output

When closely examining real hatched drawings, we found that monochrome lines are used with the two properties: variable width and the possibility to dissolve solid lines into dots. Therefore, we have two parameters—line width and line density. Because speed is not an issue when creating output for high quality reproduction, we made sure that line caps and bends are round. Another noteworthy feature is the possibility to use negative values for the line width. This helps to seamlessly blend out strokes because visible line tips can now appear in the middle of a stroke segment. This decouples the visual occurrence of a line end from the positions of the individual stroke vertices. This means that a line may now also terminate in the middle of a stroke segment (see lower two examples in Figure 6).

The use of dotted lines in hand-made hatching is caused by the limitation to monochrome primitives. In addition, it is also in part caused by restrictions of reproduction techniques that can only handle monochrome ink. When trying

Figure 6: *Decreasing the width of the line first results in tapering and later also in shortening the visible portion when the width gets negative.*

to reproduce grey values, these would be dithered into black and white pixels which would destroy the appearance of a hatching (see Figure 7(a)). In order to avoid this to occur for shaded hatching lines, we also represent line density by a form of one-dimensional stippling (see Figure 7(b)). This one-dimensional line stippling is, therefore, not intended or used to simulate regular two-dimensional stippling as done in previous approaches. Instead, we use it as an alternative to change the tone of a line, which does not rely on line width. To accomplish this the line is broken down into a string of dots that are placed to locally approximate the required density of the specific stroke. The distance of two dots is chosen in a way to match the ratio of black and white space inside the quadrilateral spanned through their centers to the average density value for that region. This can be further controlled by specifying a minimum distance between dots. In order to draw segments in full color where the computed distance would lead to distances smaller than desired, the segment is subdivided at places where exactly this minimum distance is reached.



(a) Desired density of the stroke.

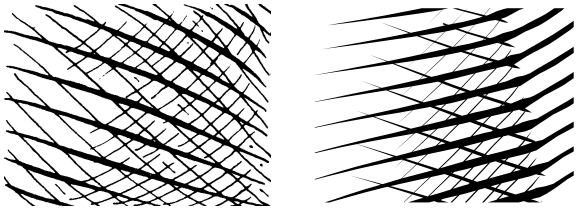
(b) Approximation of the line density using dots.

Figure 7: *Achieving a desired line density using line halftoning using one-dimensional stippling along the stroke.*

We implemented two specific renderers with respect to the two different goals for the images to be created. The first is able to output lines and arcs to create the final rendition as PDF files for high resolution and high quality reproduction. The second is for WYSIWYG-purposes and supports designers that want to interactively create hatching illustrations. This second renderer triangulates the strokes into triangle strips that can afterwards be rendered at interactive frame-rates using OpenGL.

In addition to producing single hatched images, it is also possible to combine multiple rendering passes. This is frequently used in hand-made images as shown in Figure 8(a).

It is easily possible to generate cross-hatching effects by rotating the whole direction field around arbitrary angles and compositing several layers of differently shaded strokes (see Figure 8(b)).

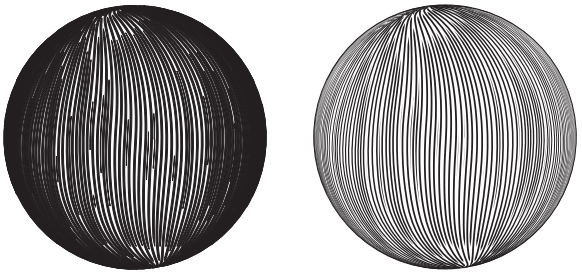


(a) Closeup of a real artistic rendition (from [Hod89]).

(b) Three layers of strokes resulting from a rotated direction field.

Figure 8: *Comparison of hand-made hatching with a magnified section of a computer-generated image (Figure 8(b) is a detail from Figure 10(b)).*

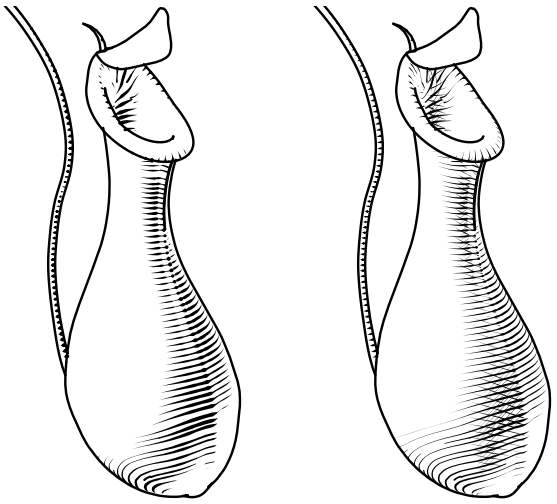
As we have outlined above, the hatching lines are generated on the object's surface and, thus, in object-space. A disadvantage of working in object-space that has to be considered for high-quality rendering is that shading not only depends on the width of the lines but also on their distance to each other after the projection to screen space. If the lines get closer to each other, the perceived brightness of these regions will be darker (see Figure 9(a)). This is particularly noticeable close to the silhouette or if the scene is scaled. Solutions have been thoroughly discussed in, for example, [SABS94], [WS94], and [SALS96]. We determine a correction factor for each vertex of a stroke. Specifically, a normalized vector perpendicular to the line direction and the surface normal is computed using the cross product. This is a local estimate of the distance to imaginary nearby hatching lines. The length of the projection of this vector onto the viewplane is measured and used as the correction factor for the line width. It reduces the unwanted effect of differences in the perceived brightness of strokes with different distances to each other (see Figure 9(b)). Therefore, the line shading can now be determined depending entirely on the parameterized NPR shading model.



(a) Without a correction factor.

(b) With the correction factor.

Figure 9: *Use of a correction factor that compensates differences in the perceived brightness. Unfortunately, the printing process may weaken the correction due to, for example, a minimum possible line width or effects related to dot gain.*



(a) Single hatching.

(b) Triple hatching.

Figure 10: *Single hatching and triple hatching of the model of a tropical pitcher plant's trap. Note that the hatching is able to effectively convey the small ridge that runs at the front side of the pitcher. This is done by varying line width or by employing triple hatching.*

5. Examples

Now we will give a number of specific examples along with descriptions of how the effects in the images were achieved.

A first example compares single and multiply hatched illustrations of the same object. Figure 10(a) shows the illustration of a tropical pitcher plant's trap with only single hatching applied. Note that small details such as the small ridge that runs at the front side of the pitcher are still clearly visible because they are emphasized through the line shading model. In the second version of the same object in Figure 10(b), three layers of hatching have been combined and emphasize the details even more.

Instead of just using black and white, background and foreground colors can be changed in order to create a composed rendition. Figure 11 shows an example where several layers of hatching in different colors have been composited along with certain background colors. This method can easily be applied to create illustrations for use in colored printing. If spot colors are used, no unwanted dithering artifacts are created when the rendition is reproduced and printing cost can often be lowered in contrast to the usual four color printing process.

Figure 12 demonstrates that line stippling alone can be used to convey shading information. Therefore, the line

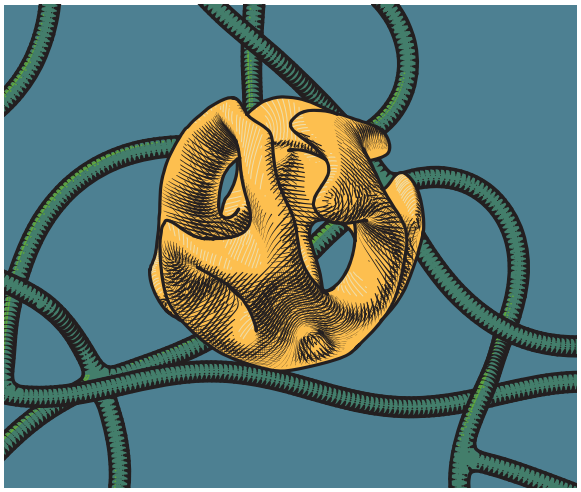


Figure 11: *Compositing a number of hatching layers in different colors and background colors in order to create a multi-color illustration (five colors and black).*

width has been set to constant in the example. This way not only shading information is transported but also the form of the individual leaves is accentuated, which would be not as clear with pure two-dimensional stippling.

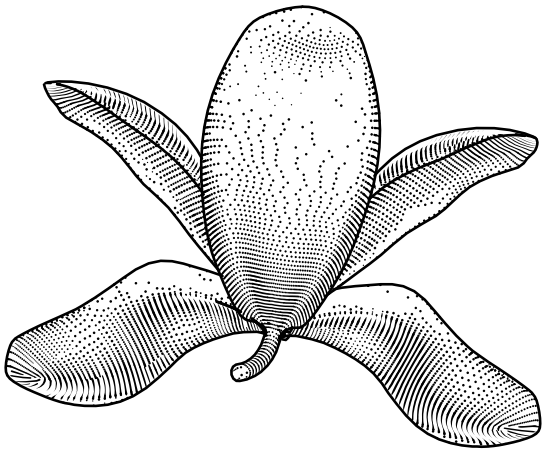


Figure 12: *Orchid-backside with quasi constant line width. Shading has been achieved by varying line density.*

Figure 13 demonstrates the effect of varying line widths. In Figure 13(a), the line width is used to convey shading information and the line ends are rounded without using separate dots. In contrast, Figure 13(b) shows the use of sharp line tips while the individual lines are placed closer to each other. Making use of cross-hatching, the statue in Figure 13(c) is rendered using lines that are dissolved into individual dots in order to show the shading effects. Finally, Figure 13(d) makes use of fairly wide lines that overlap each other. This produces a very sharp contrast between shaded and lit areas.

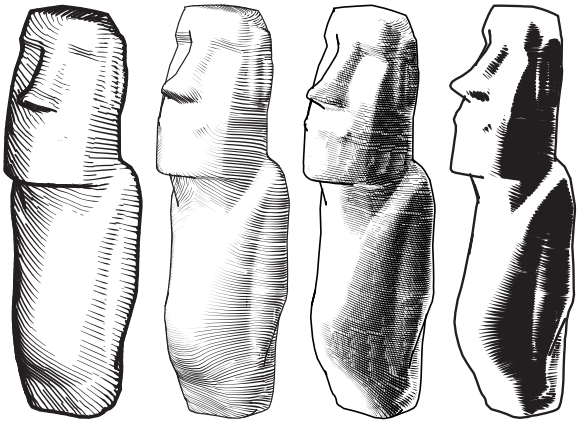


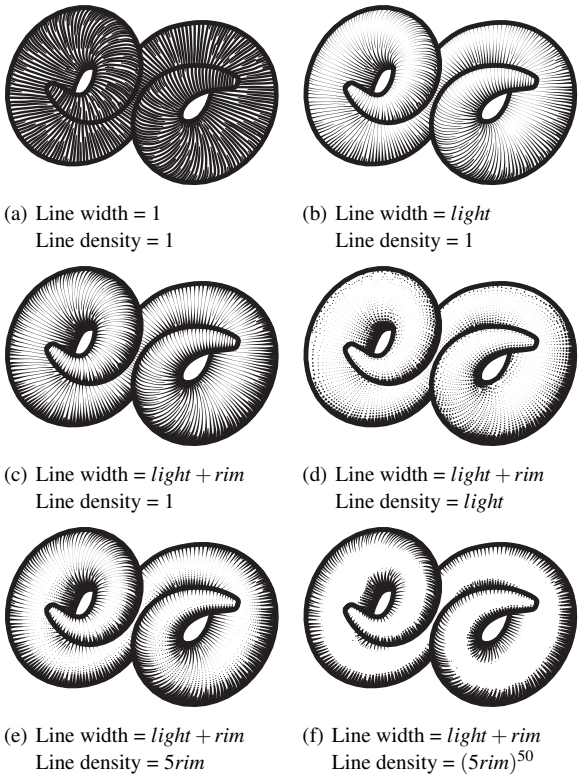
Figure 13: *Different effects that can be achieved by varying the number and the distance between neighboring hatching lines.*

Figure 14 illustrates how the appearance of an object changes while the user adapts the two shading formulas at run-time. In Figure 14(a) the raw strokes can be seen with constant terms for line width and density. This results in a quasi-constant tone. The use of a single light source as a modulator for the linewidth adds depth as seen in Figure 14(b). To enhance the edges, rim shadow lighting has been added in Figure 14(c). Using the light to also modulate the line density as seen in Figure 14(d) creates a fairly contrastless halftoning-like effect. If the density term is further changed this can be constrained to a rather small band (Figure 14(e)). The addition increase of the contrast may be used to eliminate lines and as a means to place highlights (Figure 14(f)).

6. Conclusion

In this paper we have presented a hatching method for generating high quality vector-oriented images mainly aimed at reproduction in print media. Our technique computes the hatching lines in object-space in a pre-processing step which gets rid of frame-incoherences and the shower-door effect as well as allows for fast rendering of the lines at run-time. Therefore, we can offer interactive frame-rates for designing the hatching illustration using a hybrid HLR technique. We have shown that the illumination model can be modified at run-time giving the designer a great freedom of expression. Using the examples shown in the paper, we demonstrated that this allows a wide range of different effects that can be achieved by parameterizing this model.

We have demonstrated how to solve a variety of issues that may occur when attempting high-quality reproduction



(a) Line width = 1
Line density = 1

(b) Line width = *light*
Line density = 1

(c) Line width = $light + rim$
Line density = 1

(d) Line width = $light + rim$
Line density = *light*

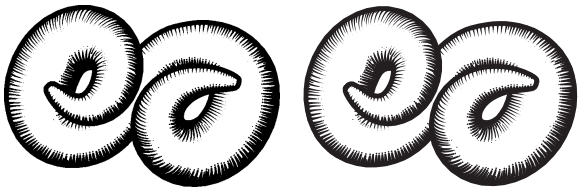
(e) Line width = $light + rim$
Line density = $5rim$

(f) Line width = $light + rim$
Line density = $(5rim)^{50}$

Figure 14: *The process of altering the two line shading formulas until a desirable result is achieved.*

in print. Those are, in particular, the possibility to use negative line widths to allow for a more flexible line tapering, the use of dotted lines in order to avoid the dithering effects from line shading, the removal of small artifacts that result from HLR, and employing a correction factor for achieving an approximately equal perceived grey value across the object to account for different viewing angles on the surface.

In addition, we provide two renderers for these two goals for the presentation of the hatching illustration. The first is designed for interactive on-screen display (see Figure 15(a)) while the second aims at generating the high-quality renditions for reproduction in print media (see Figure 15(b)).



(a) On-screen rendition.

(b) Vector graphic output.

Figure 15: *The two output media used compared on the basis of Figure 14(f).*

For future work we plan to further improve the computation of the streamlines in order to create a more aesthetic line placement. An hierarchic construction-order should get rid of some of the more striking placement artifacts and at the same time keep the uniformity of line distances after projecting to screen-space. In addition, a downside of the on-screen renderer is the lack of anti-aliasing, which is a serious disadvantage because line art needs good anti-aliasing for lower resolutions. However, on newer graphics hardware it is possible to use full scene anti-aliasing but the resulting frame-rate drop is severe in our implementation. This will be improved in future versions of the software so that the on-screen version better resembles the printed version of the line drawing. We also envision an easier and more intuitive interface to adjust the settings of the illumination model so that non-technical people (e. g., designers) can easier adjust the parameters that influence the appearance of the rendering. Improvements of the interface will also include possibilities for creating composed drawings in one step rather than compositing them afterwards. Finally, we want to integrate ways that let the system suggest additional cross-hatching directions automatically rather than specifying them manually.

Acknowledgments

We would like to thank Petra Neumann for the model of the tropical pitcher plant's trap in Figure 10. Specifically, the model represents the upper trap of a *Nepenthes alata* plant.

References

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic Polygonal Remeshing. *ACM Transactions on Graphics 22*, 3 (July 2003), 485–493.

[ARS79] APPEL A., ROHLF F. J., STEIN A. J.: The Haloed Line Effect for Hidden Line Elimination. In *Proceedings of SIGGRAPH'79* (New York, 1979), ACM Press, pp. 151–157.

[DCLK03] DONG F., CLAPWORTHY G. J., LIN H., KROKOS M. A.: Nonphotorealistic Rendering of Medical Volume Data. *IEEE Computer Graphics and Applications 23*, 4 (July/Aug. 2003), 44–52.

[DHR*99] DEUSSEN O., HAMEL J., RAAB A., SCHLECHTWEG S., STROTHOTTE T.: An Illustration Technique Using Hardware-Based Intersections and Skeletons. In *Proceedings of Graphics Interface'99* (1999), Morgan Kaufmann Publishers Inc., pp. 175–182.

[Elb95a] ELBER G.: Line Art Rendering via a Coverage of Isoparametric Curves. *IEEE Transactions on Visualization and Computer Graphics 1*, 3 (Sept. 1995), 231–239.

[Elb95b] ELBER G.: Line Illustrations ∈ Computer Graphics. *The Visual Computer 11*, 6 (June 1995), 290–296.

[Elb98] ELBER G.: Line Art Illustrations of Parametric and Implicit Forms. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan. 1998), 71–81.

[Elb99] ELBER G.: Interactive Line Art Rendering of Freeform Surfaces. *Computer Graphics Forum 18*, 3 (Sept. 1999), 1–12.

[GIHL00] GIRSHICK A., INTERRANTE V., HAKER S., LEMOINE T.: Line Direction Matters: An Argument for the Use of Principal Directions in 3D Line Drawings. In *Proceedings of NPAR 2000* (New York, 2000), ACM Press, pp. 43–52.

[Gol01] GOLDFEATHER J.: *Understanding Errors in Approximating Principal Direction Vectors*. Tech. Rep. 01-006, University of Minnesota – Computer Science and Engineering, 2001.

[Hod89] HODGES E. R. S. (Ed.): *The Guild Handbook of Scientific Illustration*. Van Nostrand Reinhold, New York, 1989.

[HZ00] HERTZMANN A., ZORIN D.: Illustrating Smooth Surfaces. In *Proceedings of SIGGRAPH 2000* (New York, 2000), ACM Press, pp. 517–526.

[IHS02] ISENBERG T., HALPER N., STROTHOTTE T.: Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum 21*, 3 (Sept. 2002), 249–258.

[Int97] INTERRANTE V.: Illustrating Surface Shape in Volume Data Via Principal Direction-Driven 3D Line Integral Convolution. In *Proceedings of SIGGRAPH'97* (New York, 1997), ACM Press, pp. 109–116.

[JL97] JOBARD B., LEFER W.: Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing* (1997), pp. 45–55.

[Lei94] LEISTER W.: Computer Generated Copper Plates. *Computer Graphics Forum 13*, 1 (Mar. 1994), 69–77.

[NM00] NORTHRUP J. D., MARKOSIAN L.: Artistic Silhouettes: A Hybrid Approach. In *Proceedings of NPAR 2000* (New York, 2000), ACM Press, pp. 31–37.

[Ost99] OSTROMOUKHOV V.: Digital Facial Engraving. In *Proceedings of SIGGRAPH'99* (New York, 1999), ACM Press, pp. 417–424.

[PB94] PNUELI Y., BRUCKSTEIN A. M.: **Dig$^{\mathbf{i}}$***Dürer* – A Digital Engraving System. *The Visual Computer 10*, 5 (Apr. 1994), 277–292.

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-Time Hatching. In *Proceedings of SIGGRAPH 2001* (New York, 2001), ACM Press, pp. 581–586.

[RK99] RÖSSL C., KOBBELT L.: Approximation and Visualization of Discrete Curvature on Triangulated Surfaces. In *Vision, Modeling, and Visualization (VMV) '99 Proceedings* (St. Augustin, Germany, 1999), infix, pp. 339–346.

[RK00] RÖSSL C., KOBBELT L.: Line-Art Rendering of 3D-Models. In *Proceedings of Pacific Graphics 2000* (Los Alamitos, CA, 2000), IEEE Computer Society Press, pp. 87–96.

[RKS00] RÖSSL C., KOBBELT L., SEIDEL H.-P.: Line Art Rendering of Triangulated Surfaces Using Discrete Lines of Curvature. In *Proceedings of WSCG'2000* (2000), The University of West Bohemia, Plzeň, Czech Republic, pp. 168–175.

[SABS94] SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H.: Interactive Pen-and-Ink Illustration. In *Proceedings of SIGGRAPH'94* (New York, 1994), ACM Press, pp. 101–108.

[SALS96] SALISBURY M. P., ANDERSON C., LISCHINSKI D., SALISIN D. H.: Scale-Dependent Reproduction of Pen-and-Ink Illustration. In *Proceedings of SIGGRAPH'96* (New York, 1996), ACM Press, pp. 461–468.

[SS02] STROTHOTTE T., SCHLECHTWEG S.: *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers, San Francisco, 2002.

[SWHS97] SALISBURY M. P., WONG M. T., HUGHES J. F., SALESIN D. H.: Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proceedings of SIGGRAPH'97* (New York, 1997), ACM Press, pp. 401–406.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-Generated Pen-and-Ink Illustration. In *Proceedings of SIGGRAPH'94* (New York, 1994), ACM Press, pp. 91–100.

[WS96] WINKENBACH G., SALESIN D. H.: Rendering Parametric Surfaces in Pen and Ink. In *Proceedings of SIGGRAPH'96* (New York, 1996), ACM Press, pp. 469–476.