Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Fast Primitive Distribution for Illustration

Adrian Secord, Wolfgang Heidrich and Lisa Streit†

Department of Computer Science
The University of British Columbia
Vancouver, Canada

Abstract

In this paper we present a high-quality, image-space approach to illustration that preserves continuous tone by probabilistically distributing primitives while maintaining interactive rates. Our method allows for frame-to-frame coherence by matching movements of primitives with changes in the input image. It can be used to create a variety of drawing styles by varying the primitive type or direction. We show that our approach is able to both preserve tone and (depending on the drawing style) high-frequency detail. Finally, while our algorithm requires only an image as input, additional 3D information enables the creation of a larger variety of drawing styles.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Nonrealistic Rendering, Halftoning & Dithering, Monte Carlo Techniques, Image Processing, Printing, Rendering

1. Introduction

Illustrations are interesting alternatives to traditional photorealistic rendering in terms of both artistic expression and the power to convey information more effectively using a possibly limited palette of tones. A number of approaches have been proposed to simulate illustration with computer generated imagery. Pen-and-ink illustration techniques are the most comparable to our work and can be grouped roughly into two categories: image-space methods (e.g.[25, 2]) and object-space methods (e.g.[31, 32, 18, 11, 14, 22]).

The former have usually aimed at producing high-quality images for printing, while most of the latter have been designed for interactive applications. Consequently, image-space methods focus more on representing tone and avoiding the undesirable "shower door"[16] effect, where strokes "stick" in image space independent of object movement. Object-space methods concentrate on achieving interactive rates and frame-to-frame stroke coherency. Some image-space methods discretize tone for cartoon shading effects or use a minimum number of strokes for expressiveness. Those that attempt to accurately represent tone with a significant number of *primitives* (pen or brush strokes or stipples) are not suitable for interactive applications because of their computational cost and their lack of primitive coherency between individual frames. Alternatively, object-space methods are typically restricted to attaching stroke textures directly to the 3D geometry. Such methods cannot easily simulate certain expressive drawing styles, such as allowing strokes to cross boundaries or the use of motion lines. Not insignificantly, object-space methods require 3D geometry and cannot be applied directly to images or photographs.

In this paper we propose an image-space method that accurately represents global tone in a scale-independent manner while preserving frame-to-frame coherence. The method is capable of producing both high-quality images suitable for printing and interactive renderings at lower resolutions. Our approach is similar to that of Salisbury et al.[25] in that we derive a probability density function (PDF), or "importance" from the input image. A sequence of precomputed, uniformly distributed points in 2D is then redistributed according to the PDF. Our PDF is designed *a priori* to account for the tone that will be generated by the redistribution process. This eliminates the expensive incremental updating procedures of the importance required in Salisbury et al.[25] Also, since the PDF varies continuously as the input image changes we can achieve frame-to-frame coherence of the primitive positions by reusing the same sequence of uniformly distributed input points for every frame. We support a large variety of different drawing styles including examples

† {ajsecord, heidrich, streit}@cs.ubc.ca

Figure 1: *A hatched pen-and-ink illustration using the image gradient for primitive orientation.*

that preserve sharp features and edges and others that allow individual strokes to cross edges. Figure 1 shows an example rendered with a feature-preserving style. Additional styles are possible if additional 3D geometry is available.

In the next section, we describe the necessary previous work, followed by the derivation of the PDF and the redistribution of the input points in Section 3. Section 4 describes different choices of input point distributions. Finally, in Section 5 we describe how to achieve different drawing styles, before presenting results in Section 6 and concluding with a discussion of future work in Section 7.

2. Related Work

Our work attempts to automatically place primitives for illustrative purposes while accurately representing tone with a limited palette and maintaining frame-to-frame coherence at interactive frame rates. Relevant related work includes automatic illustration techniques, real-time NPR techniques and techniques for accurately representing tone. Illustration techniques based on user-guided systems for placing primitives (e.g.[24, 4]) are less related and will not be discussed.

Accurate representation of continuous tone has long been an important goal of many traditional artists. However, many of the current automatic illustration techniques focus on simulating a particular medium or style and are less concerned with representation of tone and may even purposely limit the tonal range.[6] Consequently, the placement of primitives is often guided less by tone than by the desire to represent textures or features. However, increasing the accuracy of tonal reproduction provides the artist with a continuous rather than discrete palette and aids in producing less heavily stylized illustrations. The current automatic approaches to illustration can be grouped into two categories, object-space and image-space methods. We focus mainly on the pen-and-ink techniques in these categories, since our approach also uses a two-tone palette.

Object-space techniques are often used for interactive applications and attach individual primitives directly to the 3D geometry. For example, Winkenbach and Salesin[31] developed the concept of a *stroke texture*, which is a prioritized list of strokes for a single geometric object. A subset of these strokes is then used to approximate the local target tone. Hertzmann and Zorin[11] place hatches and cross hatches along a *cross field* derived from the local curvature of a 3D object. They approximate a discretized set of tones using regions of cross, single or no hatching (in addition to undercuts and Mach bands to improve contrast) and vary stroke width for continuity between tones. Klein et al.[14] developed the concept of *art maps*, which are prefiltered textures representing a certain drawing style for different target resolutions. Praun et al.[22] extended this concept to *tonal art maps*, which allow for smooth transitions between art maps to achieve continuous tone images.

These object space methods can usually achieve interactive frame rates with a high degree of coherence between successive frames by attaching the strokes to the 3D geometry. Stroke locations remain fixed in object space, and only the intensities of the strokes vary depending on the view. These methods limit the drawing styles that can be achieved. Some drawing styles in traditional pen-and-ink illustration call for fuzzy outlines or individual strokes crossing silhouette edges[8] which cannot be easily simulated with object-space methods. Praun et al.[22] achieve real-time frame rates and accurate representation of tone, but cannot move the strokes off the surface of the object.

Image-space methods place primitives using an input image, which can of course be the result of a rendered 3D model. Using images as input rather than 3D models, Salisbury et al.[23] developed a method for generating pen-and-ink illustrations that preserve tone across scales by using fairly expensive data structures to track discontinuities and preserve edges. Deussen et al.[2] present a stippling method that uses relaxation to ensure that the final points are uniformly spaced (Poisson disk, or "blue noise" distributed) which is visually more pleasing. Energy minimization in image-space has been used for certain types of primitive placement, including paint and streamline visualization.[12, 28] Salisbury et al.[25] place individual strokes in the image and distribute them according to an importance function that is derived from the difference between the input image and the sum of the preceding strokes. The importance function is stored in an image pyramid that needs to be updated after every stroke is placed. Our proposed approach uses a similar distribution of primitives. Techniques for accurate

representation of continuous tone in a discrete domain have been widely studied for creating photo-real representations in the field of *halftoning*. Some halftoning techniques that extend into NPR place varying types of primitives based on tone[27] or convey pattern by carefully constructing dither matrices based on image tone[20] or 3D scene information[30]. Using 3D geometry as input has allowed some image-space methods to use silhouette information, but replace straight silhouette edges with sketchy lines (e.g. Markosian et al.[18]) in image-space, or replace individual parts of the geometry with textured polygons that are oriented to face the viewer (Deussen and Strothotte[3]). While the described image-space algorithms produce images of very high quality, they usually require off-line processing and either do not address issues of accurately representing tone with larger primitives or do not address the "shower door" effect.

Some methods use a combination of techniques to achieve interactive frame rates. Kowalski et al.[15] place primitives similar to Salisbury et al.'s[25] image-space method, but fix them in object-space to achieve multiple frames-per-second. Kaplan et al.[13] achieve interactive rates by extending Kowalski's work to include a method of interactively placing primitives. They obtain frame-to-frame coherence by varying the shape and size of the primitives. Lake et al.[16] achieve real-time rates by using precomputed textures to represent three or four tones. The tones are associated with materials on the surface of 3D objects and strokes are added in image-space for expression. Freudenberg et al.[5] provide a method for preserving tone of filtered textures by manually constructing carefully designed *ink-maps*. While the latter three methods achieve interactive rates and can place primitives in image-space, which allows a wider variety of styles, they do not attempt to ensure accurate representation of continuous tone. The former method maintains continuous tone at interactive rates but requires too much manual construction of textures.

In our work, we derive a probability density function (PDF) from the input image, which is similar to the importance function of Salisbury et al.[25] The PDF is designed *a priori* to position primitives to achieve the correct output tone. The rendering itself then merely consists of taking a sequence of precomputed, uniformly distributed points in 2D, and redistributing them according to the PDF. Once the primitive locations are defined a variety of drawing styles can be used. Since the locations vary continuously with changes in the input the technique naturally achieves frame-to-frame coherence and scale-independent tone. We show that both the derivation of the PDF from an image and the actual rendering can be done at interactive frame rates.

3. Real-time Image-space Illustration

As outlined in Section 2, our method takes a set of possibly precomputed uniformly distributed random or quasi-random points, and redistributes them according to a probability density function (PDF) derived from an input image. We then place primitives at the redistributed points to obtain a pen-and-ink style image of the original scene.

We first describe the details of this redistribution process and then discuss the derivation of a PDF from a given input image. The tone of the original image in the stippled or hatched pen-and-ink output is preserved by the derived PDF.

3.1. Generating Random Points With a 1D PDF

Given a one-dimensional probability density function $p(x)$, $x \in [0,1]$, and a set of uniformly distributed random samples $\{x_i\}$ over $[0,1]$, we can redistribute the samples according to $p(x)$. An example of a 1D probability density function is shown in Figure 2(a). The *transformation method* is well-described in the literature (e.g.[21]):

We compute the *cumulative density function*

$$C(x) = \int_0^x p(t)\,\mathrm{d}t$$

as shown in Figure 2(b). We then invert $C(x)$ and transform each sample x_i to get $x_i' = C^{-1}(x_i)$. This is shown graphically in Figure 2(c) by mapping a set of samples on the y-axis through to the x-axis. The set $\{x_i'\}$, drawn as circles on the x-axis of Figure 2(c), is distributed according to the original probability density function $p(x)$. Note that for the purposes of the illustration, the input sample set is regularly distributed, but it should be chosen from a uniform random distribution. Also, an intuitive explanation of the mechanics of the transformation method is possible with Figure 2. The peaks in the probability density function are transformed through integration into areas of high slope. These areas of high slope consequently gather more input samples than other areas. Symmetrically, valleys in the density function correspond to areas of low slope in the cumulative density function, which intersect fewer input samples.

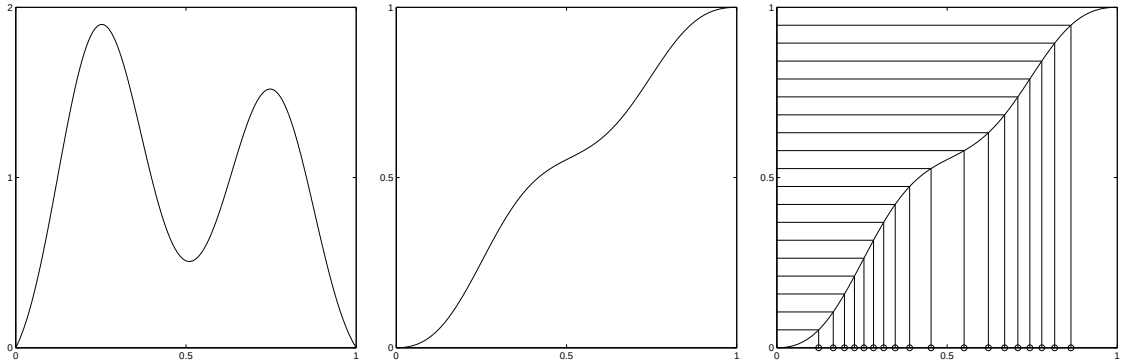
3.2. Generating Random Points With a 2D PDF

Given a 2D probability density function $p(\mathbf{x})$, $\mathbf{x} \in [0,1]^2$, and a set of uniformly distributed random points $\{\mathbf{p}_i\}$ over $[0,1]^2$, we can redistribute the points according to $p(\mathbf{x})$ using the transformation method.

To find the y-coordinate $\mathbf{q}_{i,y}$ of a redistributed point $\mathbf{q}_i$ we compute the *cumulative density function*

$$M(y) = \int_0^y m(t)dt, \text{ where } m(y) = \int_0^1 p(\mathbf{x})\,\mathrm{d}x,$$

and obtain $\mathbf{q}_{i,y} = M^{-1}(\mathbf{p}_{i,y})$. The function $m(y)$ is the *marginal density function* of $p(\mathbf{x})$. In a 2D image, $m(y)$ can be considered the average intensity of the scanline y. Note that $M(y)$ is monotonic, but not necessarily strictly monotonic if some scanlines have zero intensity. Hence, the inverse $M^{-1}(y)$ exists almost everywhere except at isolated points.



(a) Example 1D probability density function.

(b) Example 1D cumulative density function.

(c) Example 1D cumulative density function and regular sampling of its inverse.

Figure 2: *1D example of redistributing points according to a probability density function. Compare the samples (circles) at the bottom of Figure 2(c) to the density function in Figure 2(a).*

Given $\mathbf{q}_{i,y}$, we can now determine the x-coordinate $\mathbf{q}_{i,x}$ by redistributing $\mathbf{p}_{i,x}$ according to the PDF of the respective scanline. Mathematically, this is described by a conditional PDF

$$c(x|\mathbf{q}_{i,y}) = \frac{p(x,\mathbf{q}_{i,y})}{m(\mathbf{q}_{i,y})}$$

and its cumulative density function

$$C(x|\mathbf{q}_{i,y}) = \int_0^x c(s|\mathbf{q}_{i,y})\,\mathrm{d}s.$$

As before, the x component of the new point is given by the inverse of that function: $\mathbf{q}_{i,x} = C^{-1}(\mathbf{p}_{i,x}|\mathbf{q}_{i,y})$.

For a discrete PDF $p(\mathbf{x})$, such as one arising from an image, $M(y)$ and $C(x|y)$ are easily precomputed as a 1D and a 2D table, respectively.

3.3. Deriving a PDF From an Image

In stippling, hatching and other applications, the samples that we would like to place have finite area. It is possible, and with increasing density quite likely, that two randomly placed samples will overlap. Once a pixel has been covered by a primitive, further overlap has no effect. Thus the area covered by all samples does not change linearly with the number of samples. For example, if we were to set N pixels uniformly at random in an image of M total pixels, the ratio of the number of pixels set to the number of pixels chosen is nonlinear and reaches only about 60% when $N = M$. The non-linearity is solely due to overlap – pixels chosen more than once do not change the resulting image. It is, however, possible to correct for overlap by modifying the density function accordingly. We will first consider the simple case where all primitives cover a single pixel.

Let $I(\mathbf{x}_i)$ be the intensity of the base image at pixel $\mathbf{x}_i$ as a value between 0 and 1. Note that 2D pixel positions do not play a role in the case of single pixel primitives, so we use the simpler notation with a single subscript to distinguish between pixels. The drawing process should create a binary image where the probability that a pixel $\mathbf{x}_i$ is set is identical to the intensity of the corresponding pixel in the base image, that is, $p(\mathbf{x}_i = 1) = I(\mathbf{x}_i)$. Furthermore, we want to be able to independently place random samples for generating this binary image. We need to find a PDF $q(\mathbf{x})$ that can be used with the algorithm from Section 3.2 such that once primitives have been placed and overlap taken into consideration, the probability of each pixel being set is as above.

After placing N samples according to the density function $q(\mathbf{x})$, the probability that pixel $\mathbf{x}_i$ has *not* been set is $(1 - q(\mathbf{x}_i))^N$. Therefore, we require that

$$\begin{aligned} 1-(1-q(\mathbf{x}_i))^N &= p(\mathbf{x}_i = 1) \\ q(\mathbf{x}_i) &= 1 - \sqrt[N]{1-p(\mathbf{x}_i=1)} \end{aligned} \qquad (1)$$

for all $i = 1 \ldots n$. We also require that $q(\mathbf{x}_i)$ integrate to one so that it satisfies the definition of a probability density function. Combining this constraint with Equation 1 gives us

$$\sum_{i=1}^{n} q(\mathbf{x}_i) = 1 \Leftrightarrow \cdots \Leftrightarrow \sum_{i=1}^{n} \sqrt[N]{1-p(\mathbf{x}_i=1)} = n-1. \qquad (2)$$

This non-linear equation for N can only be solved directly for a constant intensity image $I(\mathbf{x}_i) = I_0$. For general images this equation system cannot be solved directly, however standard numerical methods work well. We bracket the value of N between a minimum and maximum value and perform a binary search. The maximum can be chosen as the number of samples that would be required to cover an image of con-

stant intensity very close to black. Furthermore, in practice this binary search does not have to extend over the full sum in Equation 2. Every term in the sum only depends on the probability $p(\mathbf{x}_i = 1)$ for each pixel, which in turn depends only on the intensity $I(\mathbf{x}_i)$ of the base image, and that is usually quantized, for example to 256 levels. Therefore, if we denote the quantized intensity levels as I_j, $j = 1 \ldots K$, and the histogram of the intensities in the base image as $H(I_j)$, we can rewrite Equations 1 and 2 as

$$q(I_j) = 1 - \sqrt[N]{1 - I_j} \tag{3}$$

and

$$\sum_{j=1}^{K} H(I_j) \sqrt[N]{1 - I_j} = n - 1 \tag{4}$$

which involves sums over only K levels rather than n pixels.

A similar formula can be derived for primitives larger than a single pixel (see Appendix A). That method requires the additional assumption that the tone to be reproduced is approximately constant over the support of the stroke. This is the case, for example, if the strokes are thin in one direction and they are oriented orthogonal to the gradient of the tone in the source image. Note that drawing styles that require individual strokes to cross discontinuity edges automatically compromise on the tone reproduction, so this assumption is not a serious limitation. Similarly, it is clear that no pen-and-ink rendering style can precisely preserve features smaller than the stroke width. A grayscale ramp illustrating the tone preservation with this method is shown in Figure 3.

3.4. Algorithm

To summarize the previous theory and the following implementation details, we list the steps of the complete algorithm:

1. Capture an input image, possibly from a 3D object
2. Build the histogram of the image intensities
3. Search for N, the number of primitives to place, using Equation 4
4. Compute $q(I_j)$, $j = 1 \ldots K$ using Equation 3
5. Integrate scanlines and invert to form M^{-1}
6. Integrate and invert to form C^{-1}
7. Distribute the N primitives according to M^{-1} and C^{-1}

4. Input Point Distributions

To this point we have assumed that the input points are Poisson distributed, i.e. are independently chosen, uniformly distributed random points. This does not necessarily have to be the case; we can choose the input points from a variety of different random distributions and quasi-random sequences.

Poisson distributions tend to form small clusters of points that are relatively close together. It is widely accepted in the stippling and halftoning literature[29, 2] that a more uniform point spacing yields more visually pleasing results, so that Poisson disk distributions ("blue noise"[29]) are usually preferred.

Our method maps any given input point set through a distribution function to determine the final location of primitives. If the base image used to generate the distribution function is approximately constant locally around a given point, then the mapping obtained by the distribution function is approximately linear locally. This indicates that desirable properties such as minimal distances in a Poisson disk distribution should be *locally* preserved in regions of approximately constant intensity. This property breaks down in regions of strong changes such as edges, but that is also the case for traditional images.

We use the first N samples of a precomputed point sequence (where N is determined according to Section 3.3) for every frame in order to maintain frame-to-frame coherence. This means that we are restricted to point sequences where any subset consisting of the first N samples is well distributed. This eliminates, for example, the Hammersley[10] point set, which is only well-distributed in its entirety, but allows us to use the Halton[9] or Sobol[26] sequences.

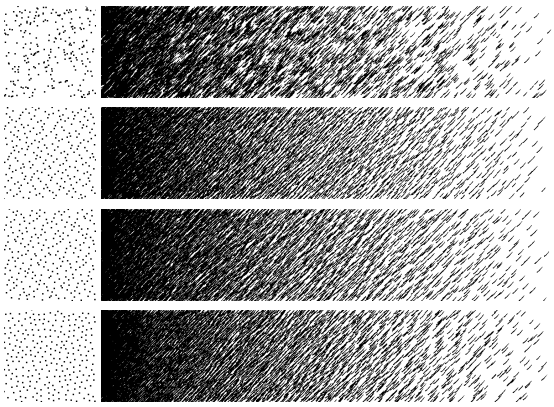


Figure 3: *Comparison of different input distributions with a point distribution and a grayscale ramp. From top to bottom: Poisson distribution, Halton sequence, Sobol sequence and hierarchical Poisson disk sequence.*

In addition, we experimented with Poisson disk distributed points generated with the hierarchical approach of McCool and Fiume[19]. Besides being faster than other methods for generating Poisson disk distributed points, this approach also has the advantage of producing a sequence of points that are Poisson disk distributed with a decreasing disk radius. In Figure 3 we compare the performance of the different approaches by applying them to a simple grayscale ramp.

It can be seen that the Poisson distribution clusters the individual samples, resulting in a very noisy and irregular image. The other distributions avoid excessive clustering and

achieve more uniform sample spacing. The Sobol sequence, however, sometimes tends to form undesirable patterns. The Halton sequence and the hierarchical Poisson disk distribution are free of those artifacts. Based on this experience, we usually choose the Halton sequence for our method since it is computationally more efficient to generate. The other images in this paper are rendered using the Halton sequence unless otherwise noted.

5. Drawing Styles

The algorithms from Section 3 can be used to determine stroke positions from input images. What remains open are issues of acquiring the input images from 3D models and orienting the strokes. Different choices of methods for these tasks give rise to a large variety of rendering styles.

The basic algorithm for placing the strokes only requires a grayscale input image to generate a PDF. Hence, our algorithm is able to work from data like photographs or paintings. These can be preprocessed with any kind of image processing tool to achieve the desired effect. For example, *indication*[8, 31] can be achieved by increasing or decreasing the brightness in selected parts of the input image. In addition, we include a transfer function in our system, which allows for on-the-fly adjustments of contrast or tone on the whole image.

Starting from a 3D model rather than an input image, we first use an OpenGL rendering pass with traditional shading and specular and diffuse lighting to generate an image. However, by starting from a 3D model we can easily generate additional information, encode it into the image, and later use it to drive different drawing styles. For example, we can draw silhouette outlines either using environment maps[7] or by explicitly traversing the input mesh and finding silhouette edges. Similarly, we can encode additional information for determining the stroke *direction* into other color channels. In an RGBA framebuffer, this gives us three channels to be used for such additional information. We have experimented with the following options and more are certainly possible:

- Computing a per-vertex tangent or parametric direction, projecting it into image space, and interpolating it for every pixel (all steps are possible in a single pass in hardware) to be used as a stroke direction. This yields results similar to the ones described by Winkenbach and Salesin[32].
- Computing and encoding a normal direction in much the same way. This is an interesting way of drawing short fur or grass.
- Encoding 2D stroke directions into a camera-aligned environment map. This can be used to give areas facing different directions different stroke properties.
- Encoding object or material identifications for individual parts of the scene, which can then be used to render different objects in different styles.

Based on this information, we can generate variations on three basic drawing styles:

Point Stippling

The simplest rendering style is point stippling, which does not require any directional information. We simply place OpenGL points or scanned stipple textures at the positions obtained with the method described in Section 3.

Figure 4: *A stippled image of foot bones.*

Hatching

For hatching we either use OpenGL lines of different widths and lengths, or scanned texture maps of real pen-and-ink strokes. For the stroke orientation we can choose between a constant direction or any kind of direction encoded into color channels as described above. Finally, we can also compute the gradient field of the image intensity, and orient strokes orthogonal to it. This works both for images and 3D models, and orients strokes parallel to discontinuity edges, which helps preserving the tone of the input image as described above. In areas in which the intensity gradient is not strong enough, we can either default to a constant stroke direction, or search for a stronger gradient in an image pyramid.

Cross Hatching

For cross hatching we can vary the hatch direction probabilistically. Say we want to start cross hatching for some intensity threshold $I(\mathbf{x}) > \alpha$. Whenever we place a stroke at pixel $\mathbf{x}_{i,j}$), we choose the orientation as follows: if the intensity of the input image $I(\mathbf{x}_{i,j})$ at that location is less then the threshold, we compute the stroke direction with any of the methods described above. If $I(\mathbf{x}_{i,j})$ is larger than the threshold, however, then we choose the original orientation with a probability of $\alpha/I(\mathbf{x}_{i,j})$, and a secondary direction with

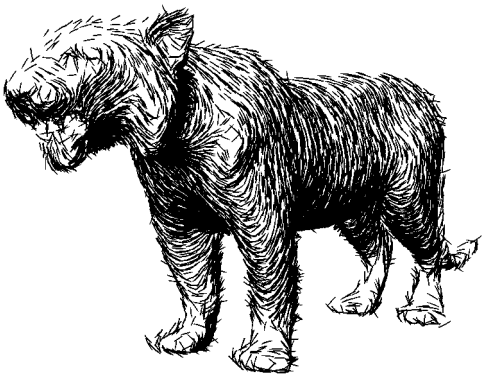


Figure 5: *A tiger model hatched with large strokes to achieve a furry look.*

probability $(1-\alpha)/I(\mathbf{x}_{i,j})$. This secondary orientation can either be a fixed direction, the original direction rotated by some fixed angle, or any rotation inferred from an encoding of an object-space property as discussed above.

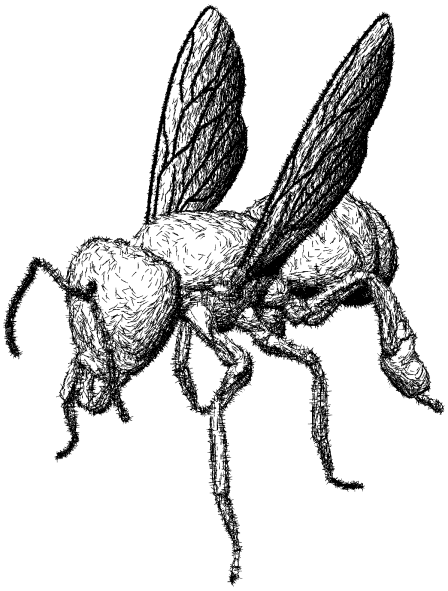


Figure 6: *Cross hatching in very dark areas together with silhouette extraction cause a hairy appearance of the legs and antennae.*

6. Results

We have implemented a system demonstrating our approach to distributing primitives that preserves continuous tone in a scale-independent manner. We can either load images or 3D scenes as a starting point for our method. The different drawing methods described in the previous section, as well as the parameters for the initial rendering of the 3D model, can produce a multitude of different styles of which we can only show a small subset.

Figure 1 shows a bust model rendered with hatching, where the hatches are aligned against the image gradient. Only the rendered scene is used to align the strokes, and not the 3D model information. Figure 1 shows a good preservation of sharp features. Figure 4 and 5 show results for stippling and hatching, respectively. Figure 6 shows a bee rendered with cross hatching, which gives the legs and the antennae a hairy appearance. Figure 8 shows how our system accurately preserves continuous tone. Figure 9 (Top) shows every fifth frame of an twenty-five frame animation. All of the primitives in the frame on the far left appear in the frame on the far right at adjusted locations. Extra primitives have been added over the twenty-five frames since the image darkens. The primitives have been "shape-encoded" to help trace their movement. More examples can be found at the end of this paper.

The performance of our method depends on several factors. The first one is the size of the table used for the PDF and the distribution functions. This size is algorithmically independent of the resolution of the final image, so small table sizes can be used to reduce rendering cost. However, if the table gets too small, sharp edges will start to alias. The resolution at which these artifacts start to be visible depends on the size of strokes, but we generally found that using a PDF table of half the size of the target image produces good results except for stippling with very small point sizes. The second factor for performance is the rendering style. Computing image-space gradients for orienting the strokes is relatively expensive, while generating the orientation from encoding a 3D direction is practically free. The cost of rendering the initial 3D model is negligible since the meshes do not need to be very detailed to serve as a basis for pen-and-ink illustration. The number of strokes rendered is another cost factor. Very dark images take longer than light ones, and images with large strokes render faster than images with very small ones. The major cost here is the rendering of the individual points, lines, or texture-mapped strokes using OpenGL (which is a geometry-bound operation on current graphics hardware).

Depending on the above parameters, we were able to achieve between 2 frames per second and 15 frames per second on a 1.5GHz Pentium with a GeForce 3 card, using PDF sizes between 256×256 and 512×512 for the scenes shown in this paper. No specialized hardware was used with our software implementation to obtain these frame rates.

A common goal of many real-time NPR techniques is to achieve frame-to-frame coherence of primitive placement. In our approach individual primitives are placed in image-space, are therefore not attached to 3D geometry and can move around freely. Thus, we cannot expect them to move exactly with the 3D object points. However, since we re-use the same set of uniformly distributed points, small changes

in the input image (and therefore in the PDF) will only result in small adjustments of the point positions. For example, if a highlight moves across a 3D object due to a moving light source, then the individual strokes will rearrange themselves continuously around the highlight as shown in the top of Figure 9. Similarly, if the transfer function is adjusted continuously, the strokes will slowly move across the image to adapt for the changes. These effects are demonstrated in the video. Since our approach distributes primitives in real-time based on intensity and primitive size, continuous tone is precisely replicated. Even with the use of slightly larger primitives the global tone is accurately preserved and is scale-independent. Our method is based entirely on the location of primitives, allowing us to easily simulate many drawing styles by altering the primitive type or direction.

7. Conclusions and Future Work

In this paper we have presented an image-space algorithm for generating continuous tone illustrations at interactive frame rates. Our approach for distributing primitives naturally provides scale independent results with frame-to-frame coherency. That is, individual strokes move continuously as the input image changes. We demonstrate that a number of different drawing styles are possible using only an image as an input. More styles are possible if additional information such as silhouette edges or material IDs can be derived from a 3D model.

Preservation of continuous tone with our approach is dependent on the size of the primitive, the feature to be preserved (i.e. edges) and the viewing distance. This is a standard limitation, since, as with any halftoning technique, the formation of continuous tone from a discrete palette relies on the spatial integration of the eye. Thus, our approach preserves tone if the choice of drawing primitive, image or model and viewing distance is reasonable. Our frame rate depends on several factors (brightness, size of PDF table, drawing style etc.) and is thus not constant. This could limit the applicability of our method in some situations. A final limitation is the distance a primitive moves from frame to frame. While primitives move continuously, the apparent movement from frame to frame can be distracting, especially at the lower frame rates.

In the future we would like to extend the tone correction method to work for grayscale and color primitives or graftals[15]. Grayscale or semi-transparent strokes or brushes could be used to simulate other non-photorealistic rendering effects such as watercolor[1]. Additional pen-and-ink styles could also be incorporated into our method. For example, it would be easy to clip strokes against object masks rendered using an ID buffer algorithm. This would preserve discontinuities in the image without having to align strokes orthogonal to the gradient field. Finally, it should be possible in the future to offload the placement of the stipples onto the graphics chip, which would drastically reduce the bandwidth requirements of the method. This would require a programmable vertex engine similar to the one described by Lindholm et al.[17], but with the additional possibility to perform per-vertex table lookups.

8. Acknowledgements

The authors would like to acknowledge the extremely helpful nature of the anonymous reviewers' comments, which led to many clarifications and the addition of Sections 3.1 and 3.4.

References

1. C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. Computer-generated watercolor. In *Proceedings of SIGGRAPH 97*, pages 421–430, August 1997. 8

2. O. Deussen, S. Hiller, C. van Overveld, and T. Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum (Proc. of Eurographics)*, 19(3), August 2000. 1, 2, 5

3. O. Deussen and T. Strothotte. Computer-generated pen-and-ink illustration of trees. In *Proceedings of SIGGRAPH 2000*, pages 13–18, July 2000. 3

4. F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Eurographics Workshop on Rendering 2001*, pages 71–82, June 2001. 2

5. B. Freudenberg, M. Masuch, and T. Stothotte. Walk-through illustrations: Frame-coherent pen-and-ink in game engine. In *Proceedings of Eurographics 2001*, pages 184–191, 2001. 3

6. B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. AK Peters, 2000. 2

7. B. Gooch, P. Sloan, A. Gooch, P. Shirley, and R. Riesenfeld. Interactive technical illustration. In *ACM Symposium on Interactive 3D Graphics*, pages 31–38, April 1999. 6

8. A. Guptill. *Rendering in Pen and Ink*. Watson and Guptill Publications, 1997. 60th anniversary edition, edited by S. Meyer. 2, 6

9. J. Halton and G. Weller. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, July 1964. 5

10. J. Hammersley and D. Handscomb. *Monte Carlo Methods*. Wiley, 1964. 5

11. A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of SIGGRAPH 2000*, pages 517–526, July 2000. 1, 2

12. Aaron Hertzmann. Paint By Relaxation. In *Proceedings Computer Graphics International 2001 (Hong Kong, July 2001)*, pages 47–54, Los Alamitos, 2001. IEEE Computer Society Press. 2

13. M. Kaplan, B. Gooch, and E. Cohen. Interactive artistic rendering. In *Proceedings of NPAR 2000*, pages 64–74, June 2000. 3

14. A. Klein, W. Li, M. Kazhdan, W. Correa, A. Finkelstein, and T. Funkhouser. Non-photorealistic virtual environments. In *Proceedings of SIGGRAPH 2000*, pages 527–534, July 2000. 1, 2

15. M. Kowalski, L. Markosian, J. Northrup, L. Bourdev, R. Barzel, L. Holden, and J. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999. 3, 8

16. A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of NPAR 2000*, pages 13–20, June 2000. 1, 3

17. E. Lindholm, M. Kilgard, and H. Moreton. A user programmable vertex engine. In *Proceedings of SIGGRAPH 2001*, pages 149–158, August 2001. 8

18. L. Markosian, M. Kowalski, S. Trychin, L. Bourdev, D. Goldstein, and J. Hughes. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 97*, pages 415–420, August 1997. 1, 3

19. M. McCool and E. Fiume. Hierarchical poisson disk sampling distributions. In *Graphics Interface '92*, pages 94–105, May 1992. 5

20. V. Ostromoukhov and R. Hersch. Artistic screening. In *Proceedings of SIGGRAPH 1995*, pages 219–22, August 1995. 3

21. J. Pitman. *Probability*. Springer, 1992. 3

22. E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proceedings of SIGGRAPH 2001*, pages 579–584, August 2001. 1, 2

23. M. Salisbury, C. Anderson, D. Lischinski, and D. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of SIGGRAPH 96*, pages 461–468, August 1996. 2

24. M. Salisbury, S. Anderson, R. Barzel, and D. Salesin. Interactive pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, pages 101–108, July 1994. 2

25. M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of SIGGRAPH 97*, pages 401–406, August 1997. 1, 2, 3

26. I. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Computational Math. And Math. Phys*, (4):86–112, 1967. 5

27. L. Streit and J. Buchanan. Importance driven halftoning. In *Computer Graphics Forum (Proc. of Eurographics)*, pages 207–217, August 1998. 3

28. Greg Turk and David Banks. Image-Guided Streamline Placement. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 453–460. ACM SIGGRAPH, Addison Wesley, August 1996. ISBN 0-89791-746-4. 2

29. R. Ulichney. *Digital Halftoning*. MIT Press, 1987. 5

30. O. Veryovka and J. Buchanan. Comprehensive halftoning of 3d scenes. In *Computer Graphics Forum (Proc. of Eurographics)*, pages 13–21, August 1999. 3

31. G. Winkenbach and D. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, pages 91–100, July 1994. 1, 2, 6

32. G. Winkenbach and D. Salesin. Rendering parametric surfaces in pen and ink. In *Proceedings of SIGGRAPH 96*, pages 469–476, August 1996. 1, 6

Figure 7: *Rendering of the Lena image using hatching and cross hatching. Primary strokes are aligned perpendicular to the gradient in regions of strong gradients and at a* 45° *angle in areas where the gradient is small.*

Appendix A: Tone Correction for Large Stroke Sizes

Tone reproduction is possible for larger stipples or strokes under two assumptions. Firstly, it is clear that drawing styles that require strokes to cross discontinuities cannot *precisely* preserve tone around those discontinuities. We can therefore only hope for exact tone reproduction if we either align strokes parallel to discontinuities, or clip them accordingly. Secondly, we can only hope to preserve feature sizes that are no smaller than the width of the strokes used.

Under these assumptions, however, the tone correction procedure from Section 3.3 can be extended to strokes and stipples of larger size. Where we could previously ignore spatial relationships between the different pixels, these now have to be taken into account. We adapt our notation to 2D indices for the images and probability tables (e.g. $p(\mathbf{x}_{i,j})$ rather than $p(\mathbf{x}_i)$).

We can then describe the probability $p((\mathbf{x}_{i,j}) = 1)$ that a pixel is set after rendering N stipples as

$$p(\mathbf{x}_{i,j} = 1) = 1 - \left(1 - \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} q(\mathbf{x}_{k,l}) s(\mathbf{x}_{k-i,l-j})\right)^N \quad (5)$$

where $i = 1 \ldots n_x$, $j = 1 \ldots n_y$ and $s(\mathbf{x}_{k-i,l-j})$ is the image of

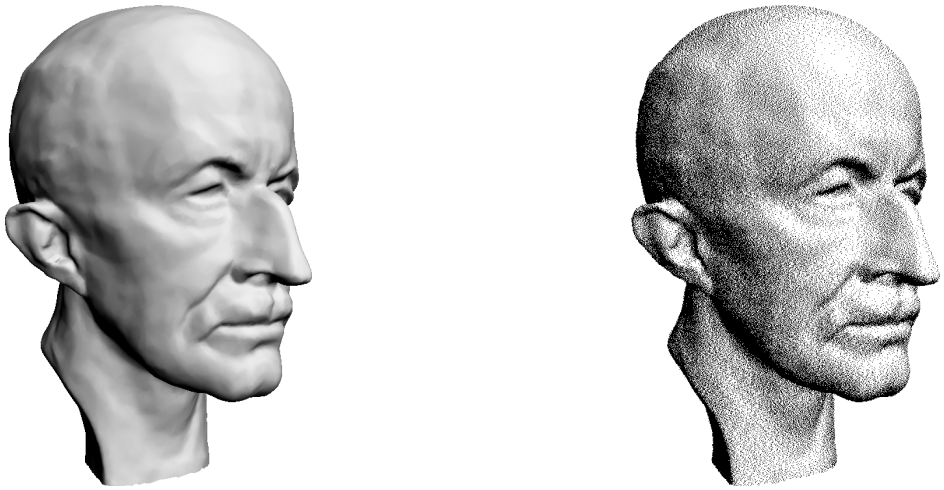


Figure 8: *A bust stippled with pixel-sized stipples. The left image is the original greyscale input and the right is the stippled image. This result is similar to the ones that would be obtained by halftoning methods. Any apparant differences in tone between these two images, when viewed from a distance of several feet, is due to the halftoning screen of the individual printer.*

the stroke or stipple that would be drawn if a sample was to be placed in pixel $(\mathbf{x}_{k,l})$. The value of $s(\mathbf{x})$ is either one or zero; we do not consider grayscale values. With the updated notation, the constraint from Equation 2 becomes

$$\sum_{k=1}^{n_x}\sum_{l=1}^{n_y} q(\mathbf{x}_{k,l}) = 1. \tag{6}$$

To determine $q(\mathbf{x}_{i,j})$ and the number of samples N, Equations 5 and 6 have to be solved simultaneously. However, it is easy to see that this system of equations does not in general have a solution: consider an image with a one-pixel wide dark line on white background, and a circular stipple pattern of radius larger than 1 pixel. Since the line is very dark, the stipples have to be densely spaced on it. However, since every individual stipple is wider than the line itself, it will automatically also cover adjacent parts of the image which should be white, i.e. $p((\mathbf{x}_{i,j}) = 1) = 0$. Thus it is obvious that the system of Equations 5 and 6 is not solvable if the image contains features narrower than the stipples or strokes used for hatching.

An alternative strategy would be to minimize the difference between the left and the right side of Equation 5 subject to Equation 6 with the additional constraint that $q(\mathbf{x}_{k,l}) \geq 0$ for all k,l. This is a non-linear constraint optimization problem with a large number of variables (equal to the number of pixels in the final image). While this system is rather sparse for reasonable stroke sizes, it is beyond hope of solving in real time.

However, we can simplify Equation 5 by assuming the target tone is approximately constant over the support of a stipple. This may sound restrictive, but is easily achieved by aligning the strokes orthogonal to the gradient field of the input image or otherwise preventing the stroke from crossing discontinuities as described above. With this simplification, Equation 5 becomes

$$1 - \left(1 - \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} q(\mathbf{x}_{k,l}) s(\mathbf{x}_{k-i,l-j})\right)^N \tag{7}$$

$$\approx\ 1 - \left(1 - q(\mathbf{x}_{i,j}) \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} s(\mathbf{x}_{k-i,l-j})\right)^N \tag{8}$$

$$=\ 1 - \left(1 - q(\mathbf{x}_{i,j}) s_{i,j}\right)^N$$

where $s_{i,j} \equiv \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} s(\mathbf{x}_{k-i,l-j})$. Intuitively, $s_{i,j}$ is the number of pixels (x_k, y_l) that would cause pixel $(\mathbf{x}_{i,j})$ to be covered if a sample was placed in one of them. Note that this relationship allows for different stroke sizes and orientations depending on the local intensity or even pixel position. However, this relationship should be simple to compute in order to facilitate a rapid calculation of the $s_{i,j}$.

Once $s_{i,j}$ has been computed, we can solve Equation 8 for $q(\mathbf{x}_{i,j})$ just as in Section 3.3. The search for the correct number of samples to be placed can again be sped up by using a histogram based approach, where the size of the histogram is the product of the number of quantization levels for the image intensity and the number of different sizes used for strokes during rendering. For a small set of differently sized strokes this will still be more efficient than a search directly on the full resolution image.

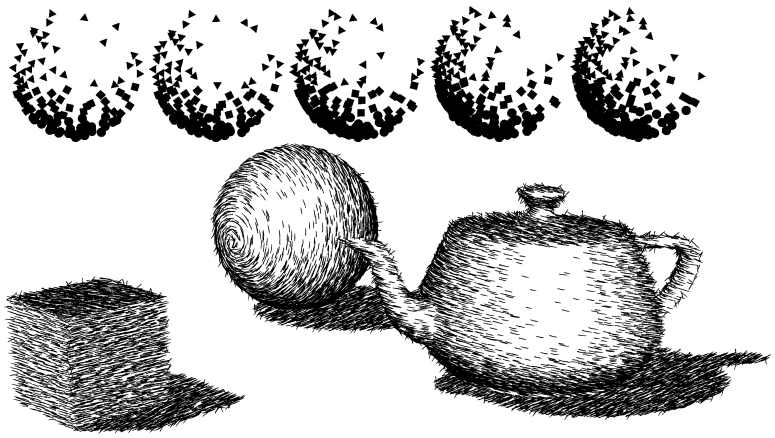


Figure 9: *Top: Frames of an animation showing stroke movement. Bottom: A simple 3D scene rendered with different stroke directions for the different objects using material IDs and object-coded directions.*

Figure 10: *Stanford bunny hatched with curved strokes.*

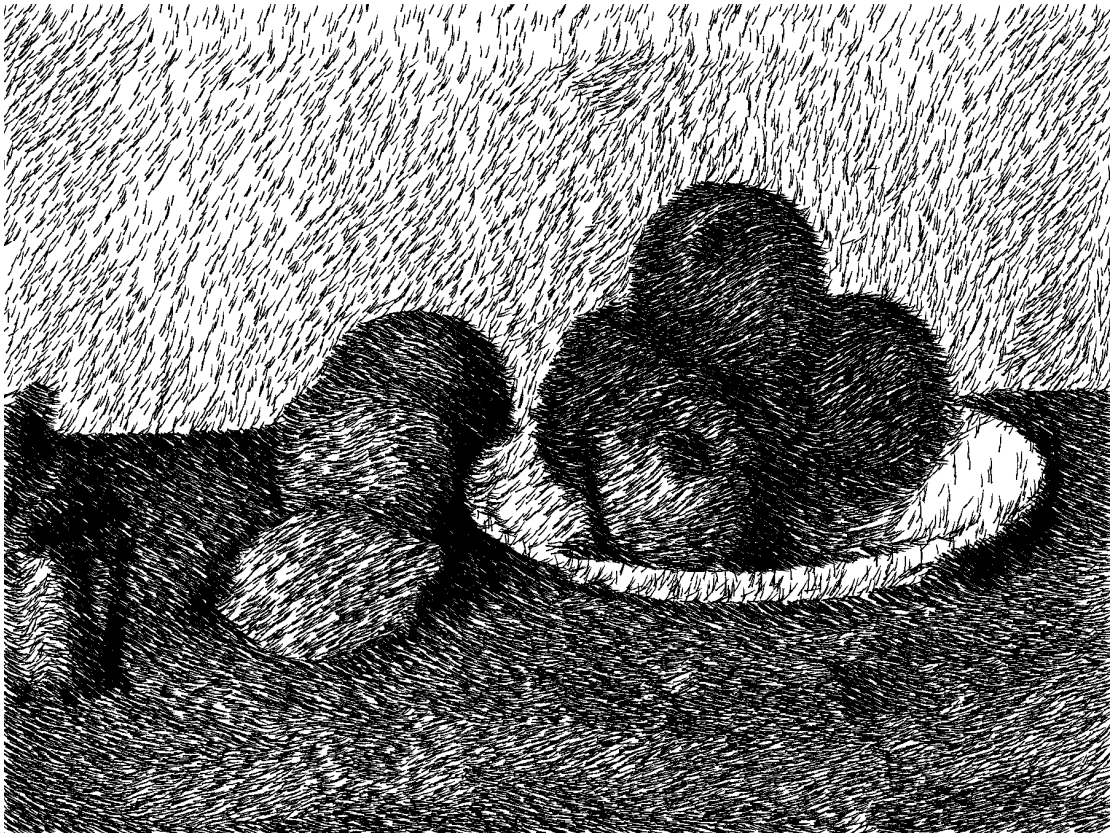


Figure 11: *A hatched rendering using Cezanne's "Still Life With Apples" as input. The stroke directions are a function of the color in the original image.*

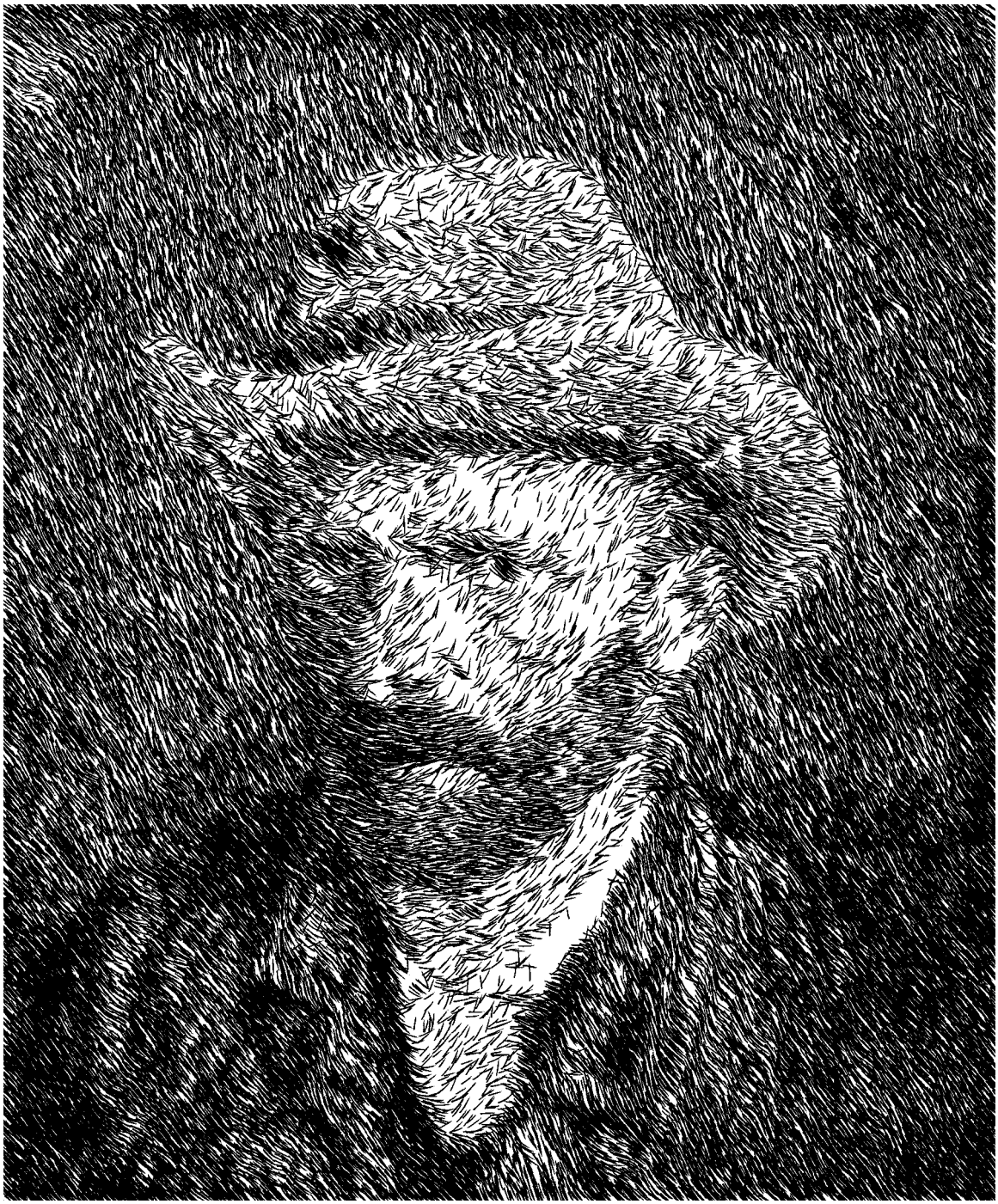


Figure 12: *A hatched rendering using Van Gogh's "Self Portrait With Felt Hat, 1888" as input. The stroke directions are a function of the color in the original image.*