Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Feature-guided Image Stippling

Dongyeon Kim[†1], Minjung Son[‡1], Yunjin Lee[§2], Henry Kang[¶3], and Seungyong Lee[‖1]

[1]POSTECH, Korea
[2]Seoul National University, Korea
[3]University of Missouri, St. Louis, USA

Abstract

This paper presents an automatic method for producing stipple renderings from photographs, following the style of professional hedcut illustrations. For effective depiction of image features, we introduce a novel dot placement algorithm which adapts stipple dots to the local shapes. The core idea is to guide the dot placement along 'feature flow' extracted from the feature lines, resulting in a dot distribution that conforms to feature shapes. The sizes of dots are also adaptively determined from the input image for proper tone representation. Experimental results show that such feature-guided stippling leads to the production of stylistic and feature-emphasizing dot illustrations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Picture/Image Generation]: Display algorithms; I.3.4 [Graphics Utilities]: Paint systems

1. Introduction

Describing a scene with a set of points has been an important and challenging issue in many areas of computer graphics, such as non-photorealistic rendering (NPR), point-based graphics, image-based rendering, and geometric processing. In this paper, we focus on point-based scene stylization, in particular, the problem of generating stylistic stipple illustrations from photographs.

Many of the previous stippling algorithms were developed and presented in the context of sampling. Assisted by carefully designed dot spacing schemes, they produce a dot distribution with reduced visual artifacts, such as aliasing. When used for image-guided stippling, these algorithms fill the image with well-spaced dots that properly describe the local tone. However, they do not in general take into account the shape or directionality of image features.

In this paper, we focus more on the 'style' of stippling rather than the spectral quality of sampling. We are particularly inspired by the professional *hedcut illustrations* (see Fig. 1), where the dots appear to follow some 'flow' along shapes. That is, the dot formation is strongly affected by the directionality of image features. As demonstrated by these illustrations, flow-guided distribution of dots adds to the

Figure 1: *Hedcut illustrations created by Randy Glass (www.randyglassstudio.com)*

† spiff@postech.ac.kr
‡ sionson@postech.ac.kr
§ yunjin@snu.ac.kr
¶ kang@cs.umsl.edu
‖ leesy@postech.ac.kr

Figure 2: *Stipple illustrations created by our method*

stylistic look, and also has the effect of enhancing or exaggerating important shapes as the dots collectively reflect the directionality of the features nearby, and it occurs not just around the features but almost everywhere.

Based on this observation, we develop an automatic dot placing algorithm that adapts dots to the surrounding shape. The core idea of our approach is to create 'feature flow' by extracting a distance field and offset lines from image features, with which to guide the dot placing along shapes. We thus call it *feature-guided stippling*. Fig. 2 shows some of our stipple rendering results.

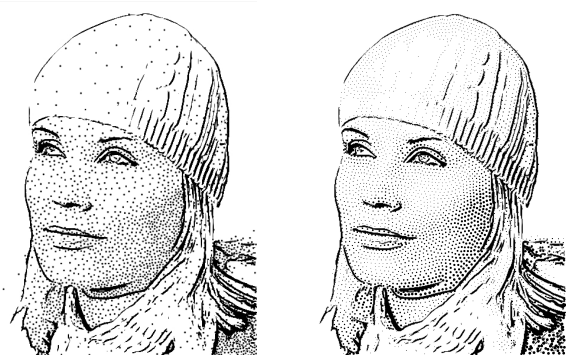
1.1. Contributions

Unlike previous stippling algorithms, we pursue a new style of stippling where stipple dots collectively follow the nearest image feature direction. To the best of our knowledge, the concept of 'directional stippling' is new in the field and has not been attempted. Imitating the visual quality of hedcut illustrations is particularly challenging as it demands artistic intuition and finesse in creating flow as well as in arranging dots. As a computerized solution, we propose a constrained Lloyd algorithm that uses a set of lines offset from the feature lines. We also develop a weighted centroid computation method to provide adaptive control of the influence from offset lines to the Voronoi cells. In addition, our method allows for an intuitive control of rendering style with just a few parameters.

2. Related Work

2.1. Digital image halftoning

Image halftoning refers to a technique that approximates the original image with a limited number of intensity levels, typically black and white [FS76, Ost01]. Since an output of halftoning is often a collection of black dots (pixels) on a white image, it may be viewed as a kind of stipple illustration. While halftoning is a visual approximation technique, stippling is more of an artform. In general, more freedom is given to stippling in controlling the size, density, shape, style, orientation, and intensity of the dots. Deussen et al. [DHVOS00] also pointed out that additional lines (such as feature lines) are often used in stippling to allow dots to interact with those lines.



(a) Non-directional (b) Directional (ours)

Figure 3: *Non-directional vs. Directional dot placing. (a) produced by the method of [Kopf et al. 2006]. (b) our method. In both figures, the same line drawing is superimposed onto the output of stippling.*

2.2. Stippling

Salisbury et al. [SWHS97] presented a pen-and-ink illustration technique, which is capable of producing stipple illustrations when pen strokes are replaced with dots. They

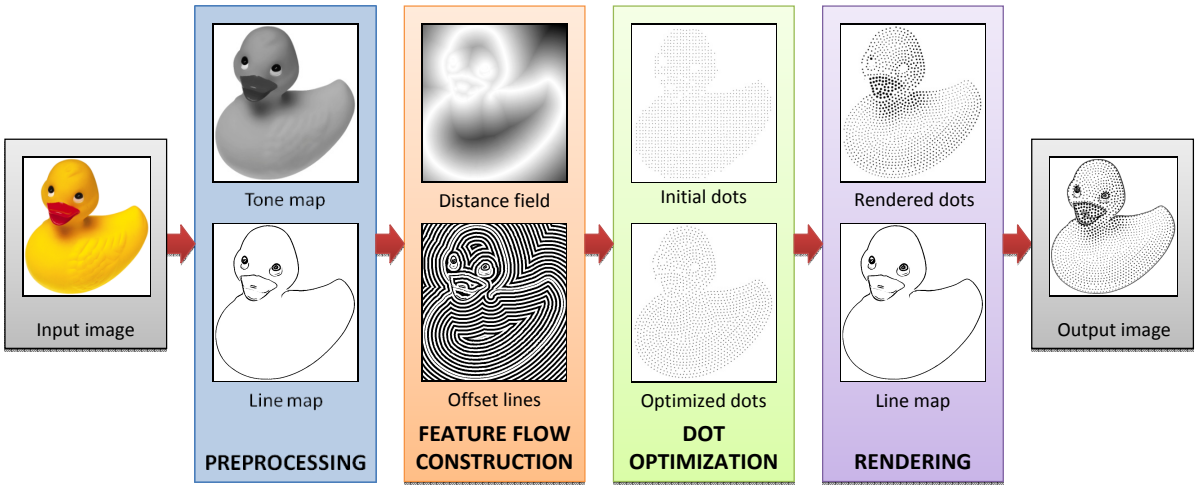


Figure 4: *Process overview: For better visualization, in the dot optimization step, the dots in the background are not shown.*

use a difference image algorithm to produce a roughly even dot distribution in local neighborhood. Deussen et al. [DHVOS00] presented a stipple drawing method based on Lloyd algorithm (i.e., construction of a centroidal Voronoi diagram) for more rigorous dot spacing, resulting in an exquisite illustration. Secord [Sec02] later modified this algorithm to produce a weighted centroidal Voronoi diagram that protects image features better than the constant-weighted version.

Stippling algorithms often rely on sophisticated sampling principles. In particular, many of them reduce aliasing artifacts by seeking a sampling property known as blue noise spectral characteristics. A dart throwing algorithm [Coo86] is a simple method to generate such point sets. Cohen et al. [CSHD03] presented a Wang-tile-based method to produce blue noise dot distribution, assisted by Lloyd relaxation. Kopf et al. [KCODL06] later proposed a recursive Wang tiling method for dynamic control of the point set density. Ostromoukhov et al. [ODJ04] introduced a fast blue noise sampling algorithm based on Penrose tiling, which was later improved by Ostromoukhov [Ost07] using rectifiable polyominoes for better spectral quality. These latter algorithms [KCODL06, ODJ04, Ost07] are all very fast, producing millions of dots per second, as the Lloyd relaxation step is preprocessed. Mould [Mou07] recently presented a stippling algorithm based on graph search (instead of Lloyd relaxation) for improve protection of image features such as edges.

While all of these cited algorithms are capable of producing high-quality stipple illustrations, they do not provide an important characteristic we are looking for – the collective dot alignment with local shapes. That is, they basically take into account the tone but not the shape of the surrounding region, and thus the resulting dots do not by themselves reveal any sense of directedness. This is illustrated in Fig. 3. While some algorithms [DHVOS00, Sec02, Mou07] do protect image features, they do not go as far as guiding all of the dots along some smooth feature flow.

2.3. Tile mosaics

Hausner [Hau01] showed that the centroidal Voronoi diagram, when computed with Manhattan distance metric, can constrain rectangular tiles to align with some user-defined feature lines and the associated direction field. While our problem at hand is similar to that of tile mosaics, distributing dots as in hedcut illustrations requires much more rigor and finesse as it calls for strict alignment of dots almost everywhere (see Fig. 1).

We thus build on Hausner's constrained Lloyd algorithm and adapt it to handle the feature-guided distribution of circular dots, rather than rectangular tiles. In particular, we incorporate a new set of constraints based on *offset lines*, to enable tight alignment of dots which directly improves the quality of the resulting illustration.

2.4. Engraving

Ostromoukhov [Ost99] addressed the problem of feature-driven tone representation in the context of facial line engraving. While his system generates a beautiful set of engraving lines flowing across the facial surface, it requires considerable user interaction as the line directions are determined by the user's interpretation of the facial structure. We aim to build an automatic and general method that can handle images of arbitrary scenes.

3. Overall Process

Fig. 4 illustrates the overview of our stippling scheme. We first process the input image to get it ready for the main stip-

pling procedure. This initial process includes tone map and line map creation. The tone map controls the tone-related dot attributes such as size, while the line map dictates the shape-related dot attribute, that is, location. In the next step, we form feature flow by extracting a distance field and a set of offset lines from the line map. The system then optimizes the regularly sampled initial dots using constrained Lloyd algorithm, for which we use offset lines as constraints so that the dots can closely follow the feature flow. Upon computing the sizes of dots, the system produces the target illustration by rendering dots together with the feature lines.

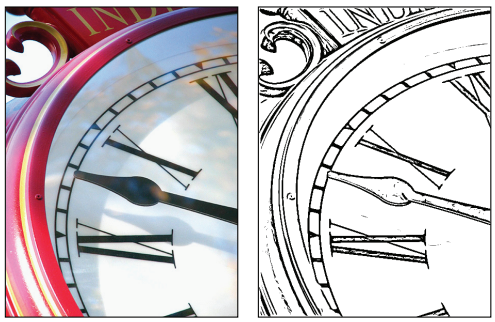
4. Preprocessing

4.1. Tone map construction

We use a grayscale image $I(\mathbf{x})$ as input, where $\mathbf{x} = (x, y)$ denotes an image pixel. In case I is too dark or of low contrast, we perform brightness adjustment and/or contrast stretching on I. The resulting image is denoted $\mathcal{T}(\mathbf{x})$, which we call a *tone map*. We let $\mathcal{T}(\mathbf{x})$ range in $[0, 1]$. The tone map is used to control the tone-related dot attributes such as size and/or intensity.

4.2. Line map construction

From $\mathcal{T}(\mathbf{x})$, we find a set of feature lines that will be used to guide stippling. We employ the line drawing method presented by Kang et al. [KLC07], which produces stylistic and coherent lines. Fig. 5 shows an example input and the resulting line drawing image. We denote the resulting black-and-white *line map* by $\mathcal{L}(\mathbf{x}) \in \{0, 1\}$, where 0 (black) represents line.



(a) Input (b) Line map

Figure 5: *Line map construction*

5. Feature Flow Construction

By a 'feature flow', we mean a smoothly varying vector field that describes the direction of the nearest feature for each pixel. We perform distance transform from the feature lines to achieve this. The constructed distance field is then adaptively smoothed to reduce potential visual artifacts in the rendering result. Finally, we extract offset lines from the smoothed distance field.

5.1. Distance transform

The line map $\mathcal{L}(\mathbf{x})$ may contain some isolated black pixels due to image noise. We first remove these noise pixels by binary morphological opening on $\mathcal{L}(\mathbf{x})$ with a circular structuring element of radius $1 \sim 3$.

We then apply jump flooding method [RT06] to construct a distance field, denoted $\mathcal{D}(\mathbf{x})$, using the black pixels in $\mathcal{L}(\mathbf{x})$ as seeds (zero distance). Jump flooding is so named as it propagates information in the manner of 'jumping' from pixel to pixel. Let k denote the jump (step) size. In each round of jumping, each pixel $\mathbf{x} = (x, y)$ inspects nine pixels located at $(x+i, y+j)$ where $i, j \in \{-k, 0, k\}$, and computes distances to their associated seeds. The minimum of these distances and the corresponding seed are recorded at $\mathcal{D}(\mathbf{x})$. This jumping is repeated by halving k in each round. Therefore, the distance field is completed after $\log n$ rounds for an image of size $n \times n$. Since this algorithm operates locally on each pixel, it can be dramatically accelerated when implemented on a GPU. More importantly, it provides constant time complexity, regardless of the number of seeds. Interested readers are referred to [RT06] for more details on the algorithm. Fig. 6b shows a distance field obtained by jump flooding.

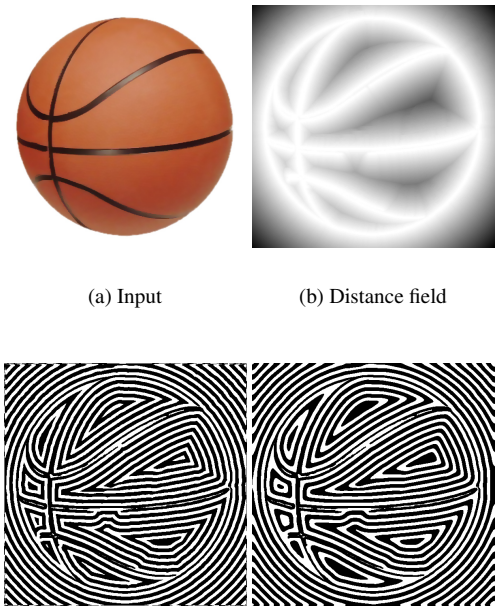
The computed distance field serves as our *feature flow*. Note a distance field can be viewed as a vector field, where each pixel $\mathbf{x}$ is associated with a vector pointing to the neighboring pixel that has the same distance value. This vector represents the feature direction at $\mathbf{x}$.

5.2. Adaptive smoothing of the distance field

When we obtain an offset line image, a crude distance field may result in some undesirable visual artifacts such as wobbly lines and sharp corners (see Fig. 6c). It is desirable to reduce such artifacts as they may become noticeable in the final rendering and hence divert the viewer's attention. We resolve this by obtaining offset lines from $G_\sigma * \mathcal{D}(\mathbf{x})$, the Gaussian-smoothed distance field. We use the distance value at each pixel $\mathbf{x}$ to determine its smoothing kernel size $\sigma(\mathbf{x})$. That is, $\sigma(\mathbf{x}) = c \cdot \min\{\mathcal{D}(\mathbf{x})/\mathcal{D}_c, 1\}$, where $\mathcal{D}_c$ is the distance for the maximum kernel size c. In our implementation, c = 9 and $\mathcal{D}_c$ = 120. The reason for the distance-adaptive smoothing is because it is desirable to closely follow the structure of features in a low-distance area but less so elsewhere. The adaptive distance filtering results in a set of smooth offset lines and rounded corners (see Fig. 6d), which could help reduce visual artifacts in the final stipple illustrations.

5.3. Extracting offset lines

Given the smoothed distance field, $G_\sigma * \mathcal{D}(\mathbf{x})$, the offset lines are extracted by regularly sampling distance values. Let m denote the sampling interval, and l denote the desired width



(a) Input (b) Distance field

(c) Offset lines (d) Offset lines

Figure 6: *Feature flow construction. In (c) and (d), offset lines are drawn in black.*

of each offset line. To create an offset line image, we mark a pixel if its distance is within $[m \cdot i - l/2, m \cdot i + l/2]$, where i is a positive integer. Typical values we use are $m = 6, l = 4$. As shown in Fig. 6c and 6d, the collection of offset lines clearly describe the feature flow. Such an offset line image is used to line up dots within the set of white lanes, called *offset lanes*. The width of each offset lane, denoted o, is obtained as $o = m - l$. The use of a smaller o results in stricter alignment of dots.

6. Dot Optimization

Given the offset lines and feature lines, the system now performs actual stippling. We first scatter the initial distribution of dots, which is then optimized by the Lloyd relaxation. Here we use the offset lines to constrain the Lloyd algorithm so that the dots strictly follow the feature flow.

6.1. Dot initialization

For fast initialization, we regularly sample the image pixels (in both x and y directions), except on the feature lines where no dots are sampled. The sampling interval, denoted r, is automatically computed using m, the distance between the adjacent offset lane centers, such that the entire image can be filled with a disjoint set of circles of diameter m (that is, typically $r = m$). By matching the sampling interval of dots with that of the offset lanes, we can force the distances between the adjacent dots to be roughly identical in both *intra-lane* and *inter-lane* directions, after relaxation. The intra-lane direction corresponds to the feature flow direction, and the inter-lane direction refers to its perpendicular direction.

6.2. Constrained Lloyd relaxation

The initial dots then go through the Lloyd relaxation. That is, we iterate the process of: (1) constructing Voronoi diagram from dots, (2) moving dots to the updated centroids of Voronoi cells.

6.2.1. Constructing a Voronoi diagram

For constructing a Voronoi diagram, we again use the jump flooding algorithm, this time however with respect to the dots as seeds. Note the jump flooding algorithm creates not only distance field but also Voronoi diagram as it records which seed each pixel is associated with. As opposed to the conventional polygon-based z-buffering algorithm for constructing Voronoi diagram [HKL*99], jump flooding provides constant time complexity regardless of the number of dots. We use the Euclidean distance in creating a Voronoi diagram with jump flooding.

6.2.2. Updating centroids

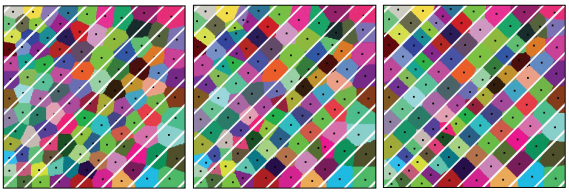
The centroid $\mathbf{c}$ of a Voronoi cell is computed as follows:

$$\mathbf{c} = \rho^{-1} \sum_i w_i \cdot \mathbf{x}_i \qquad (1)$$

where $\mathbf{x}_i$ denotes the i-th pixel in the cell, w_i associated weight, and $\rho = \sum_i w_i$ a weight normalization term. In the basic Lloyd algorithm, $w_i = 1$ for all pixels in the cell.

In our approach, we use offset lines as constraints so as the Voronoi cells to line up with those lines (see Fig. 7). For this we modify Hausner's idea which he used to push rectangular tiles away from the feature lines [Hau01]. When a part of a Voronoi cell is occluded by an offset line, we remove the part in computing the updated centroid of the cell. We can easily achieve this by setting $w_i = 0$ in Eq. 1 for all the pixels within offset lines. If a Voronoi cell is divided by an offset line, the resulting non-occluded pieces may occupy different offset lanes. Among the pieces, we pick the closest one to the previous centroid, then compute the new centroid of the cell using this selected piece only. That is, in Eq. 1, we set $w_i = 1$ for all pixels in the selected piece, and $w_i = 0$ for all other pixels in the cell (including the offset-line pixels). However, this case of multiple pieces hardly happens in our setting where the dot sampling interval r is the same as the offset line sampling interval m.

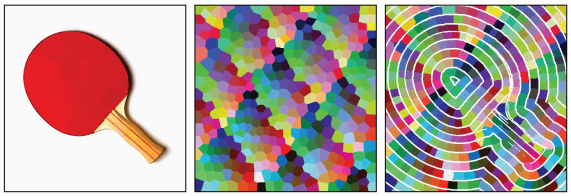
This strategy has the effect of moving the centroid towards to the center of an offset lane. It also ensures that once the centroid moves into a particular offset lane, it stays in there. Fig. 8 shows the evolution of an entire Voronoi diagram, constrained by the offset lines. Note our offset-line-based constraint is stronger than Hausner's in that it 'strictly' aligns



(a) (b) (c)

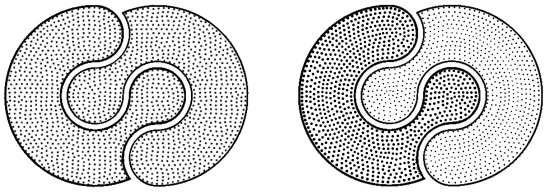
Figure 7: *Voronoi cell alignment using offset lines. Constrained Lloyd relaxation pushes each cell's centroid towards the center of its associated offset lane, as shown from (a) to (c). For visualization purpose, the offset lines are drawn thinner than they actually are.*

dots with the nearest feature lines (see Fig. 9 for comparison).



(a) Input (b) Initial configuration (c) Final configuration

Figure 8: *Evolution of a dot distribution. See how the constrained Lloyd algorithm re-organizes the initial dots to follow the feature flow.*



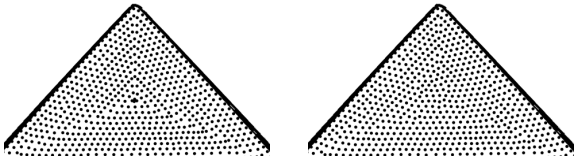
(a) Hausner's method (b) Our method

Figure 9: *Comparison with Hausner's method. Our method produces a better aligned dot distribution.*

We provide an additional control to prevent the dots from looking 'too structural' especially in the middle of an area far away from the feature lines. We accomplish this by weakening the offset-line constraints in the middle area (see Fig. 10). That is, instead of setting $w_i = 0$ for all offset lines, we give weights proportional to their distance values. The weight w_i for an offset-line pixel $\mathbf{x}_i$ is thus redefined as:

$$w_i = \min\{\mathcal{D}(\mathbf{x}_i)/\mathcal{D}_w, 1\}, \tag{2}$$

where $\mathcal{D}_w$ is the distance value for which the offset-line constraints have no effects. By default, $D_w = 100$. Since higher weight is given to the offset lines in a distant area, their constraints get weaker and the Voronoi cells there should align less strictly along the offset lines (see Fig. 10b).



(a) No control (b) Control with distance

Figure 10: *Adaptive control of the influence of offset lines*

Besides the offset lines, we also use the feature lines (i.e., black lines in $\mathcal{L}(\mathbf{x})$) as constraints so that the dots do not directly overlap with those lines and thus we can protect the features better. For this we set $w_i = 0$ in an area enclosing the feature lines, where $\mathcal{D}(\mathbf{x}) < \xi$ (with default value $\xi = 2$).

6.2.3. Iteration

We typically iterate the Lloyd algorithm t_1 times without offset-line constraints, then t_2 times by alternating Lloyd algorithm with/without constraints (by default $t_1 = 10, t_2 = 30$). The first t_1 iterations is for spreading the initial set of dots over the image. The reason for toggling the constraints on and off afterwards is similar: to avoid clustering and spread the dots more evenly across the image.

7. Rendering

Once the locations of dots have been finalized, they are rendered as black circles, together with the feature lines. The dot size is inversely proportional to $\mathcal{T}(\mathbf{x})$, meaning small dots are placed on bright area, and big dots on dark area. The dot size s at pixel $\mathbf{x}$ is thus a function of $\mathcal{T}(\mathbf{x})$, which we define as follows:

$$s(\mathbf{x}) = s_{max} \cdot (1 - \mathcal{T}(\mathbf{x}))^{\gamma}, \tag{3}$$

where s_{max} is the maximum possible dot size and depends on the rendered image size. γ is used to incorporate gamma correction for tone control. With a larger value of γ, we can have higher contrast of tones in the stippled image ($\gamma = 1.2$ by default). In the brightest area, where $s(\mathbf{x})$ is less than a small threshold, no dot is drawn. To improve the quality for printing, it is often a good idea to use a set of huge dots rendered in an expanded image space and scale down the rendered image. We typically render on an image which is six times bigger than the input.

8. Results

In Fig. 11, we show various results can be obtained from an input image using different values of parameters, m (offset lane interval) and γ (gamma correction value). The results

demonstrate that the overall stipple density and tone contrast can be intuitively controlled by m and γ, respectively, while preserving the feature-guidance of the stipple distribution.

(a) $m = 5(\gamma = 1.3)$ (b) $m = 6(\gamma = 1.3)$ (c) $m = 7(\gamma = 1.3)$

(d) $\gamma = 1.0(m = 6)$ (e) $\gamma = 1.5(m = 6)$ (f) $\gamma = 2.0(m = 6)$

Figure 11: *Parameter control*

Fig. 13 shows the stipple illustrations we produced from photographs in Fig. 12. Note the stipple dots are clearly shown to align with the feature flow, while the sizes of dots gradually vary according to the tone.

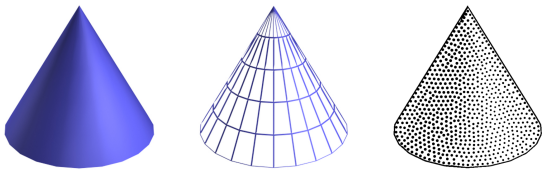
We implemented and experimented with our system on a Pentium 4 PC with an nVIDIA GeForce 8800 GT graphics card. For a 640×480 image, it takes about one minute to create a stipple illustration (using CPU implementation of jump flooding). With default parameters, typically $8,000 \sim 12,000$ dots are created for a 640×480 image. The number of dots, however, does not directly affect the performance of stippling, largely due to the use of jump flooding for distance computation.

9. Discussion and Future work

When artists create hedcut illustrations, they often use an imaginary 3D surface wrapping around the target shape, and place dots along the feature-following contours regularly sampled on the surface (see Fig. 1). Similarly, the engraving scheme of Ostromoukhov [Ost99] allows users to create, deform, and place uv-parametric surfaces such that they fit the given facial structure, then the system automatically places engraving lines along u-contours and v-contours of the surfaces (Fig. 14b illustrates this scenario for a cone). In this case, the directions of lines (or dots) are more faithful to the 3D geometry of the face, and thus the resulting illustration provides more convincing look.

Figure 12: *Input photographs*

As our method does not rely on any 3D information, the resulting dots may not properly reflect the actual 3D structure of the surface, especially when the surface does not have any interior texture or features lines (see Fig. 14b). Moreover, our method aligns dots along the feature lines, but not necessarily along their perpendicular directions. Along with these issues, a quantitative analysis of our result in comparison with professional hedcut illustrations and other computerized stipple renderings could make a valuable theme for future research, as exemplified by the recent work of Maciejewski et al. [MIA*08].



(a) Input (b) Wire-frame rendering (c) Our result

Figure 14: *Lacking the sense of 3D. Our stippling result may not properly reflect the 3D geometry of the surface.*

For a good-quality print, the stipple illustration must be generated using an appropriate number of dots with proper size and density, so that it fits the size and resolution of the printing area. Otherwise its aesthetic merit (as a dot illustration) could be diminished. One way to resolve this is to support *resolution independence* (i.e., progressive zoom-in and zoom-out while maintaining the apparent dot size and density) as in [KCODL06]. In our case, we should also maintain the directionality of dots, for which some hierarchical structuring of feature flow is in order.

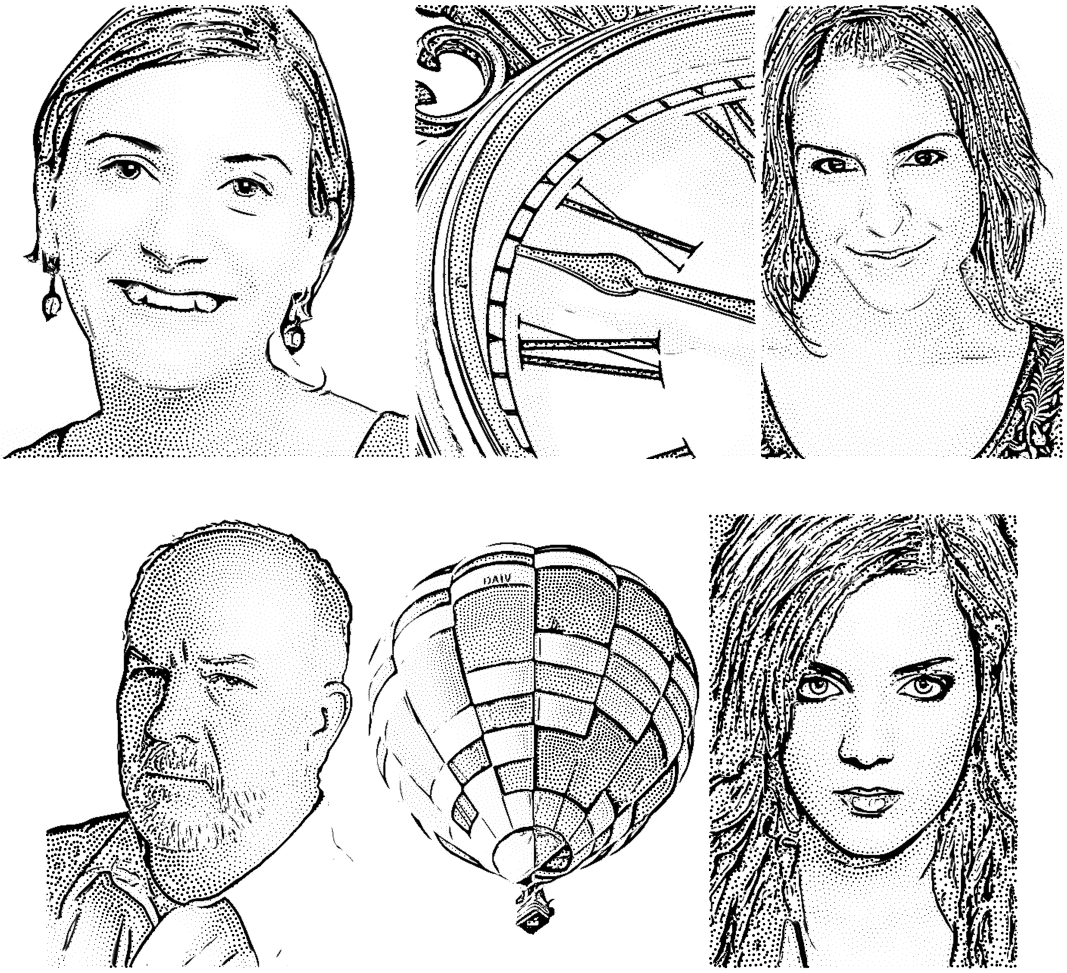


Figure 13: *Our stippling results*

Another possible future work may involve extension of our scheme to 3D objects or video. 3D feature-guided stippling calls for the development of an algorithm to create a feature flow field on an object surface, as in 3D hatching [HZ00, ZISS04]. Video stippling is a non-trivial problem as it poses a different set of challenges often seen in stroke-based animation, such as providing temporal coherence of dots between frames, as well as avoiding temporal artifacts including shower door effect, flickering, and swimming dots.

References

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (1986), 51–72.

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. In *Proc. ACM SIGGRAPH* (2003), pp. 287–294.

[DHVOS00] DEUSSEN O., HILLER S., VAN OVERVELD K., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum 19*, 3 (2000), 40–51.

[FS76] FLOYD R., STEINBERG L.: An adaptive algorithm for spatial grey scale. *Proc. Soc. Inf. Display 17*, 75-77 (1976).

[Hau01] HAUSNER A.: Simulating decorative mosaics. In *Proc. ACM SIGGRAPH* (2001), pp. 573–578.

[HKL*99] HOFF K., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. ACM SIGGRAPH* (1999), pp. 277–286.

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proc. ACM SIGGRAPH* (2000), pp. 517–526.

[KCODL06] KOPF J., COHEN-OR D., DEUSSEN O., LISCHINSKI D.: Recursive wang tiles for real-time blue noise. In *Proc. ACM SIGGRAPH* (2006), pp. 509–518.

[KLC07] KANG H., LEE S., CHUI C. K.: Coherent line drawing. In *Proc. Non-Photorealistic Animation and Rendering* (Aug. 2007), pp. 43–50.

[MIA*08] MACIEJEWSKI R., ISENBERG T., ANDREWS W., EBERT D., SOUSA M., CHEN W.: Measuring stipple aesthetics in hand-drawn and computer-generated images. *IEEE Computer Graphics and Applications 28*, 2 (2008), 62–74.

[Mou07] MOULD D.: Stipple placement using distance in a weighted graph. In *Proc. Computational Aesthetics* (2007), pp. 45–52.

[ODJ04] OSTROMOUKHOV V., DONOHUE C., JODOIN P.: Fast hierarchical importance sampling with blue noise properties. In *Proc. ACM SIGGRAPH* (2004), pp. 488–495.

[Ost99] OSTROMOUKHOV V.: Digital facial engraving. In *Proc. ACM SIGGRAPH* (1999), pp. 417–424.

[Ost01] OSTROMOUKHOV V.: A simple and efficient error-diffusion algorithm. In *Proc. ACM SIGGRAPH* (2001), pp. 567–572.

[Ost07] OSTROMOUKHOV V.: Sampling with polyominoes. In *Proc. ACM SIGGRAPH* (2007). Article No. 78.

[RT06] RONG G., TAN T.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proc. Symposium on interactive 3D graphics* (2006), pp. 109–116.

[Sec02] SECORD A.: Weighted voronoi stippling. In *Proc. Non-Photorealistic Animation and Rendering* (2002), pp. 37–43.

[SWHS97] SALISBURY M., WONG M., HUGHES J., SALESIN D.: Orientable textures for image-based pen-and-ink illustration. In *Proc. ACM SIGGRAPH* (1997), pp. 401–406.

[ZISS04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T.: High quality hatching. In *Proc. Eurographics* (2004), pp. 421–430.