Stippling by Example

SungYe Kim* Ross Maciejewski* Tobias Isenberg† William M. Andrews‡ Wei Chen§ Mario Costa Sousa¶
David S. Ebert *
* Purdue University † University of Groningen ‡ Medical College of Georgia
§ Zhejiang University ¶ University of Calgary

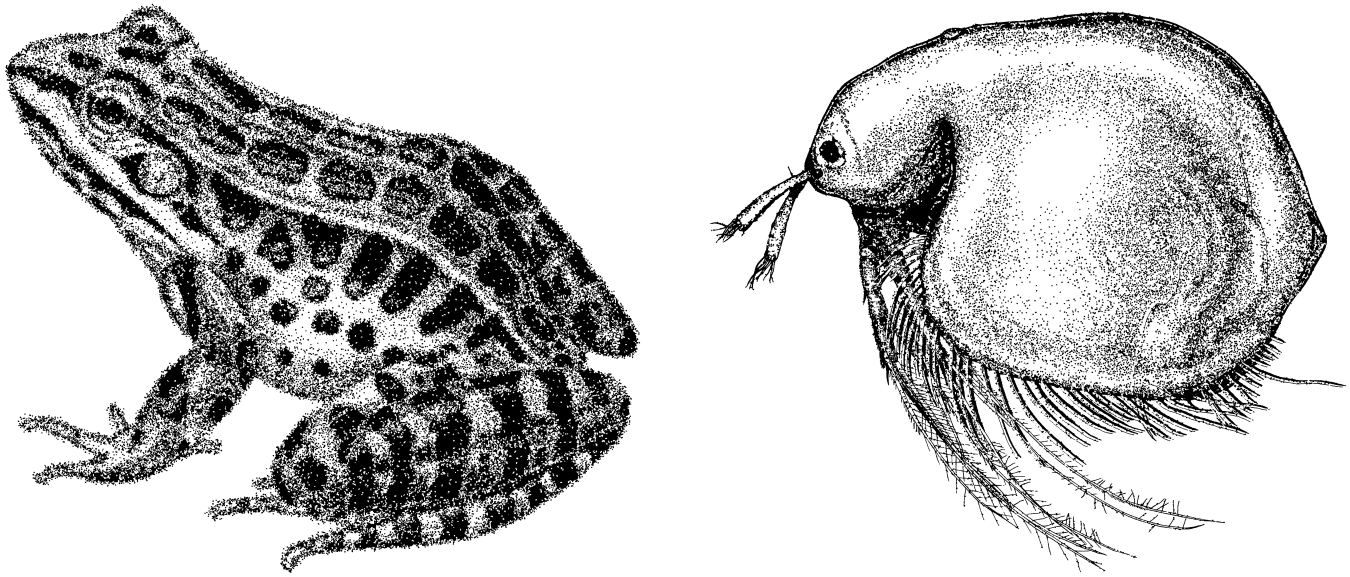

Figure 1: *Our results generated by synthesizing textures based on example textures extracted from stippling artist' tone maps.*

Abstract

In this work, we focus on stippling as an artistic style and discuss our technique for capturing and reproducing stipple features unique to an individual artist. We employ a texture synthesis algorithm based on the gray-level co-occurrence matrix (GLCM) of a texture field. This algorithm uses a texture similarity metric to generate stipple textures that are perceptually similar to input samples, allowing us to better capture and reproduce stipple distributions.First, we extract example stipple textures representing various tones in order to create an approximate tone map used by the artist. Second, we extract the stipple marks and distributions from the extracted example textures, generating both a lookup table of stipple marks and a texture representing the stipple distribution. Third, we use the distribution of stipples to synthesize similar distributions with slight variations using a numerical measure of the error between the synthesized texture and the example texture as the basis for replication. Finally, we apply the synthesized stipple distribution to a 2D grayscale image and place stipple marks onto the distribution, thereby creating a stippled image that is statistically similar to images created by the example artist.

CR Categories: I.3.m [Computer Graphics]: Miscellaneous—Non-Photorealistic Rendering

Keywords: Computer-generated stippling, stippling by example, texture analysis and synthesis, stipple mark distribution, statistical similarity.

*{inside|rmacieje|ebertd}@purdue.edu
†isenberg@cs.rug.nl
‡bandrews@mcg.edu
§chenwei@cad.zju.edu.cn
¶mario@cpsc.ucalgary.ca

1 Introduction

Stippling is an artistic technique that relies on numerous small, repetitive marks (stipples) to visually describe forms and objects [Hodges 1989]. It is a black-and-white technique, i. e., 100% black marks are made on a 100% white ground (or vice versa); there are no gray marks. However, because the size, shape and density of the marks can be varied, shades of gray are perceived within the stippled image. As such, stippling is capable of capturing a very wide dynamic range of tones, from white to black. While stippling is a well defined technique, choices in mark size, density, and irregularities in shape can all lead to slight variations, giving rise to various styles of stippling.

In the graphics community, many algorithms have been developed in an attempt to approximate hand-drawn stippling (e. g., Deussen et al. [2000], Secord [2002], Lu et al. [2003], Kopf et al. [2006]), most relying on a procedural random distribution of stipple positions rendered as round marks. Recently, researchers started comparing computer-generated pen-and-ink techniques to illustrations hand-drawn by professional artists, including stippling. Isenberg et al. [2006] used a qualitative pile-sorting technique to determine how participants understand and assess both hand-drawn and computer-generated pen-and-ink illustrations. This study found that participants, in general, could correctly distinguish between

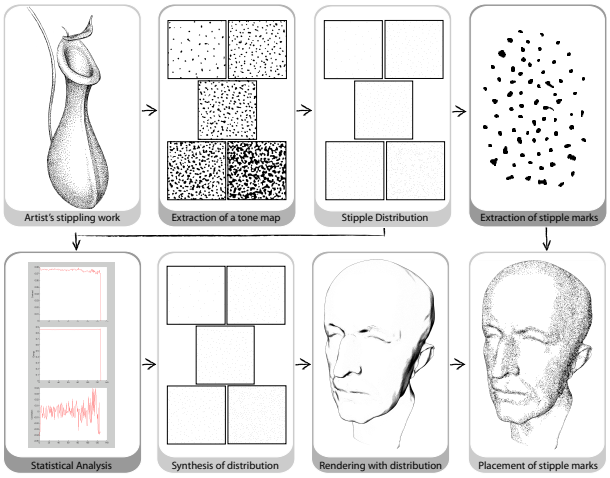

Figure 2: *Conceptual diagram of our stippling system.*

hand-drawn and computer-generated images. In addition, Isenberg et al. found several issues associated directly with stippling techniques. One was that participants often noticed that computer-generated stipple images employed a higher number of stipples per unit area, and the stipple marks themselves were too regular in comparison to their hand-drawn counterparts. Further work was done on directly quantifying these differences by Maciejewski et al. [2008]. This work employed statistical texture measures to quantify the differences between hand-drawn and computer-generated stipples and suggested that such texture measurements could be employed in texture generation. Both studies indicate that subtle variations of stippling are detectable and affect how images are perceived.

Based on observations and discussions by Isenberg et al. [2006] and Maciejewski et al. [2008], we have developed a novel stippling algorithm to capture and reproduce these stylistic differences. The goal of our work is to replicate the appearance of hand-drawn stipple textures in scientific illustration. Our system takes a sample stipple image and extracts example stipple textures. These extracted samples are then used to generate new statistically similar textures, which are in turn used to render 2D images in the same style as the artist's example image. Sample results of our stippling algorithm are shown in Figure 1. To summarize, our method, shown in Figure 2, consists of six major steps:

1. Automatic extraction of an artist's tone map from an example stipple image.
2. Extraction of the stipple distribution and stipple marks from the tone textures.
3. Statistical analysis of texture properties.
4. Synthesis of stipple distribution to match the statistical texture properties.
5. Given any gray-scale image, render the image using the synthesized textures including stipple distributions.
6. Placement of stipple marks.

2 Related Work

There are a number of approaches in NPR that attempt to replicate stippling using computer graphics. Generally, it is possible to distinguish between techniques by the model representation being used (i. e., images, surfaces, or volumes) as well as by the type of output they produce: short strokes, analytic stipple points or a pixel matrix. In this work, we focus on 2D image stippling.

Image-based stippling uses an image, typically in gray scale, as input and tries to reproduce its tonal properties with stipple points by choosing appropriate point placements, sizes, and shapes. Early techniques used iterative relaxation approaches based on Lloyd's method which uses centroidal Voronoi diagrams to distribute the generated stipple points. Deussen et al. [2000] presented an interactive, tool-based method which allows users to manipulate and adjust an initial point distribution. Secord [2002] showed an automatic technique using weighted centroidal Voronoi diagrams. Hiller et al. [2003] later extended the Lloyd's method to place line segments or polygons by using their Voronoi diagrams. Dalal et al. [2006] further extended the centroidal Voronoi diagram technique to be based on area and a primitive's perimeter for re-centering, allowing them also to stipple with arbitrary shapes.

Unfortunately, these image based approaches tend to exhibit certain artifacts, such as the tendency to form lines of stipple points. Many methods attempt to overcome these issues by incorporating more randomness into the stipple placement. Schlechtweg et al. [2005] used autonomous agents (RenderBots) to place stipple points based on an input image using relatively simple behaviors such as trying to go to dark areas and avoiding other RenderBots. Kopf et al. [2006] used progressive and recursive Wang tiles to rapidly produce point distributions that exhibit blue noise qualities. Mould [2007] demonstrated how to use path search in a weighted regular graph to produce stipple point distributions that are more irregular and follow linear features, albeit with a minimum point distance. Vanderhaeghe et al. [2007] presented a point distribution method for stroke-based rendering including stippling considering temporal coherence for animation of 3D scenes and video input. Directional stippling introduced by Kim et al. [2008] created points aligned to edge features within an image in a structured manner that is different from previous stippling research.

Our work differs from previous work in that we focus on synthesizing stipple distributions from example stipple textures while reusing stipple marks. Tonal textures are extracted, the stipple distributions of the tone textures are recreated based on texture statistics, and the distributions are used for stipple rendering while preserving the local tonal qualities of the image. In terms of an example-based method, Barla et al. [2006a] focused on hatching and stippling by example through analysis and synthesis of stroke patterns based on the similarity between neighborhoods by Barla et al. [2006b]. Ijiri et al. [2008] introduced a procedural system for the arrangement of 2D elements that is also based on matching neighborhoods by local growth and relaxation methods. In contrast, we use the analysis and synthesis of statistical characteristics of textures. Hatching by example by Jodoin et al. [2002] presented a conceptually similar approach to ours, which collects hatching strokes from an example image, generates patches of synthetic strokes based on a statistical model, and uses them to render 2D images or 3D objects. However, they mainly focused on the synthesis of example strokes remaining other parts such as automatically extracting strokes as a long term goal. In contrast, we deal with the entire approach for example-based stippling; from extracting example tone textures to rendering images with our synthesized textures.

3 Stipple Extraction, Analysis and Synthesis

As previously stated, stippling is a technique that relies on repetitive marks to create perceived shades of gray within an image, and artists commonly use a *tone map* as a reference to block out an image (determine the broad areas of same or similar tone). Figure 3

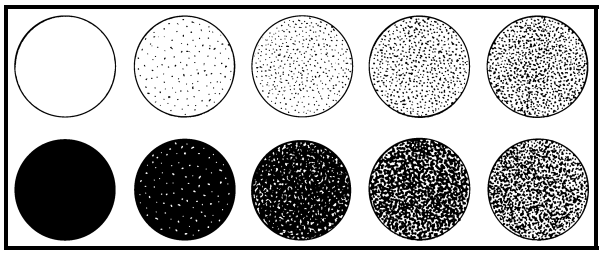

Figure 3: *Tone maps created by artist William M. Andrews (© William M. Andrews, used with permission). From left-top, it starts from tone 0 to tone 9 in a clockwise direction.*

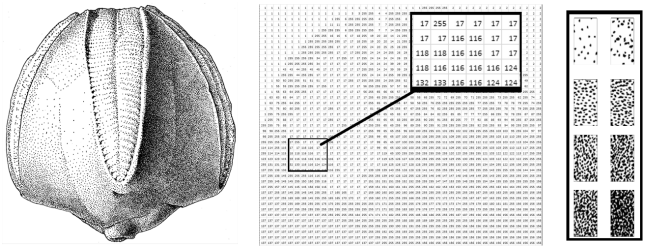

Figure 4: *Left: blastoid shell (© Emily S. Damstra, used with permission); middle: cell component values from our extraction algorithm; right: extracted sample textures. Cells with neighboring numbers join together to create a large sample from which to extract uniform texture tones.*

shows a typical stippling tone map with 10 levels (including black and white). When stippling, the artist will naturally dither the edges between two adjacent tones to create a seamless transition.

Our method (depicted in Figure 2) utilizes the concept of an artist's tone map, taking an example stipple image and automatically extracting various tone levels. In each example tone, we capture the statistics of the artist's rendering style in terms of the distribution of stipple marks. We synthesize a new texture to capture the artist's stipple distribution. Further, to maintain the stipple shape, we also extract individual stipples from the example input and place these marks based on the simulated distribution, thereby maintaining the stipple distribution, size and shape used by the artist. Once new synthesized textures are generated from the example distribution, our system can take a grayscale image as input and stipple the image in a style similar to the given stipple artist.

3.1 Extracting a Tone Map

In order to capture an artist's tone map, we use a segmentation method. We take an artist's stipple drawing and divide the image into equally sized $N \times N$ blocks. (N depends on the size of the artist's stipple image, and should be small enough to represent a single tone. In Figure 4, we use N=30.) The average gray level in each block is then calculated as the sum of all gray values over the total number of pixels in the block. Since we use a stipple image containing only black or white pixels, we can calculate the average gray level as the ratio of black and white pixels in the block.

To group blocks that have similar average gray levels within a threshold, T_1, we perform a connected component analysis using an 8-point neighborhood system. During the analysis, each block is assigned a number in order to identify a segmented class of blocks, and blocks assigned with the same number form one segmented region. We then sort the segmented regions by their average gray level and determine the difference of gray levels between two neighboring regions. If the tonal values assigned to the segmented regions are separated evenly within a threshold, T_2 we extract a rectangular texture from that segmented region. Figure 4 (middle) shows the segmented regions identified by class numbers for a hand-drawn stipple image (Figure 4 (left)) and shows eight example textures (Figure 4 (right)) extracted from the stipple image. Although our approach allows us to extract example textures from an artist's stipple image, this cannot be applied to all stipple images because our method assumes that there exists large enough regions in the stipple image for each tone level. In addition, the initial generation of blocks may split across two apparent tones. Therefore, extraction should be somewhat controlled by adjusting the size of a block and the two thresholds. In the cases where our extraction method fails, the tones may be manually extracted. In later parts of this paper, we use William M. Andrews' tone maps (Figure 3) as the example stippling textures to explain our algorithm.

3.2 Stipple Distribution and Marker Extraction

The next step in our process is to separate the stipple marks from their distribution. To this end, we analyze the example stipple texture and create a new texture containing only the stipple distribution. Before starting an algorithmic method, we classify the example stipple textures into three groups: light, merged, and dark level. In the light level, example stipple textures have separate black stipple marks on a white background (tones 1 to 4 in Figure 3), whereas in the dark level, they contain separate white stipple marks on a black background (tones 7 to 8). In the merged level, several stipple marks merge together resulting in blobby shapes that are difficult to analyze (tones 5 to 6). Tone 0 and 9 are pure white and black respectively. In this work, we consider textures in this merged level to have black stipple marks on a white background.

Our method for extracting the stipple distribution uses two different techniques, depending on whether or not the stipples in a given tone level can be considered as separate or conjoined entities. For textures containing individual (non-merged) stipples, we perform a connected component analysis using an 8-neighborhood system to segment stipple marks. For each segmented stipple mark, a center position is computed. The segmented stipples are then stored for future use in the marker placement step (Section 4.2). The resulting texture, the *example distribution texture*, includes all representative positions of stipple marks as distribution information.

To create the example distribution texture in the merged level, we use the constrained Lloyd algorithm proposed by Kim et al. [2008] that aligns points to a feature area in an image. In our work, the constraint is black stipple marks merged in an example stipple texture. We begin with an appropriate number of initial points selected regularly, and then we generate the centroidal Voronoi diagrams to spread the initial points. During optimization using the constrained Lloyd algorithm, the points move toward the black stipple areas as shown in Figure 5 (right) (see [Kim et al. 2008] for details).

We also perform a simple geometrical analysis for use in the synthesis process (Section 3.4), calculating the average stipple size, S_{avg} and offsets between the positions of stipple marks: the minimum (O_{min}), maximum (O_{max}), and average (O_{avg}) geometrical distances from any stipple mark to the closest neighbor stipple mark.

3.3 Statistical Analysis

Once the tone maps are extracted and the stipple properties captured, the next step of our process is to use these example distribution textures to create synthetic distribution textures.

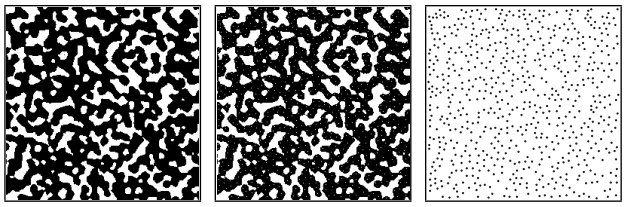

Figure 5: *Distribution extracted from an example texture in a merged level. Left is a tone 6 texture in William M. Andrews' tone maps. Middle shows the stipple centers (as white dots) found using the constrained Lloyd algorithm. Right is the extracted distribution texture with stipple centers represented as equally sized circles for visualization purposes.*

3.3.1 Texture Statistics

There are two common approaches to analyzing textures, the structural approach and the statistical approach. While the structural approach works well for regular patterns, the lack of discernible patterns in stippling requires using the statistical approach. There are many statistical texture analysis algorithms designed to represent textures for comparison; for example, GLCM [Haralick et al. 1973], Fourier power spectra, and texture spectra are some of the more common approaches. We have chosen the GLCM algorithm to analyze stipple textures because perceptual psychology studies [Julesz et al. 1973] have shown that GLCM closely matches levels of human perception. Furthermore, previous research [Davis et al. 1979; Gotlieb and Kreyszig 1990; Lohmann 1994; Copeland et al. 2001] has shown that GLCM is a powerful tool for texture analysis, synthesis, segmentation, and classification.

We use the GLCM algorithm to analyze texture statistics as opposed to more recent algorithms, such as advanced implementations of the Gray-Level Difference Histograms [Chetverikov 1994] and Gray-Level Aura Matrices [Qin and Yang 2005] or the use of joint texture statistics [Portilla and Simoncelli 2000]. While methods such as those are appropriate for texture analysis and synthesis, their ability to identify structures in textures is not needed because stipple artists try to avoid producing oriented textures or unintended patterns [Hodges 1989]. As such, methods that measure the anisotropy and texture symmetry are not necessary and may not be adequate for comparing stipple textures.

A GLCM [Haralick et al. 1973] is a two-dimensional array L in which the rows (r) and columns (c) represent a set of possible gray values G. The value of $L_{\vec{d}}[i,j]$ indicates how many times value i co-occurs with value j in a given spatial relationship defined by $\vec{d}$. If we set $\vec{d}$ to be a displacement offset vector $[dr, dc]$ where dr is the displacement in rows and dc is the displacement in columns, then the co-occurrence matrix $L_{\vec{d}}[i,j]$ for some image, I, is defined by

$$L_{\vec{d}}[i,j] = |\{[r,c] \mid I[r,c] = i \text{ and } I[r+dr, c+dc] = j\}| \quad (1)$$

Previous work in analyzing stipple textures [Maciejewski et al. 2008] demonstrated that GLCM statistics can be used to illustrate the differences between hand-drawn and computer-generated stipple images. The following three texture statistics were used: contrast, energy, and correlation:

$$Contrast = \sum_i \sum_j (i-j)^2 N_{\vec{d}}[i,j] \quad (2)$$

$$Energy = \sum_i \sum_j N_{\vec{d}}^2[i,j] \quad (3)$$

$$Correlation = \frac{\sum_i \sum_j (i-\mu_i)(j-\mu_j) N_{\vec{d}}[i,j]}{\sigma_i \sigma_j} \quad (4)$$

where $N_{\vec{d}}$ is a normalized GLCM for $\vec{d}$, μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column sums of the normalized GLCM for $\vec{d}$, $N_{\vec{d}}$.

3.3.2 GLCMs for Stipple Textures

Our algorithm (Figure 2) uses a sample stipple texture representing a single tone as an input. We refer to this input as the *example stipple texture*. We then modify the example stipple texture to contain only the stipple distribution resulting in the *example distribution texture* (as described in Section 3.2). Next, our algorithm synthesizes a new stipple distribution that is similar to the example distribution texture with slight variations, allowing us to generate various distributions from only one example distribution. To do this, our algorithm requires another input texture, denoted as the *seed texture*, that can be an empty white texture.

Given the example distribution and seed textures, the texture GLCMs are calculated. Since the input textures are bitonal images, we consider only two gray levels (black and white), meaning that all GLCMs calculated will be of size 2×2 matrices. We calculate $(n-1)$ GLCMs for the $n \times n$ example distribution and seed textures for a given displacement offset vector. Since the GLCM calculation depends on the direction of the offset vector if only one direction is used for the offset vector, undesired patterns can occur along other directions during synthesis. To reduce this issue we need to combine several different spatial relationships, linearly increasing the number of GLCMs. Here, we use three different spatial relationships; horizontal (0°), vertical (90°), and left-down diagonal (45°) for GLCM generation. Therefore, we calculate $3 \times (n-1)$ GLCMs for each of the $n \times n$ example distribution and seed textures, and use them for our synthesis process.

3.4 Synthesis of Stipple Distribution

Our synthesis step generates new distributions from the example distribution textures by iteratively minimizing the error between the example distribution texture and the seed texture's GLCMs while the current error between them is reduced within a maximum iteration. This is very time-consuming step if an accelerating method is not provided. In our work, combining geometrical information such as O_{min}, O_{max}, O_{avg}, and S_{avg} (computed while extracting the stipple distribution from the example texture in Section 3.2) into the synthesis process improves our performance by reducing the number of considered pixels.

Traditional 2D texture synthesis techniques can be roughly grouped into two categories: non-parametric and parametric texture synthesis. The former aims to directly fetch samples from the input texture in either a per-pixel [Efros and Leung 1999; Wei and Levoy 2000] or a per-patch approach [Liang et al. 2001; Cohen et al. 2003; Kwatra et al. 2003]. Parametric texture synthesis constructs a generative model with a set of parameters to guide the synthesis process. Representative work includes histogram matching [Heeger and Bergen 1995], minimum entropy statistics [Zhu et al. 1997] and GLCM/GLAM-based techniques [Copeland et al. 2001; Qin and Yang 2005]. These works synthesize new textures by matching the corresponding joint statistics of the input and output images.

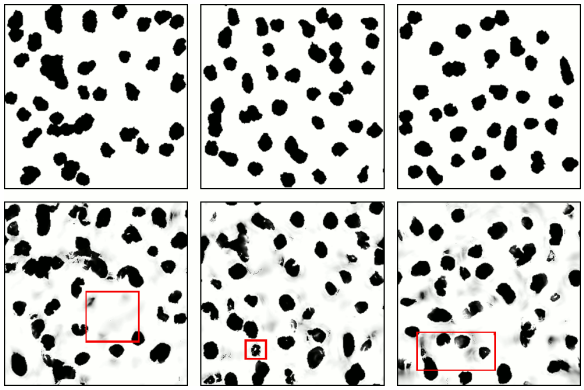

Figure 6: *Texture synthesis of example textures using Portilla and Simoncelli's [2000] method. Top: example stipple textures; bottom: synthesized texture after 25 iterations. Obvious artifacts are denoted with red rectangles in the synthesized textures.*

While such an approach fails to properly capture the stipple shape, the properties captured are sufficient. Other issues arise when trying to capture stipple shape in non-parametric models. For example, Figure 6 shows the texture synthesis results using the method developed in Portilla and Simoncelli [2000]. The results are representative of the known cases where the technique fails (e. g., a set of ellipses in which the synthesis fails to close the ellipses). Note the open stipples and smudges in the synthesized textures. As such, synthesizing stipple textures with appropriate statistics is not as straightforward as it may seem.

3.4.1 GLCM Error Minimization

We employ the average co-occurrence error (ACE) [Copeland et al. 2001] as a metric to minimize error between the GLCMs because it has been found to be highly correlated with human judgments of the visual distinctness between textures [Copeland and Trivedi 1998]:

$$ACE = \frac{1}{T_{NGLC}} \sum_{\vec{d} \in D} \sum_{i=0}^{1} \sum_{j=0}^{1} |N_{t,\vec{d}}[i,j] - N_{s,\vec{d}}[i,j]| \qquad (5)$$

where T_{NGLC} is the number of displacements, D is the set of displacement offset vectors (see [Copeland et al. 2001] for details), and $N_{t,\vec{d}}[i,j]$ and $N_{s,\vec{d}}[i,j]$ are the normalized GLCM values of the example distribution texture and the synthesized texture respectively for i, j, and $\vec{d}$ in the current iteration.

The ACE calculation is then performed for each GLCM directional pair. We calculate the ACE between the GLCMs calculated with a horizontal offset vector for both the example distribution texture and the seed texture, and repeat this calculation for other displacement offset vectors used in the GLCM calculations. This maximizes texture similarity in three directions and reduces the possibility of unwanted patterns emerging. Using the ACE metric, we employ the Metropolis Spin-Flip (MSF) algorithm in Metropolis et al. [1953] to modify the seed texture. In the MSF algorithm, the gray value of a random pixel, which is chosen from our seed texture, is changed from either black to white or white to black. Since we are now modifying the seed texture, it is more appropriate to refer to it as the synthesized texture. If the resultant ACE between the example distribution texture and the newly synthesized texture is lower than the ACE between the example distribution texture and the previously synthesized texture for all displacement offset vectors, the pixel remains flipped, otherwise, the pixel flips back and the algorithm proceeds to the next randomly chosen pixel until all pixels in the newly synthesized texture are considered. This pixel flipping reduces the error between the GLCM values of our example distribution texture and the synthesized texture, resulting in a newly synthesized texture with a minimum error in terms of the GLCM-based characteristics such as Equation (2), (3) and (4). This also indicates our algorithm is a discrete grid-based method assuming that all stipples are placed on discrete pixel positions such as black pixels on a white background or white pixels on a black background.

3.4.2 Combining geometrical information

During the GLCM error minimization phase, the pixels in the newly synthesized texture are chosen randomly for flipping; however, we constrain this random choice to maintain the average offset (O_{avg}) computed in Section 3.2. Once a random pixel is chosen, a stipple region around the pixel is determined with a radius D as follows:

$$D = O + R$$
$$O = \{O_{min}, O_{avg} \text{ or } O_{max}\}$$

$$R = \begin{cases} random(0, O_{avg}\text{-}O_{min}) & \text{if } O{=}O_{min} \\ 0 & \text{if } O{=}O_{avg} \\ random(O_{avg}\text{-}O_{max}, 0) & \text{if } O{=}O_{max} \end{cases}$$

where D is the radius of a circle for a stipple region, and O is assigned one of three offsets. The offset assigned is chosen based on a probability distribution (we use 0.1, 0.1, and 0.8 for minimum, maximum, and average offsets, respectively). R is a random factor based on the offset selected. Thus, D has the range of O_{min} to O_{max} and tries to be close to O_{avg}. The offset information (O_{min}, O_{avg}, and O_{max}) is updated whenever new stipple distribution is placed into the synthesized texture. If the random pixel is flipped in the MSF algorithm, we set all pixels within the radius D of the newly flipped pixel to "considered," meaning that the pixels cannot be chosen for flipping during the current iteration. As such, we can avoid random selections within any stipple regions. However, if all pixels become marked as considered, no pixels are left for random selection although the MSF algorithm may not yet have reached its minimum or the average gray value of the example distribution texture. In this case, we arbitrarily assign O_{min} to D until pixel flipping can occur in the GLCM error minimization. This stipple region determination using geometrical information allows us to efficiently reduce the number of pixels considered during the synthesis process and maintains the geometrical information from the example texture.

3.5 Synthesis Results and Analysis

In order to demonstrate our stipple texture synthesis process, we have synthesized a series of textures from an artist's example textures. Figure 7 illustrates several examples textures (top row), the distributions extracted from the examples textures (middle row), and our synthesized distributions (bottom row) corresponding to the example distribution textures. To verify the accuracy of our synthesis, we have performed a GLCM statistical analysis on the textures, comparing the texture properties of the synthesized distribution textures to the example distribution textures. Figure 8 shows this analysis for the correlation texture statistics using a horizontal offset vector. Results are comparable for other offset vectors (e.g., vertical, diagonal). In Figure 8, our synthesized distribution textures (indicated by light red boxes in Figure 7) for tones 1, 4, 6 and 8 are compared to their corresponding example distribution textures (light blue boxes in Figure 7) in William M. Andrews' tone

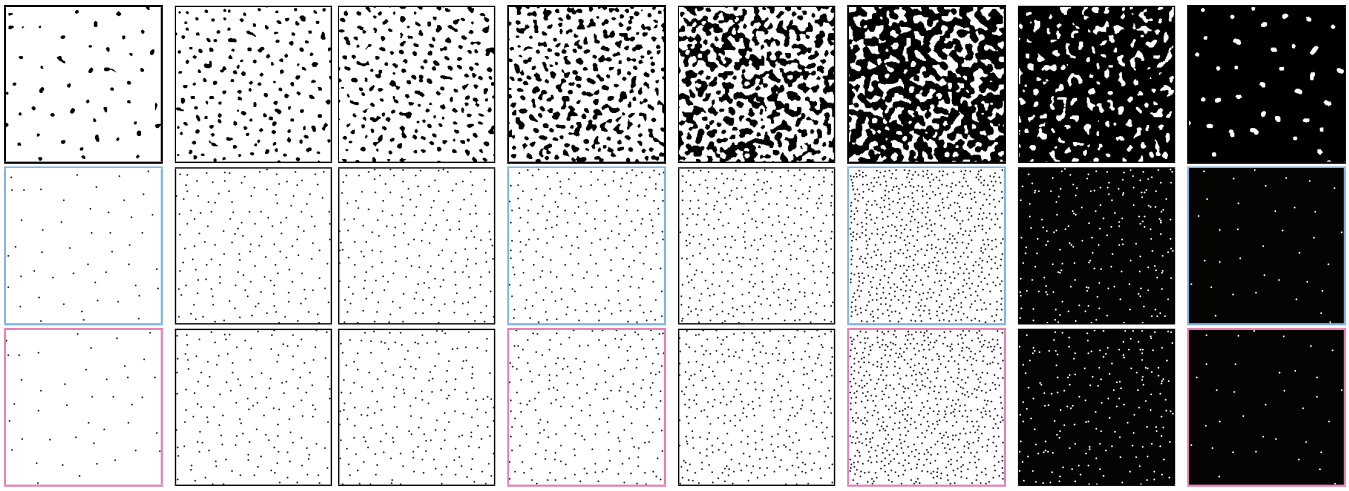

Figure 7: *From top to bottom, hand-drawn stipple tone textures by William M. Andrews, example distribution extracted by our method in Section 3.2, and our synthesized distribution textures by the algorithm in Section 3.4 corresponding to the example distribution textures. The colored boxes at tones 1, 4, 6 and 8 indicate the textures used for comparison in Figure 8. (For visualization purpose, pixels for the distribution are drawn larger than they actually are.)*

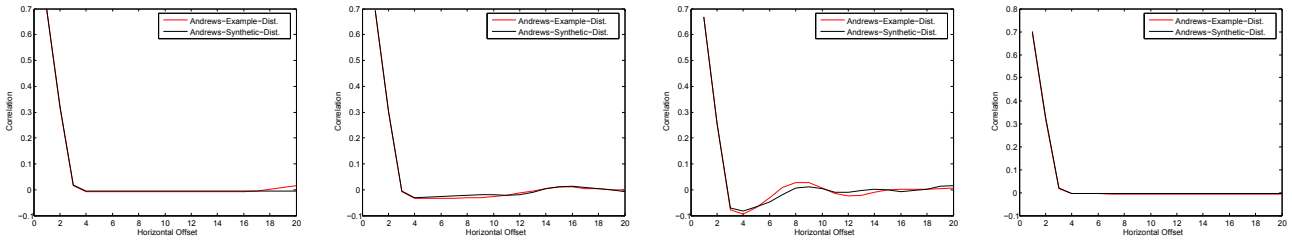

Figure 8: *From left to right, comparison of texture correlations as a function of offset between artist's stippling distribution and our synthesized distribution for tones 1, 4, 6, and 8 in Figure 7.*

maps. Here we see that the correlation property of the synthesized distribution textures is similar to that of the example distribution textures.

We also computed a correlation coefficient measure of the similarity between the correlation statistics of the synthesized and example distributions. This correlation coefficient measures the strength and the direction of a linear relationship between two variables so that its absolute value is 0 to 1, providing a simple metric for quick analysis. Generally, the absolute value of a coefficient greater than 0.8 is described as strong, whereas the absolute value of a coefficient less than 0.5 is described as weak. From our computation, our synthesized distribution textures (correlation coefficients = 0.9993, 0.9996, 0.9971, 1.0 for tones 1, 4, 6 and 8) are highly correlated with those of the corresponding example distribution textures. Note that the graphs represent the correlation of black and white pixels with respect to their distance from the origin where as the correlation coefficient denotes the similarity of the correlation statistics between the synthesized and example distributions.

The performance of our synthesis step depends on the texture size (n), the number of stipples, and the structural information. In our system, the average synthesis time is approximately 12 minutes for a 256×256 texture taken from a William M. Andrews' tone map (maximum iteration is set to 30, $O_{min} = 14.3$, $O_{max} = 40.8$, $O_{avg} = 29.7$, and $S_{avg} = 33.7$). Note that this synthesis only needs to be performed once for each artist's example stipple textures. Once completed, the synthesized textures are used to render images without repeating the synthesis.

4 Stipple Rendering

Having demonstrated that our stippling synthesis algorithm generates textures with texture statistics similar to the example stipple textures given as input, we now proceed to use these textures to create 2D stipple images in a manner similar to hand-drawn illustrations. Some artists use a continuous tone map as a reference, especially during the early development stages of a drawing when an illustrator is blocking out while determining the broad areas of the same or similar tone. For this purpose, we first create a *distribution image* using only the synthesized distribution textures in per-pixel rendering. Then, to render stipple marks, we use the stipple marks extracted from example stipple textures in Section 3.2.

4.1 Rendering with stipple distribution

To render 2D stippling images, we approximate a technique used by illustrators to match the continuous target tone by using discrete tone maps as done in many previous NPR works (e. g., [Winkenbach and Salesin 1994; Praun et al. 2001; Lee et al. 2006]). We utilize the previously synthesized distribution textures and apply them to grayscale images. However, the method in which we apply the distribution texture differs from the previous methods that use texture mapping or blending: our method deals solely with pixels describing the distribution of stipple primitives. Thus, all synthesized distribution textures are assigned their average gray value through pre-simulation in which stipple marks are placed on the synthesized

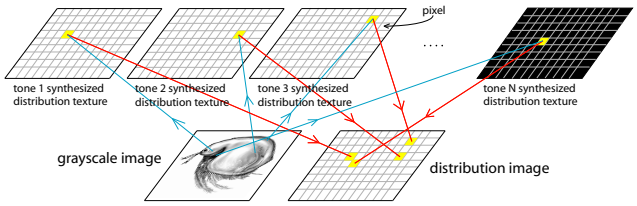

Figure 9: *Our rendering using stipple distribution textures.*

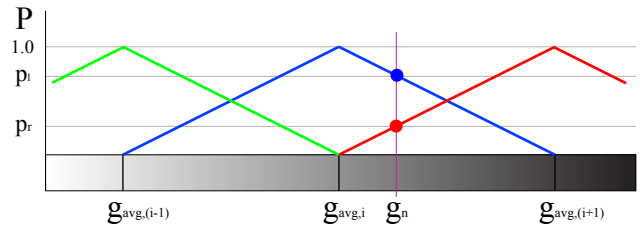

Figure 10: *Probability-based tone level selection used in our rendering step using distribution textures.*

distribution. We then fill every pixel in an input grayscale image with our synthesized distribution textures by using our tone level selection based on probability of the gray values within the input image.

Our rendering from the stipple distribution is illustrated in Figure 9, resulting in a distribution image as an output. In Figure 9, we determine a tone level for each pixel based on a gray value in the input image (light blue lines), and set a pixel value from our synthesized distribution texture corresponding to the tone level into the distribution image (red lines). In this manner, the distribution image contains the entire distribution information of stipple marks for a grayscale image.

Due to the limited number of tone levels, quantization artifacts may be introduced during the selection of a tone level for each pixel in a grayscale image. This is related to the difficulty of extracting appropriate example textures from an artist's work. To solve this problem, we employ a method based on probability for a gray value from an image. In Figure 10, g_n is a gray value of a pixel n within an input image, and $g_{avg,i}$ is the average gray value of the i^{th} tone level. We select either i^{th} or $(i+1)^{th}$ distribution texture with probability p_l or p_r respectively for the pixel n. This method preserves the statistical characteristics in our synthesized distribution textures as well as creates an image with continuous tones between discrete tone levels.

Our method is resolution dependent. Hence, to insure high quality results, the input grayscale images should have a similar resolution to that of the example stipple image from which we extracted the example stipple textures. Moreover, since we reuse stipple marks extracted from the example stipple textures, the resolution of an input grayscale image considerably affects the final image quality. Another issue in terms of resolution is that our synthesized distribution textures may be smaller than an input grayscale image. In such cases, we generate larger distribution textures by using multiple synthesized distribution textures. However, undesirable patterns may emerge in such a method. Our synthesis process minimizes these issues as each synthesized distribution texture will have slight variations. We further minimize these patterns by applying a random rotation to each small texture when merging them into their larger counterparts. Other possible techniques that could be used to minimize such patterns are image quilting by Efros and Freeman [2001] and Wang tiles by Cohen et al. [2003]. In this work, all results are rendered with 4096×4096 distribution textures stitched by randomly rotating 256×256 synthesized distribution textures.

4.2 Placement of Stipple Marks

For the placement of stipple marks into the distribution image from Section 4.1, we use the stipple marks extracted from the example stipple textures to better simulate actual hand-drawn stippling. Since our method uses two colors of stipple marks (black and white), we separate this stipple placement into two steps; first, we place black stipple marks onto the distribution image, and then we place white stipple marks. Therefore, we need to know which pixel in the distribution image is assigned from which tone level. Strictly speaking, we only need to know which pixel comes from the tone levels using either black stipple marks or white stipple marks. Thus, we use a *level map* that maintains a tone level for each stipple position to assign the appropriate stipple color. Finally, from a set of stipple marks, a random mark is uniformly selected and placed onto the distribution image, resulting in the final stipple image.

Our stipple marks placement, using the distribution image based on the probability of the gray values, creates unnecessary black and white pixels in the area rendered with middle level textures, although the overall appearance shows a smooth tone transition as shown in Figure 11 (left). In halftoning, there was also a similar problem, and there have been efforts such as Jodoin and Ostromoukhov [2001] and Zhou and Fang [2003] to solve this problem. We have tried to address this issue by identifying the small pixel chunks, both white on a black background and black on a white background, and moving them toward their true nearest stipple mark (see Figure 11 for a before and after image). While this removes the individual pixels successfully, it is not able to reproduce the blobby merging visible in hand-drawn stippling images. A more advanced technique for reproducing this pattern is necessary, which we leave for future work.

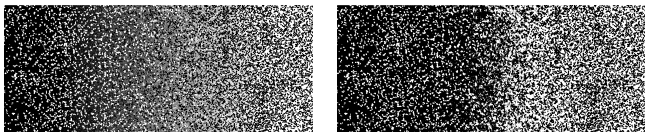

Figure 11: *Limitation: middle tonal ranges are not yet true to hand-drawn examples; before & after removing small pixel chunks.*

5 Results and Discussion

To demonstrate our method, we used hand-drawn stipple images by William M. Andrews, as sample inputs for our system. We generate a set of textures based on these examples and apply the synthesized textures to various grayscale images (see Figures 1, 12, and 13).Figure 12 compares our result to a similar hand-drawn stippling image. Figure 13 shows our result compared to other computer-generated stippling images by Secord [2002] and Schlechtweg et al. [2005]. The results illustrate the strength of our method in comparison to other computer generated methods in terms of appearing to have qualities similar to a hand-drawn image. Note that when comparing our result to the other computer generated results, we have less inherent structure, providing a less rigid feel within the images. However, our is not without limitations; Figure 12 (right) presents more chunky stipples than that of the original hand-drawn image, due to the random selection of stipple marks without considering the subtle size and orientation of stipple marks. Furthermore, the tone

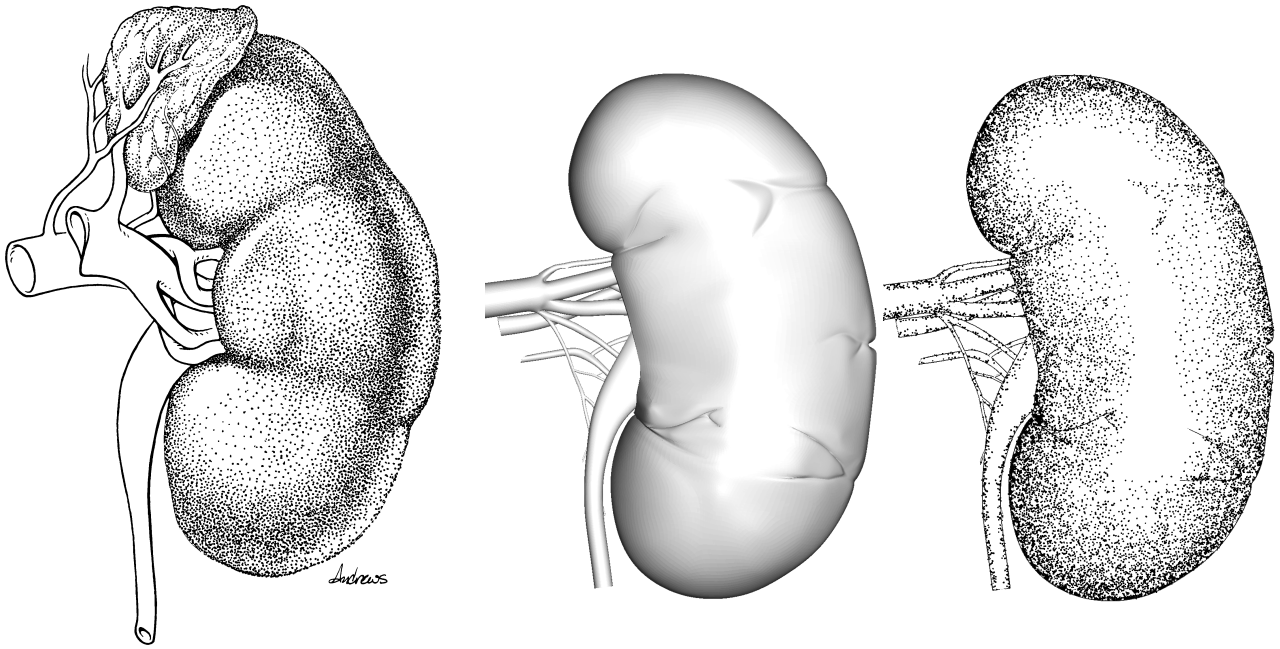

Figure 12: *Stipple result for a grayscale image (middle) of a polygonal rendering of a kidney model using textures synthesized from William M. Andrews tone maps (right). The left image is an original hand-drawn stippled medical illustration by William M. Andrews of a kidney, not based on the same model as our result (© William M. Andrews, used with permission).*

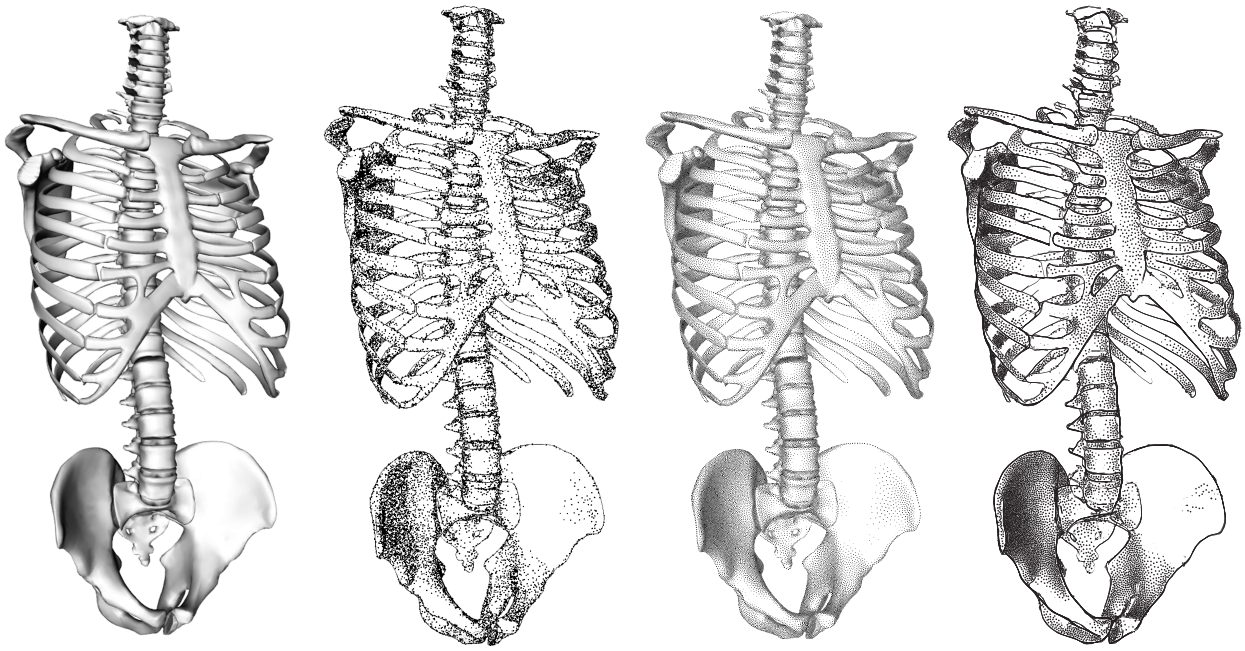

Figure 13: *Gallery of our stipple result using textures synthesized from William M. Andrews' tone maps. Left image shows grayscale input, middle-left is our result. Middle-right and right images are the computer-generated results by [Secord 2002] and [Schlechtweg 2005].*

transition issue mentioned in Section 4.2 needs to be improved for higher quality rendering.

6 Conclusions and Future Work

We have shown that our stippling by example technique is capable of capturing and reproducing stylistic variations of artists. Our synthesized distributions are statistically similar to that of the example textures as shown through the use of GLCMs, and our rendering method creates stipple images while maintaining the statistical characteristics, creating a perceptually similar appearance to the example stipple work. Our results move past "machine" rendering and its mathematical precision to "emulative" rendering and its subtle variations. Our technique can further be applied to computer-generated stipple renderings and other hand-drawn images to cap-

ture any stippling style, whether it is mechanical or expressive. As such, a system can be developed in which a user can extract stipple samples from their favorite artist and reproduce their style on any number of images. Furthermore, this technique bridges the gap found between images rendered using current NPR techniques and those rendered by hand.

Overall, our current results show that it is possible to capture and replicate properties from example stipple inputs. However, our method has several limitations. First, it is not always possible to extract example textures from hand-drawn stipple images as we mention in Section 3.1. Moreover, there are issues with representing the middle tonal ranges since we assume white stipple marks in a dark tone level as mentioned in Section 4.2. However, it is reasonable to do this algorithmically because stipple artists strive for the same distributions of the white spots in dark regions as they do for the black stipples on a white background. In addition, stippling is more than statistics. Hence, there are many issues such as structure awareness. However, in this work we focus on capturing and reproducing statistical characteristics represented in hand-drawn stipple images. Furthermore, in our method, support of resolution independence, such as in the work by Kopf et al. [2006], and the representation of continuous positions of stipple marks on a discrete pixel grid is left for future work.

Even with these limitations, our algorithm is able to better approximate hand-drawn stippling than other current comparable methods as shown in Figure 13. Furthermore, once the texture synthesis has been performed, rendering can be done on the fly. For example, Figure 1 (left) (4096×4096 resolution, stipple count = 33104 (black), 6064 (white)) takes less than 30 seconds (average 28.16 seconds) on Intel Xeon(R) CPU 2.66GHz processor and 3 GB of RAM. As such, our system is able to readily render any grayscale images in an artist's style once the tone map extraction and synthesis are completed.

Acknowledgements

We would like to thank Emily S. Damstra and Tobias Germer for permission to use their images in Figure 4 and 13 respectively. We also thank the anonymous reviewers for their careful and valuable comments and suggestions. This research has been funded by the U.S. National Science Foundation (NSF) and the U.S. Department of Homeland Security (DHS), and supported in part by Discovery Grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Natural Science Foundations of China (Nr.60873123).

References

BARLA, P., BRESLAV, S., MARKOSIAN, L., AND THOLLOT, J. 2006. Interactive hatching and stippling by example. Tech. rep., INRIA.

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F. X., AND MARKOSIAN, L. 2006. Stroke Pattern Analysis and Synthesis. *Computer Graphics Forum 25*, 3 (Sept.), 663–671.

CHETVERIKOV, D. 1994. GLDH Based Analysis of Texture Anisotropy and Symmetry: An Experimental Study. In *Proc. ICPR*, IEEE Computer Society, Los Alamitos, vol. 1, 444–448.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang Tiles for Image and Texture Generation. *ACM Transactions on Graphics 22*, 3 (July), 287–294.

COPELAND, A. C., AND TRIVEDI, M. M. 1998. Signature Strength Metrics For Camouflaged Targets Corresponding To Human Perceptual Cues. *Optical Engineering 37*, 2 (Feb.), 582–591.

COPELAND, A. C., RAVICHANDRAN, G., AND TRIVEDI, M. M. 2001. Texture Synthesis Using Gray-level Co-occurrence Models: Algorithms, Experimental Analysis, and Psychophysical Support. *Optical Engineering 40*, 11 (Nov.), 2655–2673.

DALAL, K., KLEIN, A. W., LIU, Y., AND SMITH, K. 2006. A Spectral Approach to NPR Packing. In *Proc. NPAR*, ACM, New York, 71–78.

DAVIS, L., JOHNS, S., AND AGGARWAL, J. 1979. Texture Analysis Using Generalized Co-occurrence Matrices. *IEEE Trans. PAMI 1*, 3, 251–259.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum 19*, 3 (Sept.), 40–51.

EFROS, A., AND FREEMAN, W. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proc. SIGGRAPH*, ACM Press, New York, 341–346.

EFROS, A., AND LEUNG, T. 1999. Texture Synthesis by Non-Parametric Sampling. In *Proc. ICCV*, IEEE, Los Alamitos, vol. 2, 1033–1038.

GOTLIEB, C. C., AND KREYSZIG, H. E. 1990. Texture Descriptors Based on Co-occurrence Matrices. *Computer Vision, Graphics and Image Processing 51*, 1 (July), 70–86.

HARALICK, R. M., SHANMUGAM, K., AND DINSTEIN, I. 1973. Textural Features for Image Classification. *IEEE Trans. Systems, Man, and Cybernetics 3*, 6 (Nov.), 610–621.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-Based Texture Analysis/Synthesis. In *Proc. SIGGRAPH*, ACM Press, New York, 229–238.

HILLER, S., HELLWIG, H., AND DEUSSEN, O. 2003. Beyond Stippling – Methods for Distributing Objects on the Plane. *Computer Graphics Forum 22*, 3 (Sept.), 515–522.

HODGES, E. R. S., Ed. 1989. *The Guild Handbook of Scientific Illustration*. Van Nostrand Reinhold, Hoboken, NJ.

IJIRI, T., MECH, R., IGARASHI, T., AND MILLER, G. 2008. An example-based procedural system for element arrangement. *Computer Graphics Forum 27*, 2, 429–436.

ISENBERG, T., NEUMANN, P., CARPENDALE, S., SOUSA, M. C., AND JORGE, J. A. 2006. Non-Photorealistic Rendering in Context: An Observational Study. In *Proc. NPAR*, ACM Press, New York, 115–126.

JODOIN, P.-M., AND OSTROMOUKHOV, V. 2001. Error Diffusion With Blue-Noise Properties for Midtones. In *Proceedings of SPIE Color Imaging: Device-Independent Color, Color Hardcopy, and Applications VII*, SPIE, Bellingham, Washington, R. Eschbach and G. G. Marcu, Eds., vol. 4663 of *SPIE Proceedings Series*, 293–301.

JODOIN, P.-M., EPSTEIN, E., GRANGER-PICHÉ, M., AND OSTROMOUKHOV, V. 2002. Hatching by example: a statistical approach. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 29–36.

JULESZ, B., GILBERT, E. N., SHEPP, L. A., AND FRISCH, H. L. 1973. Inability of Humans to Discriminate Between Visual Textures that Agree in Second-Order Statistics – Revisited. *Perception 2*, 4, 391–405.

KIM, D., SON, M., LEE, Y., KANG, H., AND LEE, S. 2008. Feature-Guided Image Stippling. *Computer Graphics Forum 27*, 4, 1209–1216.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Transactions on Graphics 25*, 3 (July), 509–518.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics 22*, 3 (July), 277–286.

LEE, H., KWON, S., AND LEE, S. 2006. Real-Time Pencil Rendering. In *Proc. NPAR*, ACM Press, New York, 37–45.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-Time Texture Synthesis by Patch-Based Sampling. *ACM Transactions on Graphics 20*, 3 (July), 127–150.

LOHMANN, G. 1994. Co-occurrence-based Analysis and Synthesis of Textures. In *Proc. Pattern Recognition*, IEEE Computer Society, Los Alamitos, vol. 1, 449–453.

LU, A., MORRIS, C. J., TAYLOR, J., EBERT, D. S., HANSEN, C. D., RHEINGANS, P., AND HARTNER, M. 2003. Illustrative Interactive Stipple Rendering. *IEEE TVCG 9*, 2 (Apr.–June), 127–138.

MACIEJEWSKI, R., ISENBERG, T., ANDREWS, W. M., EBERT, D. S., SOUSA, M. C., AND CHEN, W. 2008. Measuring Stipple Aesthetics in Hand-Drawn and Computer-Generated Images. *IEEE Computer Graphics & Applications 28*, 2 (Mar./Apr.), 62–74.

METROPOLIS, N., ROSENBLUTH, A., ROSENBLUTH, M., TELLER, A., AND TELLER, E. 1953. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics 21*, 6 (June), 1087–1092.

MOULD, D. 2007. Stipple Placement using Distance in a Weighted Graph. In *Proc. CAe*, Eurographics Assoc., Aire-la-Ville, Switzerland, 45–52.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision 40*, 1 (Oct.), 49–70.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-Time Hatching. In *Proc. SIGGRAPH*, ACM Press, New York, 581–590.

QIN, X., AND YANG, Y.-H. 2005. Basic Gray Level Aura Matrices: Theory and its Application to Texture Synthesis. In *Proc. ICCV*, IEEE Computer Society, Los Alamitos, vol. 1, 128–135.

SCHLECHTWEG, S., GERMER, T., AND STROTHOTTE, T. 2005. RenderBots—Multi Agent Systems for Direct Image Generation. *Computer Graphics Forum 24*, 2 (June), 137–148.

SECORD, A. 2002. Weighted Voronoi Stippling. In *Proc. NPAR*, ACM Press, New York, 37–43.

VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. 2007. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, 139–146.

WEI, L.-Y., AND LEVOY, M. 2000. Fast Texture Synthesis using Tree-Structured Vector Quantization. In *Proc. SIGGRAPH*, ACM Press, New York, 479–488.

WINKENBACH, G., AND SALESIN, D. 1994. Computer-Generated Pen-and-Ink Illustration. In *Proc. SIGGRAPH*, ACM, New York, 91–100.

ZHOU, B., AND FANG, X. 2003. Improving mid-tone quality of variable-coefficient error diffusion using threshold modulation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 437–444.

ZHU, S. C., WU, Y. N., AND MUMFORD, D. 1997. Minimax Entropy Principle and its Application to Texture Modeling. *Neural Computation 9*, 8 (Nov.), 1627–1660.