

# Point Sampling with General Noise Spectrum

Yahan Zhou\*

Haibin Huang\*

Li-Yi Wei†

Rui Wang\*

\*University of Massachusetts Amherst

†The University of Hong Kong

## Abstract

Point samples with different spectral noise properties (often defined using color names such as white, blue, green, and red) are important for many science and engineering disciplines including computer graphics. While existing techniques can easily produce white and blue noise samples, relatively little is known for generating other noise patterns. In particular, no single algorithm is available to generate different noise patterns according to user-defined spectra.

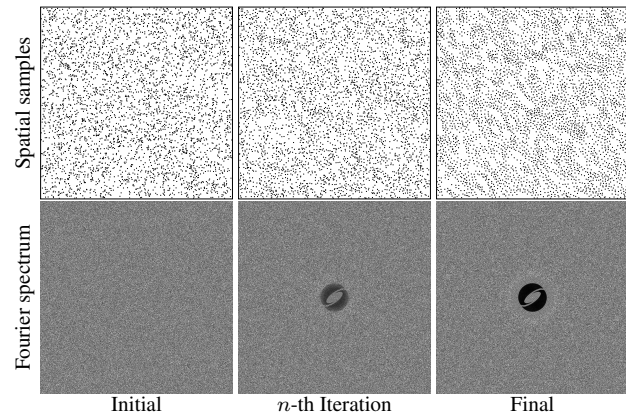
In this paper, we describe an algorithm for generating point samples that match a user-defined Fourier spectrum function. Such a spectrum function can be either obtained from a known sampling method, or completely constructed by the user. Our key idea is to convert the Fourier spectrum function into a *differential distribution function* that describes the samples' local spatial statistics; we then use a gradient descent solver to iteratively compute a sample set that matches the target differential distribution function. Our algorithm can be easily modified to achieve adaptive sampling, and we provide a GPU-based implementation. Finally, we present a variety of different sample patterns obtained using our algorithm, and demonstrate suitable applications.

**Keywords:** point sampling, noise spectrum, adaptive sampling

**Links:** [DL](#) [PDF](#)

## 1 Introduction

Point samples with different spectral noise properties have a wide range of applications across many disciplines. Such samples are often described by color names based on their spectral shapes. For example, *blue noise* is frequently used for anti-aliasing, texture synthesis, stochastic ray tracing, stippling, and remeshing [Lloyd 1983; Dippé and Wold 1985; Cook 1986; Mitchell 1987; Turk 1992; Alliez et al. 2002; Dutre et al. 2002; Pharr and Humphreys 2004; Ostromoukhov et al. 2004; Kopf et al. 2006; Ostromoukhov 2007; Balzer et al. 2009; Wei 2010; Öztireli et al. 2010; Fattal 2011; Schlömer et al. 2011]; *green noise* has been used for halftoning [Lau et al. 2003]; *white noise* is the common standard for random number generators [Tzeng and Wei 2008]; and *pink or red noise* widely exists in nature, making it suitable for simulating physical and biological distributions [Condit et al. 2000; Ostling et al. 2000]. Additionally, in image processing, it is often important to design samples with specific spectral properties in order to filter certain frequency content in the image signal.



**Figure 1:** Given the sample count and a target Fourier spectrum (a SIGGRAPH logo in this example), our algorithm produces a set of point samples (upper right) that matches the target spectrum. The computation starts from random initial samples, and iteratively updates points using a gradient descent solver. More examples can be found in Figure 2.

In these applications, we need to generate point samples that have desired spectral noise properties. Most existing work has focused on blue noise sampling, but cannot be used to produce samples with other spectral colors such as green, red, or more complex hybrid spectral colors. To our knowledge, no existing method provides such capability, including handling a wide range of different noise spectra, or even user-specified spectrum functions (see Figure 2).

In this paper, we describe an algorithm for computing point samples that conform with general, user-specified Fourier spectrum functions. Such a function can be either obtained from a known sampling method, or constructed by the user. Our goal is to produce a set of sample points whose Fourier spectrum matches the given input. Our key idea is to first convert the Fourier spectrum function into a differential distribution function [Wei and Wang 2011], which describes the samples' local spatial statistics and is equivalent to the Fourier spectrum via a cosine transform. We then use a gradient descent solver to iteratively compute a sample set that matches the target differential distribution function. Figure 1 shows an example. The main advantages of using this function instead of the Fourier spectrum function are: 1) it can easily handle spatial density changes and achieve adaptive sampling; 2) it does not require computing the Fourier transform, thus is computationally more efficient; 3) it is easy to implement on the GPU to achieve parallel computation.

We have studied this algorithm for both scientific curiosity and practical applications. The paper is organized as follows: in Section 3 we describe the problem formulation; in Section 4 we derive the algorithm and explain implementation details; then in Section 5 we present results, outline the characteristics of novel sample patterns, and demonstrate suitable applications such as green noise for image stippling and red/pink noise for distributing scene elements.

## 2 Related Work

**Color of noise.** The spectral distribution property of noise patterns is often described in terms of the Fourier spectrum color. For example, white noise has a flat spectrum, with equal energy distributed in all frequency bands. Blue noise has weak low-frequency energy, but strong high-frequency (i.e. blue) energy. In computer graphics, the use of blue noise is universal in a variety of applications. This is mainly because it can reduce aliasing by replacing low-frequency aliases with high-frequency noise that is more acceptable to human eyes [Yellott 1983; Dippé and Wold 1985; Cook 1986]. It also produces perceptually pleasing patterns that are crucial for texture synthesis, image stippling, and remeshing.

Red/pink noise is the complement of blue noise in that its spectral energy is concentrated in the low-frequency bands. It can be used to characterize the distributions of plants [Condit et al. 2000; Ostling et al. 2000] or galaxies [Martinez et al. 1995] which have been observed to form concentrated clusters.

Green noise, as its name suggests, consists of primarily mid-range frequencies. It characterizes the distributions of a variety of natural phenomena such as plumed seeds [Greene and Johnson 1989] and fallen leaves [Schua et al. 2009]. Therefore it can be used to simulate the distribution of these scene elements. In addition, green noise has been applied in [Lau et al. 1999; Lau et al. 2003] for half-toning, producing clustered-dot printing in a discrete setting. Compared to blue noise, it can benefit printing devices that favor clustered-dot rather than isolated-dot patterns.

During the study of green noise, Lau et al. [1999] introduced the Binary Pattern Pair Correlation Construction Algorithm, for generating dithering patterns from an arbitrarily shaped pair correlation function. While related to ours, their method does not guarantee that the resulting samples will closely match the input pair correlation function. In addition, every sample is sequentially generated and used to modify a global sampling PDF, thus the method is non-trivial to parallelize and achieve high performance.

**Blue noise sampling.** A variety of methods are available for generating blue noise samples. These methods can be broadly classified into four categories: 1) Poisson disk sampling, dart throwing, and variants [Dippé and Wold 1985; Cook 1986; Mitchell 1987; Öztireli et al. 2010; Kalantari and Sen 2011; Ebeida et al. 2011; Ebeida et al. 2012]; 2) optimization and relaxation-based methods [Lloyd 1983; Turk 1992; Balzer et al. 2009; Fattal 2011]; 3) tiling-based methods [Ostromoukhov et al. 2004; Kopf et al. 2006; Lagae and Dutré 2006]; and 4) half-toning including error diffusion [Ulichney 1987; Ostromoukhov 2001; Zhou and Fang 2003; Pang et al. 2008; Chang et al. 2009]. Our method is optimization-based, but in contrast to previous work, our goal is to compute a sample set that matches any given spectrum function. Therefore, our algorithm can be used to simulate existing blue noise techniques (see Figures 2(e), 4, 5), as well as produce novel noise patterns.

Parker et al. [1991] proposed an algorithm for manipulating the power spectra of blue noise halftone patterns. A blue noise spectrum defined by a step function is used as input to influence the sample distributions. It is unclear how this method can be modified to work with other noise spectra, or if the output can faithfully match the input spectrum.

**Sample analysis.** A variety of standard methods are available to measure the quality and distribution property of samples. In terms of spatial analysis [Dale et al. 2002], one can calculate the samples' discrepancy [Shirley 1991] and relative radius [Lagae and Dutré 2008]. These provide scalar measures to indicate the global

uniformity and density of samples. In terms of spectrum analysis, it is common to compute the samples' Fourier power spectrum [Ulichney 1987; Lagae and Dutré 2008], whose radial means and anisotropy graphs are then used to quantify and detect artifacts in uniformly distributed samples.

Recently, Wei and Wang [2011] proposed a new method for analyzing non-uniformly distributed samples. Their key idea is to reformulate the Fourier spectrum into a new form called the *differential distribution function*. This function characterizes the samples' local spatial statistics, and can infer their spectral properties without requiring a Fourier transform. Due to its local nature, the differential distribution function can account for spatial distance metric changes, hence enabling non-uniform sample analysis. Our method builds upon their technique. Specifically, our goal is to compute a sample set that matches a target differential distribution function.

## 3 Overview

**Fourier spectrum analysis.** Given a set of  $N$  samples  $\{\mathbf{s}_k\}$  ( $k \in [0, N-1]$ ) in an  $n$  dimensional space, the Fourier transform of the sample set is defined as

$$F(\mathbf{f}) = \sum_{k=0}^{N-1} e^{-2\pi i(\mathbf{f} \cdot \mathbf{s}_k)}, \quad (1)$$

where  $\mathbf{f}$  is the frequency vector (in 2D,  $\mathbf{f} = [f_x, f_y]$  represents the scalar frequencies in the  $x$  and  $y$  directions respectively);  $\cdot$  denotes the vector inner product. For the purpose of analyzing sample patterns, we are mainly interested in the power spectrum  $P(\mathbf{f})$  [Ulichney 1987; Bracewell 1999; Lagae and Dutré 2008], which measures the squared magnitude of  $F(\mathbf{f})$  and is defined by

$$P(\mathbf{f}) = |F(\mathbf{f})|^2 = \frac{1}{N} \left( \sum_{k=0}^{N-1} \cos(2\pi \mathbf{f} \cdot \mathbf{s}_k) \right)^2 + \frac{1}{N} \left( \sum_{k=0}^{N-1} \sin(2\pi \mathbf{f} \cdot \mathbf{s}_k) \right)^2 \quad (2)$$

The average and relative variance of  $P(\mathbf{f})$  in each frequency band (consisting of frequency vectors at the same magnitude but different orientations) provide the radial means and anisotropy measures. These measures are typically plotted into 1D graphs, and are used to analyze the samples' spectral distribution properties, including spatial artifacts and angular biases.

**Differential domain analysis.** Wei and Wang [2011] observed that the Fourier spectrum in Eq. 2 can be re-written into another form that only depends on the sample location differences. This can be derived by expanding the quadratic terms in Eq. 2, applying basic trigonometry rules, and observing that the expectation of a function (i.e.  $\cos$ ) of a random variable (i.e.  $(\mathbf{s}_i - \mathbf{s}_j)$ ) is equal to the integral of the function with the probability distribution of the random variable. Specifically, with sufficiently large  $N$ , we have

$$P(\mathbf{f}) = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \cos[2\pi \mathbf{f} \cdot (\mathbf{s}_i - \mathbf{s}_j)] \approx \int_{\Omega_d} \cos(2\pi \mathbf{f} \cdot \mathbf{d}) p(\mathbf{d}) \delta \mathbf{d}. \quad (3)$$

Here  $p(\mathbf{d})$  is called the **differential distribution function**. It describes the probability distribution of the difference  $(\mathbf{s}_i - \mathbf{s}_j)$  be-

tween every pair of samples  $i$  and  $j$ . It can be evaluated as

$$\begin{aligned} p(\mathbf{d}) &= \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \kappa(\mathbf{d}, (\mathbf{s}_i - \mathbf{s}_j)) \\ &= \kappa(\mathbf{d}, \mathbf{0}) + \frac{1}{N} \sum_{i=0}^{N-1} \sum_{\substack{j=0 \\ j \neq i}}^{N-1} \kappa(\mathbf{d}, (\mathbf{s}_i - \mathbf{s}_j)), \end{aligned} \quad (4)$$

where  $\kappa$  is a density estimation kernel. It can be a Dirac delta function, but for robust estimation it is generally a Gaussian kernel,

$$\kappa(\mathbf{d}, \mathbf{d}_0) = e^{-\frac{\|\mathbf{d} - \mathbf{d}_0\|^2}{\sigma^2}}, \quad (5)$$

where  $\mathbf{d}_0$  is the center and  $\sigma$  is the standard deviation. Note that  $p(\mathbf{d})$  sums over all pairs of samples and  $\kappa$  is rotationally symmetric, thus  $p(\mathbf{d})$  is an even function:  $p(-\mathbf{d}) = p(\mathbf{d})$ .

Eq. 3 tells us that  $p(\mathbf{d})$  relates to the Fourier spectrum  $P(\mathbf{f})$  via a cosine transform. Note that this can be viewed as the Fourier transform of an even function  $p(\mathbf{d})$ . The same relationship can also be derived by computing the discrete autocorrelation and applying the Wiener-Khinchin theorem, or alternatively by directly using the Wigner transform [Schroeder 1999]. The main difference is that the autocorrelation and the Wigner transform are typically defined in 1D for time domain signals, while we are interested in 2D spatial domain samples.

As shown in [Wei and Wang 2011], it suffices to examine  $p(\mathbf{d})$  over a short range of  $\mathbf{d}$  values. In other words, the differential distribution function  $p(\mathbf{d})$  (and consequently the Fourier spectrum  $P(\mathbf{f})$ ) is largely determined by each sample's local neighbors. Samples beyond a certain distance from each other contribute little to  $p(\mathbf{d})$  or the Fourier spectrum. This property makes it possible to use a locally defined distance metric to account for spatial or angular changes in the sampling density. As a result,  $p(\mathbf{d})$  can be used to analyze samples that are non-uniformly distributed, including adaptive and anisotropic samples, as well as samples computed with geodesic distance metric defined on a mesh surface.

**Problem Formulation.** Our goal in this paper is to design an algorithm that can compute samples according to a user-specified Fourier spectrum. This is motivated by the fact that while many techniques are available to generate specific sample patterns, such as Poisson disk samples or CCVT [Balzer et al. 2009], no single algorithm can produce a broad range of different sample patterns. Mathematically, given a target spectrum  $\Phi(\mathbf{f})$  and a desired number of samples  $N$ , we formulate the problem as finding the positions of samples  $\{\mathbf{s}_k\}$  such that their Fourier spectrum  $P(\mathbf{f})$  matches  $\Phi(\mathbf{f})$  as closely as possible, by minimizing the following error function:

$$E_P = \int [P(\mathbf{f}) - \Phi(\mathbf{f})]^2 \delta \mathbf{f}.$$

However, defining the error function this way makes it difficult to solve the problem, or use the same method for adaptive sampling or other non-uniform domains, since the Fourier spectrum is not well defined in such domains [Wei and Wang 2011]. Instead, since  $P(\mathbf{f})$  and  $p(\mathbf{d})$  are related by a Fourier transform, which preserves the L2 norm, we can re-define the error function as

$$E_p = \int [p(\mathbf{d}) - \phi(\mathbf{d})]^2 \delta \mathbf{d}, \quad (6)$$

where  $\phi(\mathbf{d})$  is the target differential distribution function corresponding to  $\Phi(\mathbf{f})$ . It is defined as the *inverse* Fourier (cosine) transform of  $\Phi(\mathbf{f})$ . The main advantage of this formulation is that since

$p(\mathbf{d})$  is a local quantity, it allows us to use the same method for non-uniform sampling. It also avoids computing the Fourier transform, thus is computationally more efficient. In addition,  $p(\mathbf{d})$  is easy to parallelize and straightforward to implement on the GPU.

## 4 Algorithms and Implementation

In this section, we discuss the algorithms and implementation details for solving Eq. 6. Our main inputs are a target spectrum function and the desired number of samples  $N$ . The target function can be given in the form of a differential distribution  $\phi(\mathbf{d})$  directly, or a Fourier spectrum  $\Phi(\mathbf{f})$  (in which case it will be converted to the corresponding  $\phi(\mathbf{d})$  as described in Section 4.2 below). We start with uniform sampling, then extend the method to adaptive sampling.

### 4.1 Gradient Descent Solver

We use a gradient descent method to iteratively minimize Eq. 6. We start with a set of uniform random (i.e. white noise) points. At each iteration we move every point  $\mathbf{s}_k$  locally towards the direction that reduces  $E_p$  the fastest. Such a direction is given by the negative gradient of  $E_p$  with respect to  $\mathbf{s}_k$ . After a sufficient number of iterations, the sample points will converge and we output the final result. Details are described below.

To begin, the gradient of  $E_p$  with respect to  $\mathbf{s}_k$  is calculated by

$$\frac{\partial E_p}{\partial \mathbf{s}_k} = \frac{\partial \int [p(\mathbf{d}) - \phi(\mathbf{d})]^2 \delta \mathbf{d}}{\partial \mathbf{s}_k} = 2 \int [p(\mathbf{d}) - \phi(\mathbf{d})] \frac{\partial p(\mathbf{d})}{\partial \mathbf{s}_k} \delta \mathbf{d}.$$

Recall that  $p(\mathbf{d})$  denotes the samples' current differential distribution function, and  $\phi(\mathbf{d})$  denotes the target distribution function. Intuitively  $\frac{\partial p(\mathbf{d})}{\partial \mathbf{s}_k}$  measures the change in  $p(\mathbf{d})$  due to a small change in sample  $\mathbf{s}_k$ . By the definition of  $p(\mathbf{d})$  in Eq. 4, we have

$$\begin{aligned} \frac{\partial p(\mathbf{d})}{\partial \mathbf{s}_k} &= \frac{\partial \left( \sum_{j \neq k}^{N-1} \kappa(\mathbf{d}, (\mathbf{s}_k - \mathbf{s}_j)) + \sum_{i \neq k}^{N-1} \kappa(\mathbf{d}, (\mathbf{s}_i - \mathbf{s}_k)) \right)}{N \partial \mathbf{s}_k} \\ &= \frac{1}{N} \sum_{j \neq k}^{N-1} \kappa'(\mathbf{d}, (\mathbf{s}_k - \mathbf{s}_j)) - \frac{1}{N} \sum_{i \neq k}^{N-1} \kappa'(\mathbf{d}, (\mathbf{s}_i - \mathbf{s}_k)). \end{aligned}$$

Combining the above two equations, we have

$$\begin{aligned} \frac{\partial E_p}{\partial \mathbf{s}_k} &= \frac{2}{N} \sum_{j \neq k}^{N-1} \int [p(\mathbf{d}) - \phi(\mathbf{d})] \kappa'(\mathbf{d}, (\mathbf{s}_k - \mathbf{s}_j)) \delta \mathbf{d} \\ &\quad - \frac{2}{N} \sum_{i \neq k}^{N-1} \int [p(\mathbf{d}) - \phi(\mathbf{d})] \kappa'(\mathbf{d}, (\mathbf{s}_i - \mathbf{s}_k)) \delta \mathbf{d}. \end{aligned} \quad (7)$$

Here  $\kappa'(\mathbf{d}, \mathbf{d}_0) = \frac{\partial \kappa(\mathbf{d}, \mathbf{d}_0)}{\partial \mathbf{d}_0}$  represents the first-order derivative of the Gaussian kernel with respect to  $\mathbf{d}_0$ . It's defined as

$$\kappa'(\mathbf{d}, \mathbf{d}_0) = -\frac{2}{\sigma^2} (\mathbf{d}_0 - \mathbf{d}) e^{-\frac{\|\mathbf{d}_0 - \mathbf{d}\|^2}{\sigma^2}}. \quad (8)$$

Note that it is a vector quantity independent of the samples.

Since the integrals in Eq. 7 are over the entire domain of  $\mathbf{d}$ , and  $p(\mathbf{d})$ ,  $\phi(\mathbf{d})$  are both even functions while  $\kappa'$  is an odd function (i.e.  $\kappa'(\mathbf{d}, \mathbf{d}_0) = -\kappa'(-\mathbf{d}, -\mathbf{d}_0)$ ), it is easy to prove that the upper and lower terms of Eq. 7 are actually equal. Therefore, we have

$$\frac{\partial E_p}{\partial \mathbf{s}_k} = \frac{4}{N} \sum_{i \neq k}^{N-1} \int [p(\mathbf{d}) - \phi(\mathbf{d})] \kappa'(\mathbf{d}, (\mathbf{s}_k - \mathbf{s}_i)) \delta \mathbf{d}.$$

Note that each term in the sum is an integral of  $[p(\mathbf{d}) - \phi(\mathbf{d})]$  with the function  $\kappa'(\mathbf{d})$  centered at  $(\mathbf{s}_i - \mathbf{s}_k)$ . Therefore we can compute it more efficiently by first convolving  $[p(\mathbf{d}) - \phi(\mathbf{d})]$  with  $\kappa'(\mathbf{d})$ , then index the result using  $(\mathbf{s}_i - \mathbf{s}_k)$ . In other words,

$$\frac{\partial E_p}{\partial \mathbf{s}_k} = \frac{4}{N} \sum_{i \neq k}^{N-1} [(p(\mathbf{d}) - \phi(\mathbf{d})) * \kappa'(\mathbf{d})](\mathbf{s}_k - \mathbf{s}_i), \quad (9)$$

where  $*$  denotes a convolution. Finally, at each iteration  $t$ , we update every sample  $\mathbf{s}_k$  by moving it in the opposite direction of  $\frac{\partial E_p}{\partial \mathbf{s}_k}$  by a small step size  $\Delta t$ :

$$\mathbf{s}_k^{t+1} = \mathbf{s}_k^t - \Delta t \frac{\partial E_p}{\partial \mathbf{s}_k}. \quad (10)$$

In sum, the gradient descent solver involves the following steps: first, calculate the current differential distribution function  $p(\mathbf{d})$ ; then compute  $[p(\mathbf{d}) - \phi(\mathbf{d})]$  and convolve it with  $\kappa'(\mathbf{d})$  defined in Eq. 8; next, for each sample  $\mathbf{s}_k$ , loop over all nearby samples  $\mathbf{s}_i$  and compute  $\frac{\partial E_p}{\partial \mathbf{s}_k}$  (Eq. 9); and finally, update  $\mathbf{s}_k$  according to Eq. 10.

**Parameter Selection.** There are two main parameters we need to consider here. The first is the Gaussian kernel’s standard deviation  $\sigma$  (Eq. 5), which indicates the kernel’s support size relative to a pixel size in the 2D histogram used to represent  $\phi(\mathbf{d})$ . Setting it too small will cause the evaluation of  $\frac{\partial E_p}{\partial \mathbf{s}_k}$  to be very noisy, and setting it too large will reduce the accuracy of matching  $p(\mathbf{d})$  to  $\phi(\mathbf{d})$ . By default we use  $128 \times 128$  histogram resolution and set  $\sigma = 2$ .

The second parameter is the step size  $\Delta t$  in Eq. 10. Setting  $\Delta t$  too large will cause the computation to be unstable, and setting it too small will increase the computation time. Generally, we would like to bound  $\Delta t \|\frac{\partial E_p}{\partial \mathbf{s}_k}\|$ , so that after one iteration the movement of any point will be bounded relative to the mean distance between samples. Our approach is to first calculate  $\|\frac{\partial E_p}{\partial \mathbf{s}_k}\|$  for every sample, then find the maximum value  $\|\frac{\partial E_p}{\partial \mathbf{s}_k}\|_{\max}$ , and finally set

$$\Delta t = \frac{c_d}{\sqrt[n]{N}} \frac{1}{\|\frac{\partial E_p}{\partial \mathbf{s}_k}\|_{\max}}, \quad (11)$$

where  $c_d = 0.1$  is a constant scaling factor. The reason  $\sqrt[n]{N}$  appears in the denominator is that  $\Delta t$  should be calculated relative to the mean distance between samples, which itself is inversely proportional to  $\sqrt[n]{N}$  in an  $n$ -dimensional space. In practice, we also set an upper limit on  $\Delta t$  to avoid it being too large, which may happen when  $\|\frac{\partial E_p}{\partial \mathbf{s}_k}\|_{\max}$  is close to zero as the computation converges.

**Convergence.** To decide when to terminate the computation, we check the value of the error function defined in Eq. 6. The error value generally decreases if the computation has not converged yet. For robust checking, we calculate the average error over each pass of 200 iterations. If the average error value is at least 2% better than the previous pass, we continue; otherwise we treat the results as converged and terminate the computation.

For blue noise samples such as Poisson disk and CCVT, their  $\phi(\mathbf{d})$  functions contain regions of zeros. This generally takes more iterations to converge because we need to ensure that the resulting  $p(\mathbf{d})$  exactly matches  $\phi(\mathbf{d})$  in the regions of zeros. While we could add a penalty term to Eq. 6 to enforce the constraint, we use a simpler method that increases the 200 iterations per pass to 1000. This has worked well in practice.

**Discussion.** Our algorithm relates to the kernel density model by Fattal [2011]. From the algorithm point of view, both update points by moving them primarily towards the negative gradient direction of an error function. Fattal’s method is formulated as MCMC sampling of a sample distribution that is defined using a spatial error function and a temperature parameter. It is inspired by statistical mechanics and is designed to compute blue-noise samples only. In contrast, our method directly minimizes an error function defined in the differential domain. Our goal is to match a target differential distribution function, so that we can compute samples with an arbitrary spectrum profile. As discussed in Section 4.3, we solve adaptive sampling by exploiting the invariance of the differential distribution function.

## 4.2 Computing and Normalizing $\phi(\mathbf{d})$

The target  $\phi(\mathbf{d})$  may be provided by the user in several ways. The first is to use  $\phi(\mathbf{d})$  obtained from a known sampling algorithm (see Figures 2(e), 4, 5 for examples). This allows us to simulate existing algorithms. The second way is for the user to plot  $\phi(\mathbf{d})$  directly (such as Figures 2(g), 9). Since  $\phi(\mathbf{d})$  is represented as a two or higher dimensional histogram, it’s more convenient for the user to plot the radial means of  $\phi(\mathbf{d})$  as a 1D curve, then produce a rotationally symmetric  $\phi(\mathbf{d})$  from the 1D curve. Finally, the user can also provide the target Fourier spectrum  $\Phi(\mathbf{f})$  instead of  $\phi(\mathbf{d})$ , and our algorithm will automatically convert  $\Phi(\mathbf{f})$  to  $\phi(\mathbf{d})$ . Most examples in the paper are generated this way. We find that this is usually more intuitive, since the Fourier spectrum graph corresponds more closely to our notion of the ‘color’ of noise.

**Properties of  $\phi(\mathbf{d})$ .** We assume that samples beyond a certain distance from each other are uncorrelated, which is true for the typical sample sets we are interested in. Thus  $\phi(\mathbf{d})$  is usually flat (like in white noise) for sufficiently large  $\mathbf{d}$ . To calculate the distribution, we only need to account for each sample’s local neighbors, defined by a neighborhood size  $\Delta_p$ . Therefore when providing  $\phi(\mathbf{d})$  we always include a flat portion at the end to make sure that the interesting features of it are sufficiently covered within  $\Delta_p$ .

The value  $\Delta_p$  is set by the user. It indicates the physical size of  $\phi(\mathbf{d})$  in the sample domain. For example, for Poisson disk samples  $\Delta_p$  is around 6 ~ 7 times the Poisson minimum distance (see [Wei and Wang 2011]). Setting  $\Delta_p$  too large increases the computation cost and also loses locality (which makes it difficult to perform adaptive sampling as described later). Setting it too small reduces the accuracy of the calculated  $p(\mathbf{d})$ . The typical value of  $\Delta_p$  is between 0.05 ~ 0.2 with respect to a unit square domain.

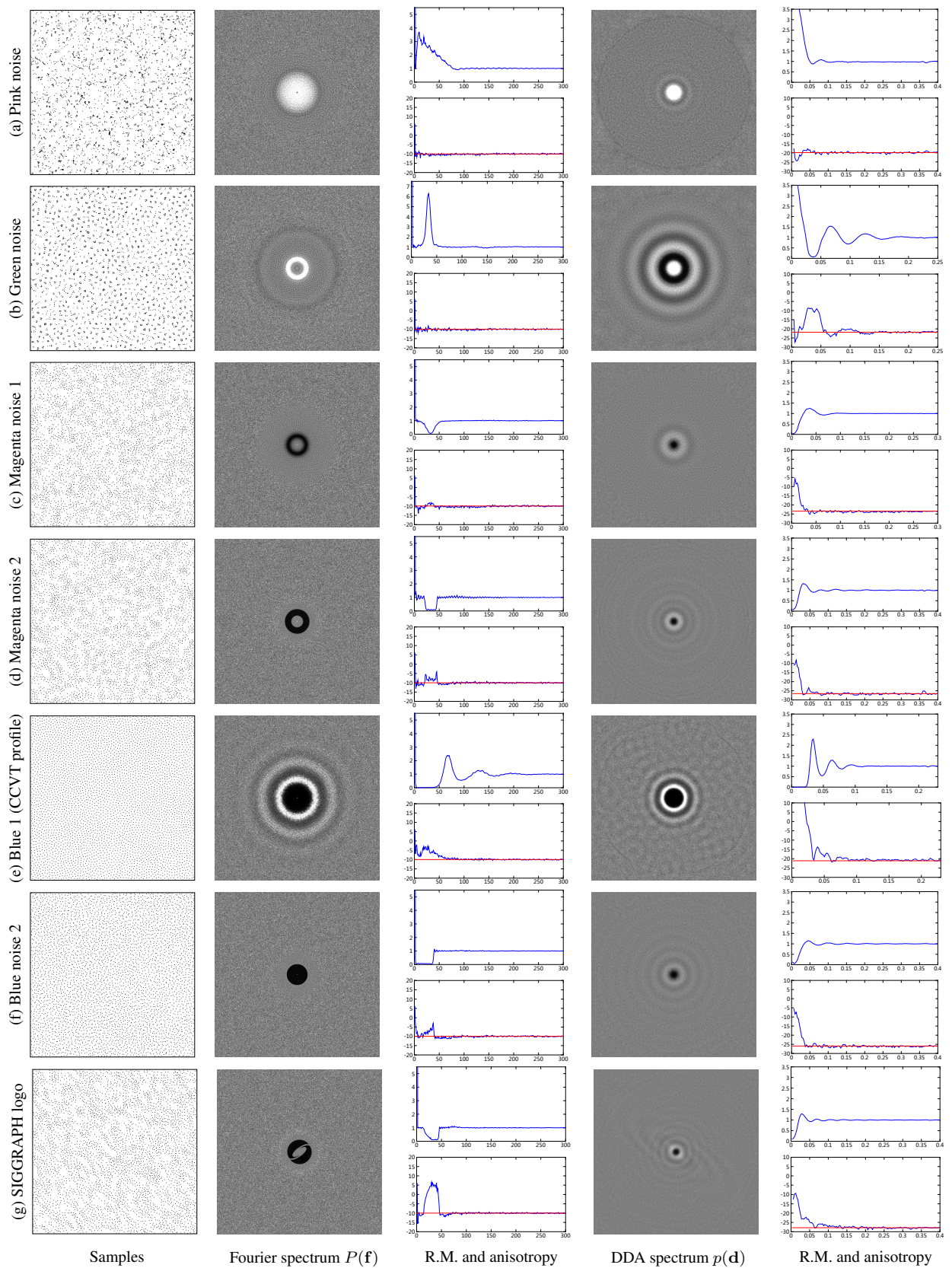
**Properties of  $\Phi(\mathbf{f})$ .** We also assume  $\Phi(\mathbf{f})$  is flat for sufficiently large  $\mathbf{f}$ . In fact,  $\Phi(\mathbf{f})$  as defined by Eq. 2 typically converges to 1, and has a value of  $N$  at the origin (i.e.  $\Phi(\mathbf{0}) = N$ ).

**Converting  $\Phi(\mathbf{f})$  to  $\phi(\mathbf{d})$ .** From Eq. 3 we know that the target Fourier spectrum  $\Phi(\mathbf{f})$  is a cosine transform of  $\phi(\mathbf{d})$ . Therefore it follows that  $\phi(\mathbf{d})$  is an *inverse* cosine transform of  $\Phi(\mathbf{f})$ , which is a cosine transform itself. In other words:

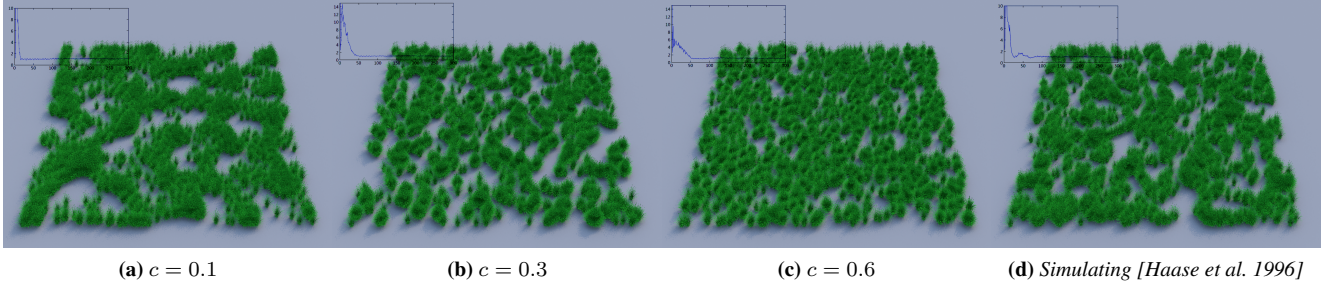
$$\phi(\mathbf{d}) = \int_{\Omega_f} \cos(2\pi \mathbf{d} \cdot \mathbf{f}) \Phi(\mathbf{f}) \delta \mathbf{f}. \quad (12)$$

This can be proved by observing that  $\Phi(\mathbf{f})$  and  $\phi(\mathbf{d})$  are actually related to each other via Fourier and inverse Fourier transforms. But since they are both even functions ( $\Phi(\mathbf{f}) = \Phi(-\mathbf{f})$  and  $\phi(\mathbf{d}) = \phi(-\mathbf{d})$ ), the imaginary (sine) portions of the Fourier and inverse transforms are zeroed out, so both are reduced to cosine transforms.





**Figure 2:** Seven sets of samples generated using our algorithm with different Fourier spectrum profiles. Each set contains 4,000 points. We show the Fourier analysis (averaged over 10 runs) and differential domain analysis [Wei and Wang 2011] results, both including radial means and anisotropy graphs. Among them (e) is generated using the spectrum profile of the CCVT algorithm [Balzer et al. 2009]; the others are all generated with user-plotted spectrum profiles. Note that the anisotropy, which measure the variance relative to radial means, is generally high where the radial means are close to zero.



**Figure 3:** Grass scenes generated from pink noise samples. The input Fourier spectrum in (a-c) is modeled as a Gaussian centered at 0 with various  $c$  (standard deviation) values; (d) is generated using a differential distribution function measured from real tree distributions [Haase et al. 1996]. The radial means of the Fourier spectrum is shown on the upper-left corner of each image.

**Normalization of  $\phi(\mathbf{d})$ .** Eq. 12 works if we account for all sample pairs in the domain, regardless of their distances. However, as we only consider each sample’s local neighbors, additional steps must be performed to normalize  $\phi(\mathbf{d})$  and make it consistent with the  $p(\mathbf{d})$  actually evaluated in the solver. Below are the details.

First, to evaluate Eq. 12, we replace  $\Phi(\mathbf{f})$  by  $\Phi(\mathbf{f}) - 1$ . This is because the integration range of  $\mathbf{f}$  is theoretically infinite, but in practice we must perform numerical integration within the finite range where  $\Phi(\mathbf{f})$  is provided. By subtracting 1 from  $\Phi(\mathbf{f})$ , the resulting function flats out to 0 for sufficiently large  $\mathbf{f}$ , hence the integration can be accurately evaluated.

Next, assume the actually calculated  $p(\mathbf{d})$  according to Eq. 4 is represented as a histogram with  $M$  bins. Then the sum of  $p(\mathbf{d})$  over all bins is simply the average number of points around each sample’s neighborhood  $\Delta_p$ , which is  $\Delta_p^2 N$  if we assume samples are evenly distributed in the domain. To match  $\phi(\mathbf{d})$  to  $p(\mathbf{d})$ , we must make sure the sum of  $\phi(\mathbf{d})$  over all histogram bins is equal to  $\Delta_p^2 N$ . Thus the normalization factor for  $\phi(\mathbf{d})$  is

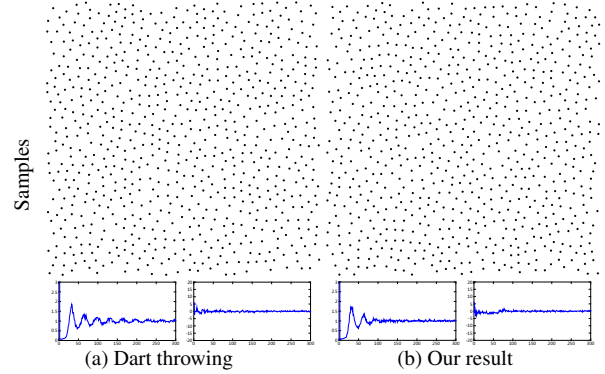
$$s_\phi = \frac{\Delta_p^2 N}{\sum_{\mathbf{d}} \phi(\mathbf{d})}. \quad (13)$$

**Discussion.** Note that given an arbitrary  $\Phi(\mathbf{f})$ , the resulting  $\phi(\mathbf{d})$  may have some negative values, since the cosine transform is not guaranteed to produce positive-only results. These negative values indicate that the provided  $\Phi(\mathbf{f})$  is generally not realizable (unless if the points are specially designed or generated in a non-stationary manner). If this happens, our algorithm will shift  $\phi(\mathbf{d})$  by a constant and re-normalize it, so that the resulting function has no negative values. Alternatively, the user can eliminate the negative portion manually. Note that a constant shift in  $\phi(\mathbf{d})$  only changes the value at  $\Phi(\mathbf{0})$ , but preserves the overall shape of the input  $\Phi(\mathbf{f})$ .

### 4.3 Adaptive Sampling

In adaptive sampling, the sample density changes spatially according to a distance field  $r(\mathbf{s})$ . This can be viewed as local changes to the distance metric. Our algorithm can be easily modified to achieve adaptive sampling by exploiting the invariance of  $p(\mathbf{d})$ . Specifically, as  $p(\mathbf{d})$  is a local quantity, it can be unwarped locally to account for changes in the distance metric, as shown in [Wei and Wang 2011]. For any given noise spectrum, the unwarped  $p(\mathbf{d})$  should be statistically the same everywhere in the domain, as long as the neighborhood size  $\Delta_p$  is small relative to the changes in  $r(\mathbf{s})$ .

Given a grayscale image with spatial intensity value  $I(\mathbf{s})$ , we define  $r(\mathbf{s}) = \frac{1}{\sqrt{I(\mathbf{s})}}$  so that the sample density is proportional to  $I(\mathbf{s})$ . We



**Figure 4:** Comparison of uniform blue noise sampling. (a) shows standard Poisson disk samples generated using dart throwing; (b) shows our result generated using the spectrum profile obtained from (a). The Fourier analysis (including radial means and anisotropy) is shown on the bottom. Note that the radial means of (b) closely resembles that of (a).

first normalize  $r(\mathbf{s})$  so that the average value of  $\frac{1}{r^2(\mathbf{s})}$  (i.e. overall image intensity) is 1.0. We then follow [Wei and Wang 2011] to define the *unwarped* differential  $\tilde{\mathbf{d}}$  between samples  $\mathbf{s}_i$  and  $\mathbf{s}_j$  as

$$\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_j) = \frac{2(\mathbf{s}_i - \mathbf{s}_j)}{r(\mathbf{s}_i) + r(\mathbf{s}_j)}. \quad (14)$$

In other words, it divides  $(\mathbf{s}_i - \mathbf{s}_j)$  by the average distance field value between them. If  $r(\mathbf{s})$  is uniform,  $\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_j)$  will be equal to  $(\mathbf{s}_i - \mathbf{s}_j)$ . Following this, we modify the calculations of  $p(\mathbf{d})$ ,  $\frac{\partial E_p}{\partial \mathbf{s}_k}$ , and  $\Delta t$  in Equations 4, 9, and 11 by replacing  $(\mathbf{s}_i - \mathbf{s}_j)$  with  $\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_j)$ . Note that the target  $\phi(\mathbf{d})$  remains unchanged.

To calculate  $p(\mathbf{d})$ , Eq. 4 now becomes

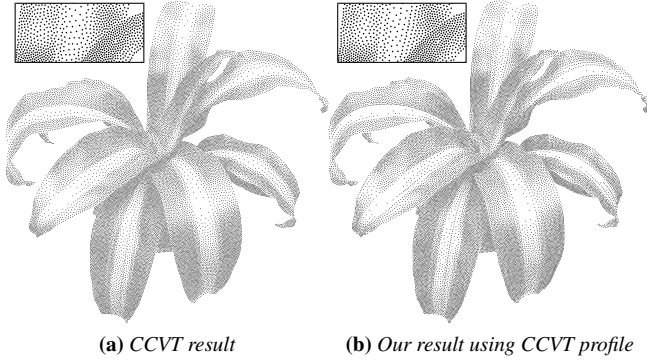
$$p(\mathbf{d}) = \kappa(\mathbf{d}, \mathbf{0}) + \frac{1}{N} \sum_{i=0}^{N-1} \sum_{\substack{j=0 \\ j \neq i}}^{N-1} \kappa(\mathbf{d}, \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_j)). \quad (15)$$

To calculate  $\frac{\partial E_p}{\partial \mathbf{s}_k}$ , Eq. 9 now becomes:

$$\frac{\partial E_p}{\partial \mathbf{s}_k} = \frac{4}{N} \sum_{i \neq k}^{N-1} J \times [(p(\mathbf{d}) - \phi(\mathbf{d})) * \kappa'(\mathbf{d})] (\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k)), \quad (16)$$

where  $J$  is the  $2 \times 2$  Jacobian matrix that comes from differentiating





**Figure 5:** Comparison of blue noise stippling. (a) shows the result using the CCVT algorithm [Balzer et al. 2009]; (b) shows our result using CCVT spectrum profile. Both are computed with 20,000 points. This example demonstrates the capability of our algorithm to simulate existing algorithms by using their spectrum profiles as input.

$\kappa(\mathbf{d}, \tilde{\mathbf{d}}(\mathbf{s}_k, \mathbf{s}_j))$  and  $\kappa(\mathbf{d}, \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k))$  against  $\mathbf{s}_k$ . It is defined by

$$J = \frac{1}{r(\mathbf{s}_i) + r(\mathbf{s}_k)} \begin{bmatrix} r_x(\mathbf{s}_i) \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k) \cdot x + 2 & r_x(\mathbf{s}_i) \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k) \cdot y \\ r_y(\mathbf{s}_i) \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k) \cdot x & r_y(\mathbf{s}_i) \tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k) \cdot y + 2 \end{bmatrix},$$

where  $r_x$  and  $r_y$  denote the partial derivatives of the distance field  $r(\mathbf{s})$  in the  $x$  and  $y$  directions respectively;  $\tilde{\mathbf{d}} \cdot x$  and  $\tilde{\mathbf{d}} \cdot y$  denote the  $x$  and  $y$  components of  $\tilde{\mathbf{d}}$ . Finally, we scale the step size  $\Delta t$  in Eq. 11 by  $r(\mathbf{s}_k)$  to make sure it is proportional to the local density.

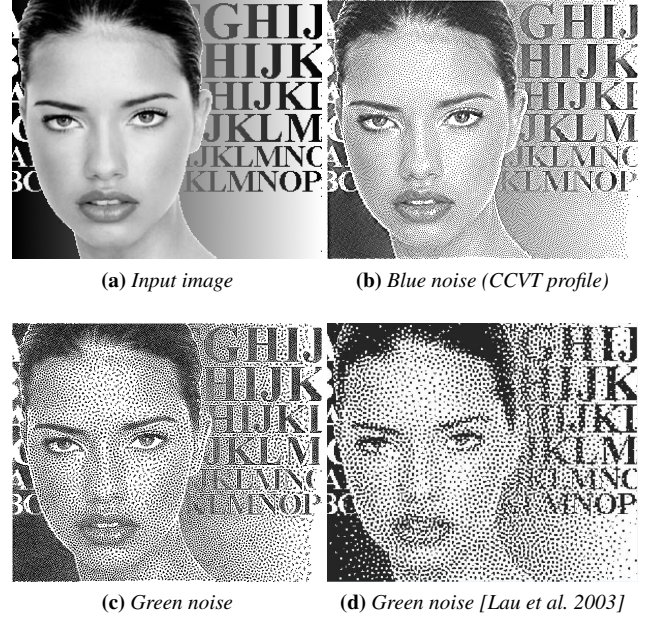
#### 4.4 Implementation Details

Our algorithm is implemented on the GPU using NVIDIA’s CUDA programming language and Thrust library. During each iteration, every sample point can be updated simultaneously, thus we achieve a considerable speedup by exploiting the GPU. To begin, the user will provide the target  $\Phi(\mathbf{f})$ , set the desired number of samples  $N$  as well as the neighborhood size  $\Delta_p$ . Our algorithm will compute and normalize the target  $\phi(\mathbf{d})$  as described in Section 4.2. For adaptive sampling, given an input grayscale image, we use the reciprocal of the square root of each pixel’s intensity value to define the distance field  $r(\mathbf{s})$ . This along with its derivative images  $r_x, r_y$  are stored in GPU’s texture memory for fast access with bilinear interpolation.

**Sample initialization.** The initial samples are computed on the CPU as white noise, and their density is proportional to the distance field. We then upload them to the GPU as a CUDA array.

**Computing  $p(\mathbf{d})$ .** To estimate the differential distribution  $p(\mathbf{d})$  of the current sample set, we first compute a 2D histogram  $h(\mathbf{d})$  of the sample differences  $(\mathbf{s}_i - \mathbf{s}_j)$  without the Gaussian kernel. To do so, we create a  $128 \times 128$  array in global memory to represent  $h(\mathbf{d})$ , then launch  $N$  threads corresponding to the number of samples. Each thread  $k$  finds sample  $k$ ’s spatial neighbors (i.e. all samples  $\mathbf{s}_i$  such that  $\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k) \leq \Delta_p$ ), and atomically increments the histogram value stored at  $h(\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k))$ . A uniform grid data structure is used to accelerate the search for neighbor samples. The resulting  $h(\mathbf{d})$  is then divided by  $N$  and convolved with a Gaussian kernel of  $\sigma = 2$  to produce  $p(\mathbf{d})$ .

**Sample update.** Next, we compute each sample’s gradient defined by Eq. 16. We first compute  $[p(\mathbf{d}) - \phi(\mathbf{d})]$  and convolve it



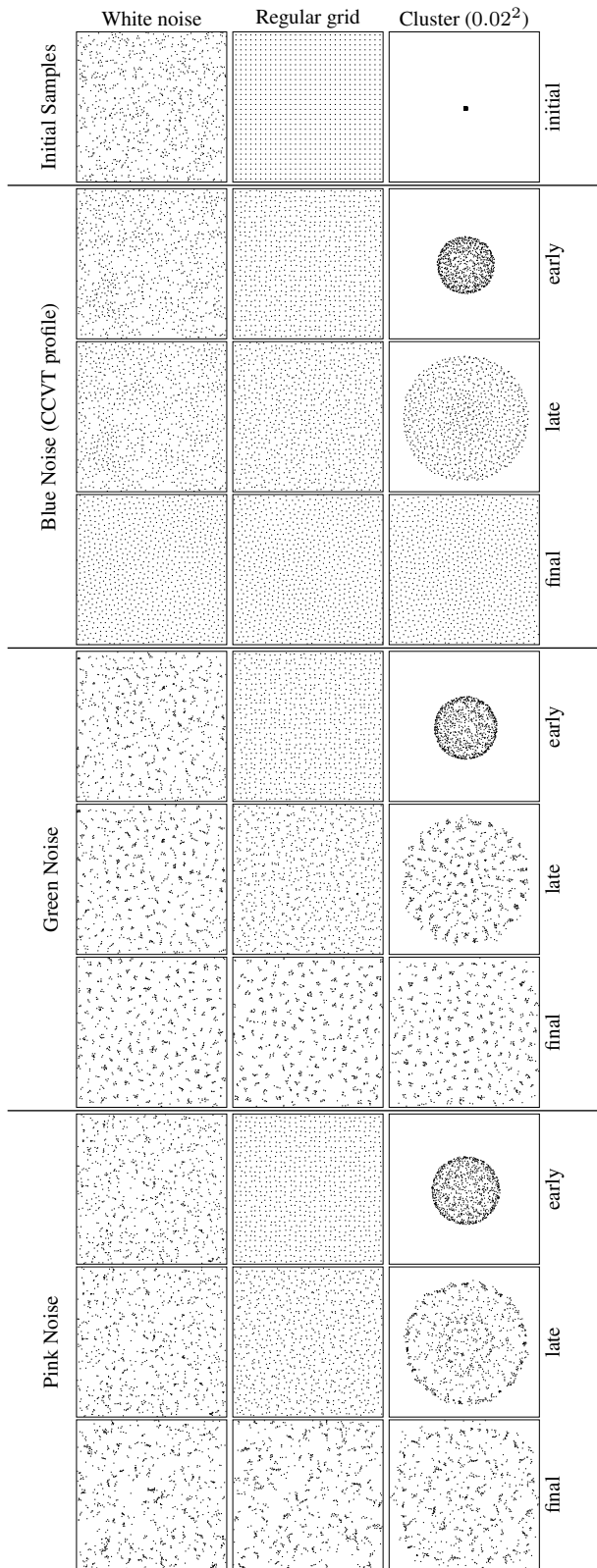
**Figure 6:** Comparing green noise and blue noise stippling results. Both (b) and (c) are generated using our algorithm. Compared to blue noise, green noise produces clustered dot effects that can be essential for practical or aesthetic reasons. (d) shows result from a previous green noise halftoning method, which applies in discrete (pixel) domain only.

with  $\kappa'(\mathbf{d})$  (Eq. 8). The result is stored in GPU texture memory. Then we launch  $N$  threads again: using the uniform grid structure, each thread  $k$  finds sample  $k$ ’s spatial neighbors  $\mathbf{s}_i$ , calculates  $\tilde{\mathbf{d}}(\mathbf{s}_i, \mathbf{s}_k)$  to index into the convolved texture, and sums up the result according to Eq. 16. This produces  $\frac{\partial E_p}{\partial \mathbf{s}_k}$ . Finally, we update each sample’s position according to Eq. 10.

## 5 Results and Applications

Figure 2 shows a number of samples sets computed using our algorithm with different noise spectra. These include pink noise, green noise, blue noise, magenta noise (whose Fourier spectrum is the complement of green noise), and a SIGGRAPH noise (whose Fourier spectrum is shaped like the SIGGRAPH logo). For each set we show the Fourier spectrum analysis, including radial means and anisotropy, as well as the differential domain analysis. For most of them, the target Fourier spectrum is plotted by the user with 1D curves. One exception is (e), where the spectrum profile of the CCVT algorithm [Balzer et al. 2009] is used as input.

**Initial Sample Patterns.** While we typically initialize samples as white noise, it is important to examine the behavior of our algorithm under different initial sample patterns. In Figure 7 we shows the results under three initial sample patterns (white noise, regular grid, and a cluster where samples are randomly distributed in the center  $0.02 \times 0.02$  square) and for three different types of noise. For each example we show the final result as well as two intermediate results before convergence. Note that in most cases the result is insensitive to the initial sample pattern. However, in the cluster case, the pink and green noise results fail to cover the entire domain. This is mainly because our algorithm currently does not enforce the sample distribution to be stationary or spatially uniform. This can cause the solver to terminate even when the spatial density is not uniform



**Figure 7:** Initial sample patterns. We demonstrate the behavior of our algorithm under different initial sample patterns (shown on the top) and for different types of noise. For each example we show the final result and two intermediate results before convergence. In most cases the algorithm is insensitive to initial samples, except in the cluster case where the green and pink noise results did not fully cover the entire domain.

yet across the entire domain. We have found that increasing the neighborhood size  $\Delta_p$  or total sample count  $N$  can help improve the results in such cases.

**Performance.** Table 1 lists the performance of our GPU implementation for five types of noise selected from Figure 2. The timings are reported on a PC with 3.0 GHz quad-core CPU and an NVIDIA GTX 480 graphics card. The primary bottleneck is in the computation of  $p(\mathbf{d})$ , which requires an atomic operation on the GPU for estimating  $h(\mathbf{d})$ . Pink noise is the slowest to compute, mainly because the samples in pink noise tend to form concentrated clusters, increasing the average number of neighbors per sample hence increasing the computation time.

In Figure 8 we show the decay of error values during the computation. The plotted values are the relative error (i.e. Eq. 6 divided by the integral of  $\phi(\mathbf{d})$  for normalization purpose) averaged over every 200 iterations. In most cases the error decreases rapidly. Note that green noise error curve in the right figure has a slight increase beyond 2,000 iterations. This happens because the per-iteration error in this example oscillates within a wider range [0.15%-0.3%] in the beginning and smaller range [0.2%-0.25%] towards the end, causing the average error to increase slightly past 2000 iterations.

**Red/pink noise.** A variety of natural plant distributions exhibit a clustering behavior that can be characterized by red/pink noise [Greene and Johnson 1989; Schua et al. 2009]. Our method can be used to produce such distributions, as shown in Figure 3. We place a grass object centered at each sample to generate a grass scene. The input Fourier spectrum is modeled as a Gaussian centered at 0 with standard deviation  $c$ . Smaller  $c$  produces more concentrated clusters, while larger  $c$  produces more scattered clusters, as evident from the results. In (d) we have applied a differential distribution function measured from real tree distributions [Haase et al. 1996]. By using a measured function from real scenes, our algorithm can create a synthetic scene with similar distributions.

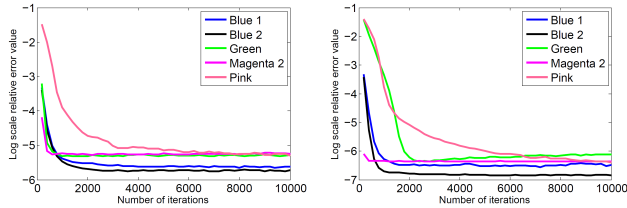
**Blue noise.** Our method can also generate blue noise. In Figure 2(e) we applied the CCVT spectrum profile as input to our algorithm to generate uniform samples; similarly in Figure 4 we applied the Poisson disk spectrum profile as input for uniform sampling. Both result in high-quality blue noise samples and demonstrate the ability of our algorithm to faithfully simulate existing algorithms. In Figure 5 we show an example of adaptive sampling using the CCVT profile for stippling. Compared to the original CCVT algorithm, our method produces results with comparable quality.

In Figure 2(f) we applied a synthetic blue noise spectrum, which is a step function that changes sharply from 0 to 1 at a given cutoff frequency. Note that unlike Poisson disk or CCVT, the corresponding  $\phi(\mathbf{d})$  of this synthetic blue noise does not contain a region of zeros (no minimum distance constraint). Thus there is a non-zero probability for samples to be arbitrarily close. This effect is also evident in the manipulated blue noise examples found in [Parker et al. 1991]. Note that the minimum distance constraint is a special property by Poisson disk or similar algorithms, but is not inherent in the definition of blue noise. If this property is desired, we can manually modify  $\phi(\mathbf{d})$  to include a region of zeros in the center.

**Green noise.** Green noise has been shown [Lau et al. 2003] to benefit clustered-dot printing, with the goal of balancing spatial uniformity and separation with device limitations. For example, some devices lack the support for printing dispersed dots as required by blue noise. However, prior green noise methods are applicable only to a discrete (pixel) domain. Our method does not have such re-

Time / 100 ite	Blue 1	Blue 2	Green	Magenta 2	Pink
1,000 points	0.29s	0.31s	0.25s	0.24s	0.25s
5,000 points	0.53s	0.45s	0.73s	0.72s	0.75s
Total time	Blue 1	Blue 2	Green	Magenta 2	Pink
1,000 points	12.0s	14.1s	2.3s	3.1s	9.3s
5,000 points	22.6s	14.5s	6.4s	4.7s	36.3s

**Table 1:** Performance of our GPU algorithm. The top three rows show the computation time per 100 iterations for generating 1000 and 5000 points respectively. The bottom three rows show the total time upon convergence.



**Figure 8:** Error decay curves for different types of noise. These plots show the decay of relative error values (in log scale) with respect to the number of iterations when generating 1,000 points (left) and 5,000 points (right).

striction and can generate samples in a continuous domain.

In Figure 6, we apply green noise for image stippling, a popular application for blue noise sampling techniques [Secord 2002; Balzer et al. 2009; Wei 2010; Fattal 2011; Li and Mould 2011]. Compared to blue noise, which produces isolated dots, green noise tends to produce clustered dots, and the centers of the clusters are distributed with uniform separation, similar to blue noise. This type of stippling provides different flavors from blue noise, and can be important for practical and aesthetic reasons.

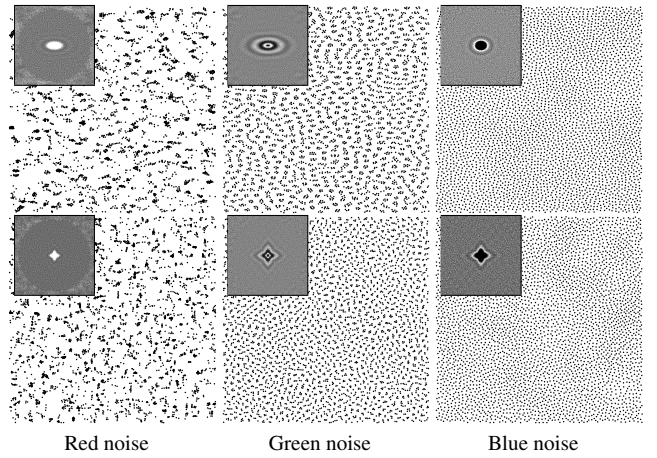
To create green noise spectrum, we model its 1D radial means as a Gaussian centered at  $\mu$  (location of the middle frequency) with standard deviation  $c$  (bandwidth). Through experiments we have found that  $\mu$  generally affects the spatial cluster size: higher  $\mu$  produces smaller clusters, while  $c$  generally affects the variation in the cluster sizes. See Figure 11 for detailed experimental results. This relationship between spectral and spatial characteristics allows us to select suitable parameters for specific target applications.

**Anisotropic noise.** Our algorithm can also work with anisotropic (i.e. radially asymmetric) spectrum profiles. As shown in [Lagae et al. 2009; Li et al. 2010], noise with anisotropic spectrum is important for a variety of applications in sampling and filtering. Prior anisotropic noise methods have been restricted to specific spectrum types, such as blue noise in [Li et al. 2010] and Gabor noise in [Lagae et al. 2009; Lagae and Drettakis 2011]. In contrast, our algorithm can produce anisotropic noise with different spectral properties. For example, in Figure 9, we have applied deformed red, green, and blue noise profiles to produce samples with desired spectrum shapes. Note that the deformed spectrum must still satisfy that  $\Phi(-\mathbf{f}) = \Phi(\mathbf{f})$ , an intrinsic property of  $\Phi(\mathbf{f})$ .

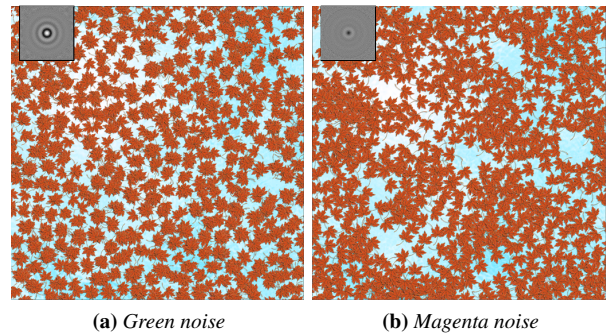
**Other examples.** Figure 10 shows two additional examples where green noise and magenta noise are used to synthesize textures by placing texture elements centered at every sample point.

## 6 Limitations and Future Work

As our main contribution, we have described a general algorithm for producing point samples that match a given Fourier power spec-



**Figure 9:** Examples of anisotropic noise. The spectrum is shown on the upper-left corner of each image. Note the visual differences in the sample distributions corresponding to the spectrum differences.



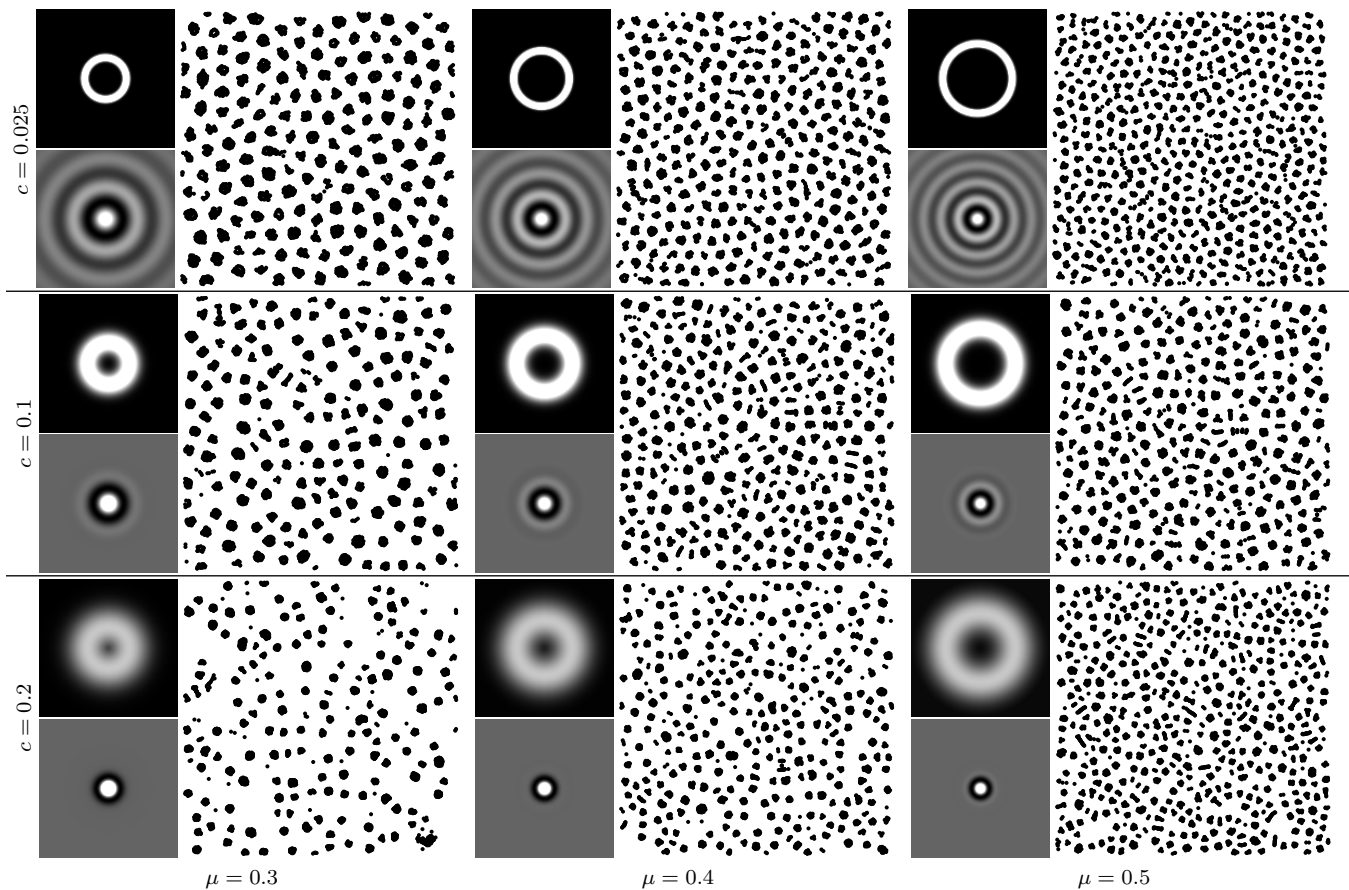
**Figure 10:** Synthesized textures using green and magenta noise. A leaf texture is placed at each sample point and randomly rotated. The differential distribution function of each sample set is shown on the upper-left corner.

trum, including ones that are constructed by the user. Due to the ambiguity of the Fourier power spectrum, there exist many solutions for each given spectrum. Our solver provides one possible solution by iteratively minimizing an error function. But we do not claim it is the best. In the future we plan to incorporate additional measures, such as stationary property, into the objective function in order to provide more control over the solution.

Our current anisotropic results (Section 5) are produced by directly deforming the spectrum shapes. A potential future direction is to impose such anisotropy from the application domain directly, such as surface mapping or anisotropic stippling as demonstrated in [Li et al. 2010]. This will require extending our current solver, which handles up to isotropic adaptive sampling, for general anisotropic sampling controlled by local Jacobians [Li et al. 2010].

Finally, we would like to seek further applications of noise patterns with arbitrary spectral properties. Many results produced by our method are never seen before, thus they may not have immediate applications. Nonetheless, we believe they can inspire future work and their importance will become more evident in the near future. For example, in coded aperture imaging, we may define an ideal frequency response of the aperture, and use our method to compute the corresponding coded aperture patterns. Similarly, in image processing, we may use our algorithm to compute optimal image





**Figure 11:** Comparison of green noise results by varying the spectrum parameters. Here the 1D radial means curve of each spectrum is modeled as a Gaussian centered at  $\mu$  with standard deviation  $c$ . For each example we show the target Fourier power spectrum (top left), differential distribution function (lower left), and the resulting sample points. Note that  $\mu$  (negatively) affects the spatial cluster size, while  $c$  (positively) affects the variation in the cluster sizes.

sampling patterns. We hope that the availability of our method will enable researchers to explore such potential applications.

**Acknowledgements.** We thank SIGGRAPH reviewers for comments and suggestions, Daniel Lau of providing the source image of Figure 6, and Andrew McGregor for discussions. This work was supported by the National Science Foundation grants CCF-0746577 and CCF-1025120, as well as the University of Hong Kong.

## References

- ALLIEZ, P., MEYER, M., AND DESBRUN, M. 2002. Interactive geometry remeshing. In *SIGGRAPH '02*, 347–354.
- BALZER, M., SCHLOMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: A variant of Lloyd’s method. In *SIGGRAPH '09*, 86:1–8.
- BRACEWELL, R. 1999. *The Fourier Transform and Its Applications*. McGraw-Hill.
- CHANG, J., ALAIN, B., AND OSTROMOUKHOV, V. 2009. Structure-aware error diffusion. In *SIGGRAPH Asia '09*, 162:1–8.
- CONDIT, R., ASHTON, P. S., BAKER, P., BUNYAVEJCHEWIN, S., GUNATILLEKE, S., GUNATILLEKE, N., HUBBELL, S. P., FOSTER, R. B., ITOH, A., LAFRANKIE, J. V., LEE, H. S., LOSOS, E., MANOKARAN11, N., SUKUMAR, R., AND YAMAKURA, T. 2000. Spatial patterns in the distribution of tropical tree species. *Science* 288, 5470, 1414–1418.
- COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph.* 5, 1, 51–72.
- DALE, M. R. T., DIXON, P., FORTIN, M.-J., LEGENDRE, P., MYERS, D. E., AND ROSENBERG, M. S. 2002. Conceptual and mathematical relationships among methods for spatial analysis. *ECOGRAPHY*, 25, 558–577.
- DIPPÉ, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. In *SIGGRAPH '85*, 69–78.
- DUTRE, P., BALA, K., AND BEKAERT, P. 2002. *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, USA.
- EBEIDA, M. S., PATNEY, A., MITCHELL, S. A., DAVIDSON, A., KNUPP, P. M., AND OWENS, J. D. 2011. Efficient maximal Poisson-disk sampling. In *SIGGRAPH '11*, 49:1–12.
- EBEIDA, M. S., MITCHELL, S. A., PATNEY, A., DAVIDSON, A., AND OWENS, J. D. 2012. A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Computer Graphics Forum* 31, 2, 785–794.



- FATTAL, R. 2011. Blue-noise point sampling using kernel density model. *ACM SIGGRAPH 2011 papers* 28, 3, 1–10.
- GREENE, D. F., AND JOHNSON, E. A. 1989. A model of wind dispersal of winged or plumed seeds. *Ecology* 70, 2, 339–347.
- HAASE, P., PUGNAIRE, F. I., CLARK, S., AND INCOLL, L. 1996. Spatial patterns in a two-tiered semi-arid shrubland in southeastern Spain. *Journal of Vegetation Science*, 7, 527–534.
- KALANTARI, N. K., AND SEN, P. 2011. Efficient computation of blue noise point sets through importance sampling. *Computer Graphics Forum (EGSR '11)* 30, 4, 1215–1221.
- KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive Wang tiles for real-time blue noise. In *SIGGRAPH '06*, 509–518.
- LAGAE, A., AND DRETTAKIS, G. 2011. Filtering solid Gabor noise. In *SIGGRAPH '11*, 51:1–6.
- LAGAE, A., AND DUTRÉ, P. 2006. An alternative for Wang tiles: colored edges versus colored corners. *ACM Trans. Graph.* 25, 4, 1442–1459.
- LAGAE, A., AND DUTRÉ, P. 2008. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum* 21, 1, 114–129.
- LAGAE, A., LEFEBVRE, S., DRETTAKIS, G., AND DUTRÉ, P. 2009. Procedural noise using sparse Gabor convolution. In *SIGGRAPH '09*, 54:1–10.
- LAU, D. L., ARCE, G. R., AND GALLAGHER, N. C. 1999. Digital halftoning by means of green-noise masks. 1575–1586.
- LAU, D., ULICHNEY, R., AND ARCE, G. 2003. Fundamental characteristics of halftone textures: blue-noise and green-noise. *IEEE Signal Processing Magazine* 20, 4, 28–38.
- LI, H., AND MOULD, D. 2011. Structure-preserving stippling by priority-based error diffusion. In *GI '11*, 127–134.
- LI, H., WEI, L.-Y., SANDER, P., AND FU, C.-W. 2010. Anisotropic blue noise sampling. In *SIGGRAPH Asia '10*, 167:1–12.
- LLOYD, S. 1983. An optimization approach to relaxation labeling algorithms. *Image and Vision Computing* 1, 2.
- MARTINEZ, V. J., PAREDES, S., BORGANI, S., AND COLES, P. 1995. Multiscaling properties of large-scale structure in the universe. *Science* 269, 5228, 1245–1247.
- MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *SIGGRAPH '87*, 65–72.
- OSTLING, A., HARTE, J., AND GREEN, J. 2000. Self-similarity and clustering in the spatial distribution of species. *Science* 290, 5492, 671.
- OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH '04*, 488–495.
- OSTROMOUKHOV, V. 2001. A simple and efficient error-diffusion algorithm. In *SIGGRAPH '01*, 567–572.
- OSTROMOUKHOV, V. 2007. Sampling with polyominoes. In *SIGGRAPH '07*, 78.
- ÖZTIRELI, A. C., ALEXA, M., AND GROSS, M. 2010. Spectral sampling of manifolds. In *SIGGRAPH ASIA '10*, 168:1–8.
- PANG, W.-M., QU, Y., WONG, T.-T., COHEN-OR, D., AND HENG, P.-A. 2008. Structure-aware halftoning. In *SIGGRAPH '08*.
- PARKER, K., MITSU, T., AND ULICHNEY, R. 1991. A new algorithm for manipulating the power spectrum of halftone patterns. In *SPSE's 7th Int. Congress on Non-Impact Printing*, 471–475.
- PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc.
- SCHLÖMER, T., HECK, D., AND DEUSSEN, O. 2011. Farthest-point optimized point sets with maximized minimum distance. In *HPG '11*, 135–142.
- SCHROEDER, M. R. 1999. *Computer Speech: Recognition, Compression, Synthesis*. Springer.
- SCHUA, K., FEGER, K.-H., WAGNER, S., EISENHAUER, D.-R., AND RABEN, G. 2009. Cause-Effect Relations with Regard to Functional and Morphological Humus Characteristics in Mixed Forest Stands. *EGU General Assembly 2009 11* (Apr.), 226.
- SECORD, A. 2002. Weighted Voronoi stippling. In *NPAR '02*, 37–43.
- SHIRLEY, P. 1991. Discrepancy as a quality measure for sample distributions. In *Eurographics '91*, 183–194.
- TURK, G. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH '92*, 55–64.
- TZENG, S., AND WEI, L.-Y. 2008. Parallel white noise generation on a GPU via cryptographic hash. In *ISD '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, 79–87.
- ULICHNEY, R. 1987. *Digital Halftoning*. MIT Press, Cambridge, MA.
- WEI, L.-Y., AND WANG, R. 2011. Differential domain analysis for non-uniform sampling. In *SIGGRAPH '11*, 50:1–10.
- WEI, L.-Y. 2010. Multi-class blue noise sampling. In *SIGGRAPH '10*, 79:1–8.
- YELLOTT, J. I. J. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science* 221, 382–385.
- ZHOU, B., AND FANG, X. 2003. Improving mid-tone quality of variable-coefficient error diffusion using threshold modulation. In *SIGGRAPH '03*, 437–444.