

Multi-Class Blue Noise Sampling

Li-Yi Wei

Microsoft Research

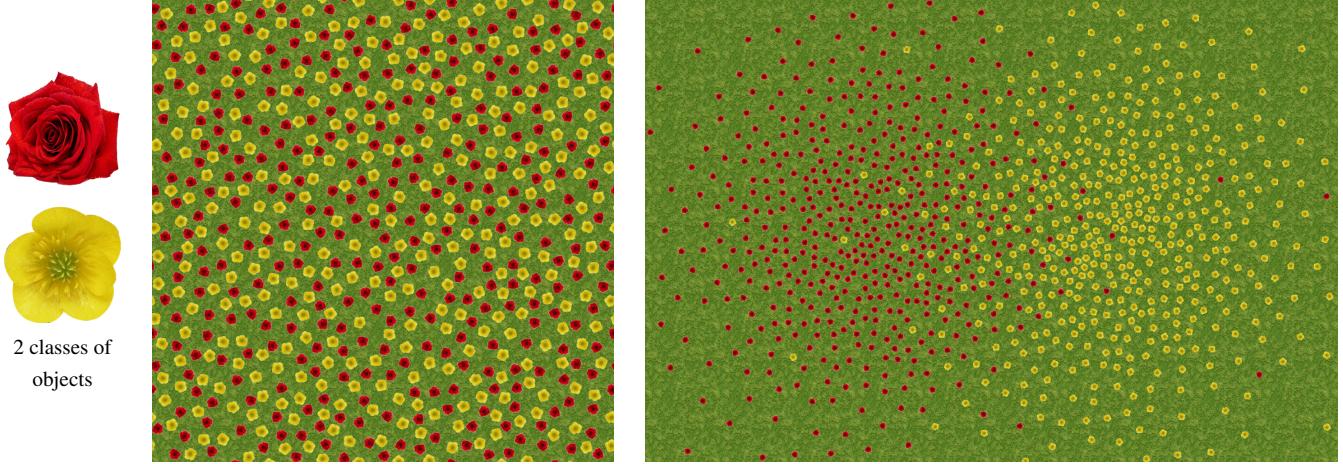


Figure 1: Object placement by multi-class blue noise sampling. Our algorithms can produce both uniform (middle) and adaptive (right) sampling results.

Abstract

Sampling is a core process for a variety of graphics applications. Among existing sampling methods, blue noise sampling remains popular thanks to its spatial uniformity and absence of aliasing artifacts. However, research so far has been mainly focused on blue noise sampling with a single class of samples. This could be insufficient for common natural as well as man-made phenomena requiring multiple classes of samples, such as object placement, imaging sensors, and stippling patterns.

We extend blue noise sampling to multiple classes where each individual class as well as their unions exhibit blue noise characteristics. We propose two flavors of algorithms to generate such multi-class blue noise samples, one extended from traditional Poisson *hard* disk sampling for explicit control of sample spacing, and another based on our *soft* disk sampling for explicit control of sample count. Our algorithms support uniform and adaptive sampling, and are applicable to both discrete and continuous sample space in arbitrary dimensions. We study characteristics of samples generated by our methods, and demonstrate applications in object placement, sensor layout, and color stippling.

Keywords: multi-class, blue noise, sampling, Poisson hard/soft disk, dart throwing, relaxation

1 Introduction

Sampling is important for a variety of graphics applications, including rendering, imaging, and geometry processing. Although differ-

ent applications may favor different sampling patterns, blue noise sampling remains quite popular and is widely adopted. Inspired by the distribution of primate retina cells [Yellott 1983], a blue noise distribution contains samples that are randomly located but remain spatially uniform. The resulting sample set has a blue noise power spectrum, with a signature lack of low frequency energy and structural residual peaks. Beyond a blue noise power spectrum, additional desiderata for graphics applications include support for adaptive/importance sampling, efficient computation, as well as the ability to place samples in both discrete and continuous sample spaces.

Due to its importance, blue noise sampling has been researched extensively [Cook 1986; Mitchell 1987; McCool and Fiume 1992; Ostromoukhov et al. 2004; Jones 2006; Dunbar and Humphreys 2006; Kopf et al. 2006; Ostromoukhov 2007; Bridson 2007; White et al. 2007; Wei 2008; Fu and Zhou 2008; Balzer et al. 2009; Cline et al. 2009]. However, prior methods are mainly concerned about a single class of samples, and thus are not directly applicable to a variety of natural or man-made phenomena involving multiple classes of samples, such as the distribution of cone and rod cells in human retinas, the placement of multiple categories of objects, and the usage of multiple colored dots for stippling. In these situations, it is often desirable to have each individual class of samples as well as their union to exhibit blue noise distribution. Unfortunately, such simultaneous blue noise properties across multiple classes of samples are not guaranteed by previous single-class sampling methods. See Figure 2 for an example.

We present multi-class blue noise sampling algorithms that not only guarantee a blue noise spectrum for each individual class as well as their unions, but also allow samples to be placed in both discrete and continuous sample spaces. Our methods also support adaptive sampling and arbitrary sample space dimensionality.

To support different application scenarios, we present two flavors of algorithms: one derived from Poisson *hard* disk sampling [Cook 1986] for explicit control of sample spacing, and another based on our *soft* disk sampling for explicit control of sample count as in relaxation [Lloyd 1982]. (In a nutshell, a hard disk, centered on each sample, can neither deform nor intersect another, while a soft disk can intersect another, but subject to an energy penalty which, when minimized, produces uniform distribution.) Our main

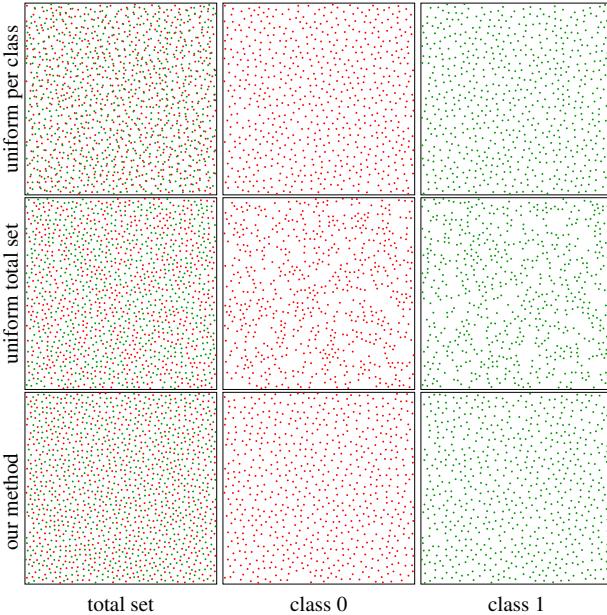


Figure 2: Comparison between single- and multi-class blue noise sampling. The top row is produced by applying single-class dart throwing to individual classes, but the total set is highly non-uniform. The middle row is produced by applying single-class dart throwing to the total set, but the individual classes are highly non-uniform. Our approach produces samples that exhibit blue noise distribution for each class as well as the total set. Each class contains ~ 650 samples generated with $r = 0.02$.

idea is to extend single-class dart throwing for multi-class soft/hard disk sampling by replacing the spacing parameter r in the former with a $c \times c$ symmetric matrix \mathbf{r} for c sample classes. Since many blue-noise sampling methods are descendants of these two seminal algorithms [Lloyd 1982; Cook 1986] with different quality, performance, and usage tradeoffs, having multi-class extensions with both flavors could benefit different applications. Several such applications we show include object distribution [Cohen et al. 2003; Lagae and Dutré 2005], stippling [Kopf et al. 2006; Balzer et al. 2009], sensor layout and color filter design [Ben Ezra et al. 2007], involving both continuous/discrete sample space, uniform/adaptive sampling, control for sample spacing/count, and preferences for spatial-uniformity/spectrum-quality.

Our method is related to color halftoning and especially vector error diffusion [Baqai et al. 2005; Pang et al. 2008], which can also produce multiple classes of blue noise samples. However, our method differs from color halftoning in several significant ways. First of all, halftoning methods are mainly about computing colors for a given set of discrete samples, not for general purpose sampling in graphics that might require computing both color and position information in either a discrete or continuous domain. For halftoning methods that rely on limited neighborhood sizes such as dithering (predetermined masks) or error diffusion (predetermined distribution coefficients), such regular discretization could be undesirable (see e.g. [Alliez et al. 2003; Ostromoukhov et al. 2004]). Simply increasing the output resolution may not eliminate these discretization artifacts because this would reduce the effective spatial extent of the fixed neighborhoods. Thus, even in the traditional single class setting, halftoning is not a replacement for general blue noise sampling. Furthermore, even though certain halftoning techniques like error diffusion have implied blue noise properties, there is no guarantee that this will be carried over in the multi-class setting. To our knowledge, the best halftoning methods for generating multi-class blue noise samples rely on iterative optimization (e.g. the pioneering work of [Wang and Parker 1999]), which is often

slow/complex and restricted to uniform/regular/discrete sampling.

2 Multi-Class Hard Disk Sampling

Dart throwing is a classical algorithm [Cook 1986] for Poisson *hard* disk sampling, a particular kind of blue noise distribution where samples are not only randomly and uniformly distributed but remain at least a minimum distance r away from each other. In dart throwing, a trial sample is drawn randomly from the entire domain. If the sample is not within a user-specified distance r from any other existing samples, it is accepted. Otherwise, it is rejected. This process is repeated until reaching certain termination criteria, e.g. a target number of samples and/or a maximum number of trials.

Our multi-class hard disk sampling algorithm follows a similar process, with necessary extensions to handle multiple classes of samples. Specifically, instead of a single number r , the user specifies a set of numbers $\{r_i\}_{i=0:c-1}$ for the c classes of samples. During the sample generation process, instead of checking whether a new trial sample is at least r away from all existing samples, we use a (symmetric) $c \times c$ matrix \mathbf{r} for conflict check, where two samples in classes k and j have to be at least $\mathbf{r}(k, j)$ away from each other. This \mathbf{r} -matrix is built from $\{r_i\}$ with each diagonal entry $\mathbf{r}(i, i) = r_i$. Finally, we also need to determine the class for each new trial sample. Below we describe the algorithm in detail.

2.1 Sample class

For multi-class sampling, we have to decide from which class to sample for the next trial. To ensure that each class is well sampled throughout the entire process, we always pick the next trial sample from the class that is currently most under-filled. We measure this via *fill rate*, defined as the number of existing samples for a particular class over the target number of samples for that class. To maintain an equal fill rate across different classes, we define N_i , the target sample number of class i , as follows:

$$N_i = N \frac{\frac{1}{r_i^n}}{\sum_{j=0}^{c-1} \frac{1}{r_j^n}} \quad (1)$$

where N is the total number of target samples, n the sample space dimension, and $\{r_i\}$ the specified per-class minimum distances. Not maintaining equal fill rates among all classes can easily lead to non-uniform sample distribution.

2.2 Sample control

To ensure easy usage and a uniform sample distribution, it is desirable to generate the classes together (instead of one after another) and maintain a consistent fill rate among different classes throughout the sampling process. (We present a more detailed analysis in Section 4.) However, always drawing the next trial sample from the most under-filled class (Section 2.1) alone is not enough to achieve this goal, as it may be unable to find a new sample not in conflict with existing ones. This can happen quite early in the process when the output distribution is far from being uniform, so we cannot simply stop there. One possible remedy is to accept a trial sample s if it fails to be accepted for the most under-filled class but succeeds for another one. However, as shown in the left case of a simple 2-class experiment in Figure 3, even though the classes might maintain consistent fill-rates throughout the early part of the process, in the end the fill-rates may become unbalanced as eventually it becomes difficult for the class with a larger r value to compete with another with smaller r .

Another possibility is to tune the relative probability to sample from each class to achieve the desired fill rates at the end of the process, as shown in the middle case of Figure 3 (notice the two curves

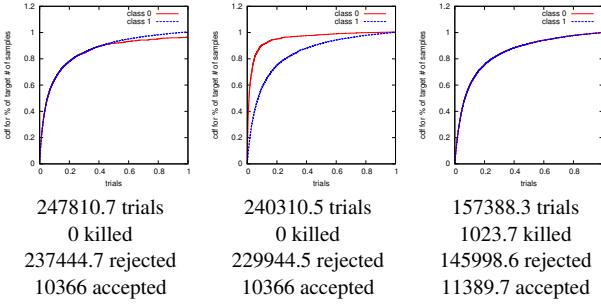


Figure 3: Sampling history comparison. Here, we plot the fill-rates for two classes throughout the sampling process (with the number of trials normalized to [0 1]). $r_0 = 0.02$ and $r_1 = 0.00756$. The left case is generated by always drawing a new sample from the most under-filled class, the middle case with a constant class probability $p_0 = 0.425$ and $p_1 = 0.575$, and the right case by our final algorithm that allows removal of samples. We also measure # of trials as well as # of accepted, rejected, and killed samples (averaged over 10 runs). Notice that all 3 cases have the same final number of samples 10366 (= 11389.7 accepted - 1023.7 killed for the right case).

meet together at the end). However, there are two problems for this method. First, it is very tricky to come up with the right class probabilities; we estimated the numbers for the middle case in Figure 3 by exhaustively trying a vast number of possible combinations. Second, due to the stochastic nature of dart throwing, even with perfectly tuned probabilities, there is no guarantee that a different run with the same parameters would reach the same end condition, i.e. the two curves might still not meet in the end.

To resolve these issues, we allow the removal of existing samples \mathbf{n}_s that are in conflict with a new trial sample s if (1) it is impossible to add another sample to class c_s (this can be figured out by tracking the still available spaces [Dunbar and Humphreys 2006] or by using a simple timeout mechanism), (2) each $s' \in \mathbf{n}_s$ (i.e. these in conflict with s) belongs to a class $c_{s'}$ with a smaller r than the class c_s for s and (3) each $c_{s'}$ is at least as filled as c_s . See `Removable()` in Program 1. Intuitively, this means that we only remove samples from classes that are easier to sample from (i.e. having a smaller r value) and are already as filled as the current class.

```
function bool Removable( $\mathbf{n}_s, s, \mathbf{r}$ )
    foreach  $s' \in \mathbf{n}_s$ 
        if  $\mathbf{r}(c_{s'}, c_{s'}) \geq \mathbf{r}(c_s, c_s)$  or  $\text{FillRate}(c_{s'}) < \text{FillRate}(c_s)$ 
            return false
    return true
```

Program 1: Can we remove the set of conflict samples \mathbf{n}_s for s ?

Although it may sound unusual to allow removal of existing samples, we have found this essential to maintain an equal fill-rate across all classes at all times. As shown in the right case of Figure 3, the two classes maintain consistent fill-rate throughout the sampling process. Furthermore, even though killing samples may in theory increase the computation, in practice we have found that the number of killed samples usually far below the number of accepted and rejected samples. In fact, as shown in Figure 3, the efficiency brought by the sample removal may actually reduce the total number of trials, making the process even more efficient.

2.3 r-matrix construction

As discussed above, we fill the diagonal entries $\mathbf{r}(i, i)$ of \mathbf{r} as r_i , the user specified intra-class minimum distance. But how should we compute the off-diagonal entries of \mathbf{r} ? If we fill the off-diagonal entries with 0, our algorithm will reduce to decoupled single-class sampling (i.e. the top row in Figure 2). On the other hand, if we treat the samples as geometric disks and define the off-diagonal en-

```
function  $\mathbf{r} \leftarrow \text{BuildRMatrix}(\{r_i\}_{i=0:c-1})$ 
    //  $\{r_i\}$ : user specified per-class values
    //  $c$ : number of classes
    for  $i = 0$  to  $c-1$ 
         $\mathbf{r}(i, i) \leftarrow r_i$  // initialize diagonal entries
    end
    sort the  $c$  classes into priority groups  $\{\mathbf{P}_k\}_{k=0:p-1}$  with decreasing  $r_i$ 
    // classes in the same priority group have identical  $r$  values
     $C \leftarrow \emptyset$  // the set of classes already processed
     $D \leftarrow 0$  // the density of the classes already processed
    for  $k = 0$  to  $p-1$ 
         $C \leftarrow C \cup \mathbf{P}_k$ 
        foreach class  $i \in \mathbf{P}_k$ 
             $D \leftarrow D + \frac{1}{r_i^n}$  //  $n$  is the dimensionality of the sample space
        end
        foreach class  $i \in \mathbf{P}_k$ 
            foreach class  $j \in C$ 
                if  $i \neq j$ 
                     $\mathbf{r}(i, j) \leftarrow \mathbf{r}(j, i) \leftarrow \frac{1}{\sqrt[n]{D}}$  //  $\mathbf{r}$  is symmetric
                end
            end
        end
    end
    return  $\mathbf{r}$ 
```

Program 2: \mathbf{r} -matrix construction for uniform sampling.

tries $\mathbf{r}(k, j)$ as $\frac{r_k + r_j}{2}$, we will get results as in the middle row of Figure 2, where the individual classes can be highly non-uniform caused by samples in other classes “getting in the way”.

Our algorithm for computing \mathbf{r} is shown in Program 2. To understand how it works, let’s start with two classes ($c = 2$) only. Since each class i will have expected sample density proportional to $\frac{1}{r_i^n}$ in a n -dimensional sample space, the off-diagonal entries r_ϕ of \mathbf{r} should be computed via the following formula so that the total set has the expected density $\sum_{i=0}^{c-1} \frac{1}{r_i^n}$:

$$\frac{1}{r_\phi^n} = \sum_{i=0}^{c-1} \frac{1}{r_i^n} \quad (2)$$

The bottom row in Figure 2 is produced by \mathbf{r} constructed in this fashion. It can be seen, both experimentally and intuitively, that an r_ϕ value deviating from the one computed via Equation 2 will produce worse results, i.e. a smaller value will produce a less uniform total set as in the top row of Figure 2, whereas a larger value will produce less uniform individual classes as in the middle row of Figure 2. The method described above could also be applied to compute a uniform off-diagonal \mathbf{r} matrix entry value for $c > 2$ classes if they share an identical r value.

However, for $c > 2$ classes with different r values, computing a uniform off-diagonal entry value via Equation 2 will produce sub-optimal results. Recall that a Poisson disk sample set possesses a blue noise power spectrum, with an inner ring radius $\frac{1}{r}$ within which the power spectrum has very low energy. However, in multi-class blue noise sampling, the power spectrum of a class c_j with parameter r_j could interfere with the power spectrum of another class c_i with $r_j > r_i$, as the noise/energy outside frequency $\frac{1}{r_j}$ of class c_j would fall within the inner ring $\frac{1}{r_i}$ of class c_i . Thus, to minimize the pollution inside its inner ring $\frac{1}{r_i}$, each class c_i would need to ensure that the union of all classes $\{c_j\}$ with $r_j > r_i$ has as uniform a joint distribution as possible.

We achieve this goal by the algorithm described in Program 2. Below is an intuitive explanation. We begin by assigning the diagonal entries of \mathbf{r} from the user specified parameters $\{r_i\}$. To compute the off-diagonal entries, we first sort the classes by their r values in a decreasing order. We then add them in that order to the set of

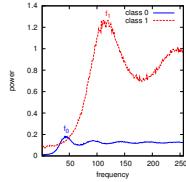


Figure 4: Inter-class spectrum interference. Here we show the radial mean plots for 2 classes with different r values. Each class has its energy peak around frequency $f = \frac{1}{r}$. Note that the peak energy of c_0 falls into the inner ring of c_1 , since $f_0 < f_1$ ($r_0 > r_1$).

already considered classes C , while simultaneously computing the off-diagonal entries that involve the newly added classes. The computation is performed to ensure that at any moment the set C is as uniform as possible, according to the merit of Equation 2.

Discussion An r -matrix built by the aforementioned method can exhibit discontinuous changes in the off-diagonal entries when the r values of a group of classes change from being identical to slightly different. However, we have not found this to impact distribution quality. For real application scenarios, we believe it will be more common to use classes with either identical or sufficiently different r values (to avoid this issue). In addition, the user can choose to group the classes differently from the default behavior in Program 2.

2.4 Adaptive sampling

So far we have described multi-class dart throwing only for uniform sampling. Here we describe how to extend it for adaptive sampling. The main difference between uniform and adaptive sampling is that the user-specified constants $\{r_i\}_{i=0:c-1}$ for the former could be general spatially-varying functions $\{r_i(\cdot)\}_{i=0:c-1}$ for the latter. This requires us to make corresponding spatially-varying extensions for the r -matrix, the conflict check metric, and the criterion in determining if a sample s' is removable relative to s :

- For building r -matrix, we simply apply the algorithm in Program 2 for every sample location s , i.e. $\mathbf{r}(s) = \text{BuildRMatrix}(\{r_i(s)\}_{i=0:c-1})$.
- For conflict check, we use $\hat{\mathbf{r}}(s, s') = \frac{\mathbf{r}(s, c_s, c_{s'}) + \mathbf{r}(s', c_{s'}, c_s)}{2}$ instead of $\mathbf{r}(c_s, c_{s'})$. This is analogous to the use of $\frac{r(s)+r(s')}{2}$ instead of r for single-class adaptive sampling.
- For `Removable()`, we use the sample location s in addition to its class number c_s in Program 1; see Program 3.

```
function bool Removable( $\mathbf{n}_s, s, \mathbf{r}(\cdot)$ )
  foreach  $s' \in \mathbf{n}_s$ 
    if  $\mathbf{r}(s', c_{s'}, c_{s'}) \geq \mathbf{r}(s, c_s, c_s)$  or  $\text{FillRate}(c_{s'}) < \text{FillRate}(c_s)$ 
      return false
  return true
```

Program 3: `Removable()` for adaptive sampling. The colored portions highlight differences from the uniform sampling algorithm in Program 1.

3 Multi-Class Soft Disk Sampling

In Section 2 we extend single-class hard disk sampling for multiple classes of samples. Here, we extend the method further by placing a soft disk centered on each sample. These soft disks behave like energy blobs with local support. They do not have hard boundaries and thus can intersect each other, but the amount of overlap is subject to an energy penalty which, when minimized, produces a uniform distribution. The main advantage of our soft disk sampling is that it serves as a good complement to hard disk sampling, offering explicit control for sample count ($\{N_i\}_{i=0:c-1}$ for c classes) while producing distributions with more spatial uniformity. Even though relaxation [Lloyd 1982] can typically achieve these benefits, we have found that it might not produce good results for the multi-class setting.

Below, we define the notion of a soft disk and an energy function measuring distribution uniformity, followed by a soft dart throwing method that minimizes this energy for blue noise sampling.

3.1 Uniformity measurement

We quantify sample *uniformity* via the following formula:

$$\begin{aligned} \mathbf{E}(s) &= \sum_{s' \in S} \omega(c_s, c_{s'}) \phi_{s', \sigma(s, s')}(s) \\ \phi_{s', \sigma(s, s')}(s) &= e^{-\frac{(s-s')^2}{\sigma(s, s')^2}} \end{aligned} \quad (3)$$

where s is the query sample for energy value, s' any sample in the sample set S , $\omega(c_s, c_{s'})$ a user specifiable scalar weight factor for class combination $(c_s, c_{s'})$, $\phi_{s', \sigma}$ a Gaussian blob with center s' and width σ that depends on the sample pair (s, s') . Intuitively, $\mathbf{E}(\cdot)$ tends to be smaller for an S with a more uniform sample distribution. Below we provide more details about the parameters:

- This parameter $\omega(c_s, c_{s'})$ allows the user to specify different importance to different class combinations. We have found it adequate to simply set $\omega = 1$, treating all classes equally.
- The width σ of a blob $\phi_{s', \sigma(s, s')}(s)$ depends not only on its center s' but also the query sample s . This follows naturally from our multi-class hard disk sampling algorithm presented in Section 2, as the desired spacing between two samples s' and s depends on not only their respective class ids but also their locations for adaptive sampling. Intuitively, $\sigma(s, s')$ should be proportional to $\hat{\mathbf{r}}(s, s')$ in Section 2.4, so that the blob $\phi_{s', \sigma(s, s')}(s)$ properly measures the energy according to the desired distance between s and s' . In our experiments we have found that $\sigma(s, s') = 0.25 \times \hat{\mathbf{r}}(s, s')$ works well.
- To evaluate $\hat{\mathbf{r}}$ we will need to know the sample spacing parameters $\{r_i\}$. These are not given explicitly in soft disk sampling, but can be estimated from the specified sample counts $\{N_i\}$ by setting $r_i = r_{i, \max}$, the average inter-sample distance computed from the maximum packing of N_i samples.

3.2 Soft dart throwing

One possible method to generate samples minimizing Equation 3 is to extend the multi-class dart throwing algorithm in Section 2 for soft disk samples. Our soft dart throwing algorithm is similar to its hard counterpart, with the major difference being that instead of rejecting a new trial sample s' when it is too close to any existing samples, we always accept s' . This feature allows the user to exactly control the final number of samples across all classes. To help ensure the sample uniformity, we perform multiple attempts of s' and pick the one with minimum energy $\mathbf{E}(s')$ among all trials. (A similar idea is used in best candidate dart throwing [Mitchell 1991].) During the initial phase of the algorithm when the domain Ω is sparsely populated, it might be wasteful to perform many trials. To speed this up, the user can optionally specify a threshold energy \mathbf{E}_t to allow early termination of trials when a trial sample s' with $\mathbf{E}(s') < \mathbf{E}_t$ is found. We have found that $\mathbf{E}_t = 0.01$ works well. Note that Equation 3 measures only spatial uniformity and could favor a regular distribution. Our soft dart throwing method, due to its stochastic nature, avoids such potential regularity.

4 Analysis

We use the methods in [Lagae and Dutré 2008] to analyze the spatial and spectrum properties of sample distributions. For spatial uniformity, we utilize the relative radius $\rho = \frac{r}{r_{\max}}$, where r is the minimum spacing between any pair of samples and r_{\max} is the av-

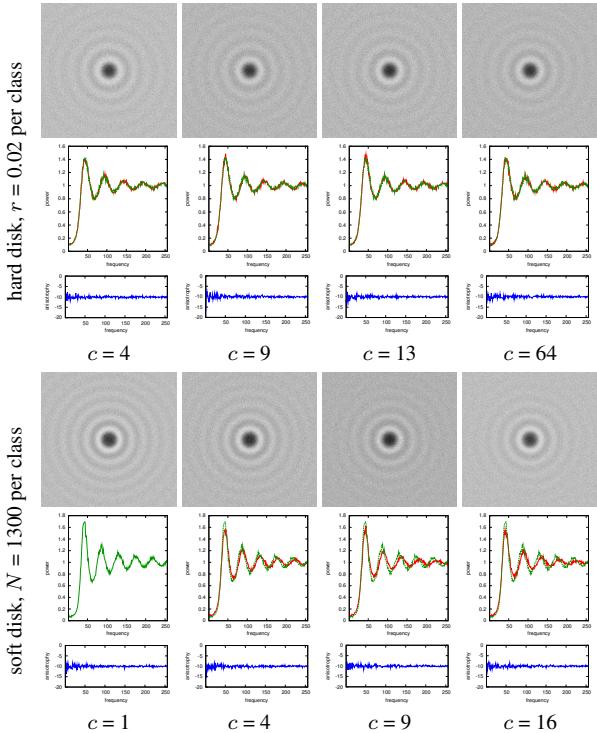


Figure 5: Spectrum results for different number of classes. From top to bottom (within each group): power spectrum, radial mean power, and radial variance/anisotropy. Each column is produced with a different number of classes c as indicated. For easy comparison, we overlay the ground truth mean curve ($c = 1$) in green color with all other cases ($c > 1$).

verage inter-sample distance computed from the maximum packing of a given number of samples. For spectrum analysis, we compute the Fourier power spectrum and measure the radial mean and variance/anisotropy, all averaged over 10 runs.

Single-class soft dart throwing Under the single-class setting, soft dart throwing produces results exhibiting good spatial uniformity with $\rho = 0.75$. The power spectrum analysis also confirms the quality, as shown by the $c = 1$ case in Figure 5. Due to its stochastic and non-iterative nature, soft dart throwing does not tend to settle down into hexagonal shaped local minimum as in traditional Lloyd relaxation [Lloyd 1982] and thus could be used as an alternative for applications that require an exact sample count.

Number of classes We start our analysis for multi-class sampling with the simplest case where all the classes have the same parameters, i.e. r for hard disk sampling and N for soft disk sampling. As shown Figure 5, the multi-class statistics remain similar to the single-class setting across a variety of c numbers.

As recommended by [Lagae and Dutré 2008], ρ should be in the range $[0.65, 0.85]$ for single-class blue noise sampling. However, for the multi-class setting, we have found that $\rho \in [0.65, 0.70]$ is achievable but beyond that may require excessive number of trials or iterations. Fortunately, this issue does not seem to worsen progressively with the increasing number of classes; due to our r -matrix construction algorithm, the inter-class r values decrease with the increasing number of classes, thus they tend to cancel each other out in terms of imposing additional constraints. All results shown in the paper have $\rho \geq 0.67$ unless indicated otherwise.

Non-uniformity Next we examine what happens if the classes have different parameters. We start with the simplest case of only

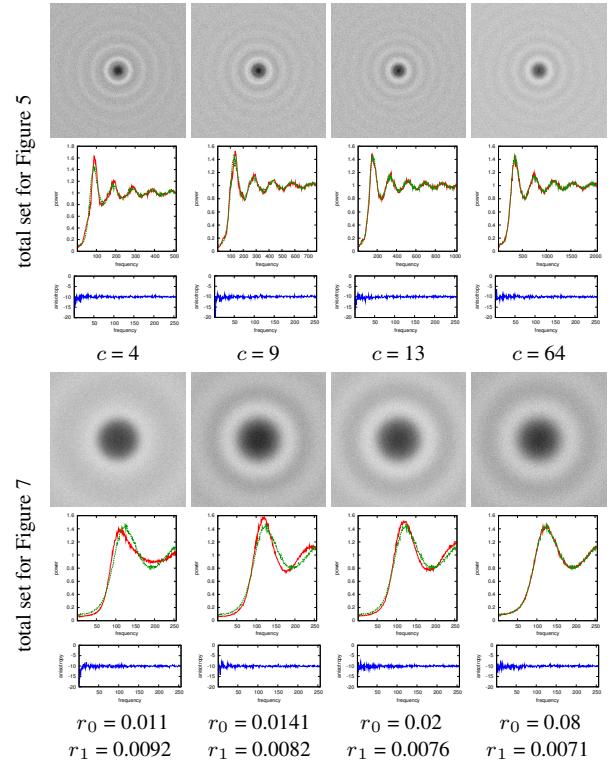


Figure 6: Total sample set distribution. For the radial mean plot, we overlay the ground truth produced by single-class dart throwing as green curves for easy comparison. Shown here are only hard disk results; the soft disk ones have similar quality.

two classes as shown in Figure 7. We produce several sets of 2-class sample sets with different r_0 and r_1 values so that the total is a sample set with the same r value (where $\frac{1}{r^n} = \frac{1}{r_0^n} + \frac{1}{r_1^n}$). We start with similar r_0 and r_1 on the left and with increasing disparity towards the right. Here, we can observe several interesting facts:

- Class 0 remains indistinguishable from single-class sampling. However, class 1 might deviate from the single-class results, as manifested by the small “humps” between their radial mean curves. These humps are caused by the spectrum peaks of class 0, centered at frequency $\frac{1}{r_0}$ as discussed in Figure 4.
- The deviation between class 1 and ground truth is less obvious when r_0 and r_1 are either sufficiently similar or sufficiently dissimilar. For the former, the hump will happen around the existing peak of class 1, making it non-obvious. For the latter, class 0 simply has too few samples to have a major impact on the power spectrum of class 1. However, even for the case with maximum discrepancy ($r_0 = 0.02$ and $r_1 = 0.0076$) we have not found noticeable differences in sampling results.

Class priority As we discussed in Section 2.2, generating all classes together allows us to maintain consistent fill-rates and thus distribution quality among all classes. Generating the classes sequentially does not allow us to do this, and would require us to specify criteria for stopping the generation of one class and starting another one. This might not be easy as it is very hard to predict if an earlier class would over-constrain the generation of a later one. Furthermore, generating the classes sequentially might actually harm the distribution quality; as illustrated in Figure 8, when classes c_0 and c_1 are produced prior to c_2 , c_2 might have little rooms left, resulting in a non-uniform distribution. This effect is particularly pronounced for the soft disk sampling case. However, we wish

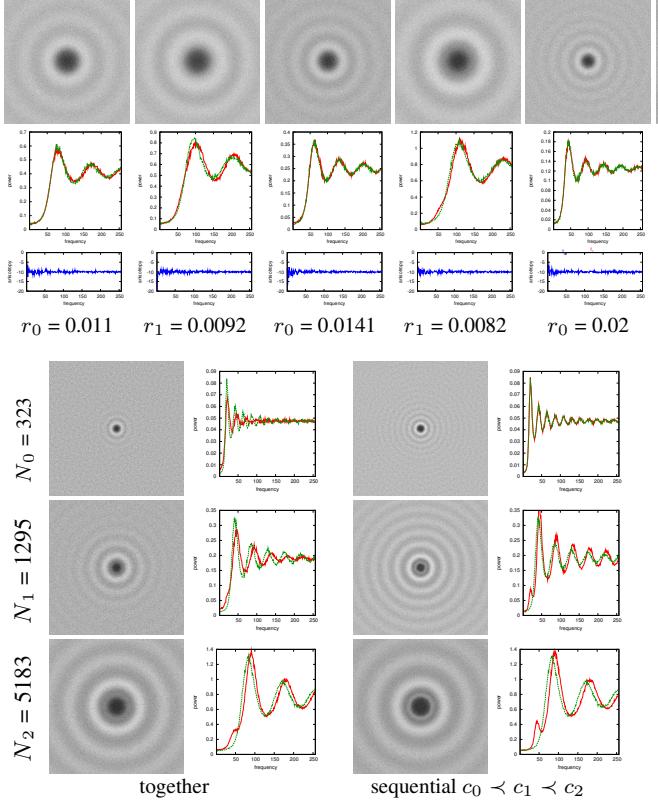


Figure 8: Class priority. Here we generate 3 classes of samples either together (left group) or sequentially (right group). We use soft- rather than hard-disk sampling due to more pronounced effects. In the radial mean plots, the red/green curves correspond to multi/single-class results. Note that generating the classes sequentially would introduce more low frequency noise for the classes produced latter.

to emphasize that both our hard and soft disk sampling algorithms can be used to generate the sample classes sequentially if the user desires so.

Total sample set Figure 6 shows the power spectrums for the total sample distribution from selected sets of results in Figure 5 and Figure 7. As shown, the power spectrums of the total sets stay close to the ground truth blue noise profiles. The deviation is most obvious when there are a small number of classes and/or when the classes have similar r values. When there are a larger number of classes, the probability that a sample has a neighbor in a different class is higher, thus geometrically making the entire sample set more similar to a single-class distribution; see the progression in the top cases. When the classes have dissimilar r values, the class with larger r has fewer samples to impact the overall power spectrum; see the progression in the bottom cases.

Performance Using a simple grid/tree data structure [Bridson 2007; Wei 2008] for storing samples and checking conflicts, our current implementations are able to achieve reasonable performance, as tabulated in Table 1. For soft disk sampling, we usually cut off the Gaussian blobs beyond 3σ , thus localizing all energy updates and evaluations. The performance decreases with an increasing number of classes since the sample placement is more constrained, incurring more computations during the generation process. Since a grid/tree data structure allows us to perform the conflict check in constant time, the total time for generating N samples is χN , where χ is the ratio of the total number of trials over N . We

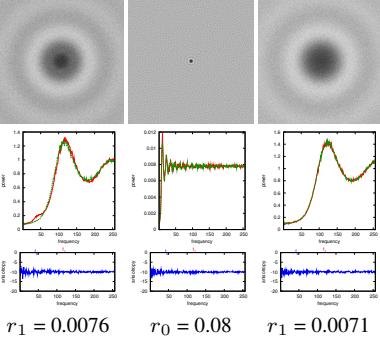


Figure 7: Spectrum results for two classes with different r values. Here we show the hard disk results for easy explanation; the soft disk results have similar but slightly more pronounced effects. Each pair of columns is produced together with different r_0 and r_1 values so that their total is a sample set with $r = \frac{0.01}{\sqrt{2}}$. For the radial mean plots, we overlay the single-class ground truth as green curves for easy comparison.

are not yet able to determine an accurate formula for χ since it depends on not only the number of classes but also certain implementation details, e.g. how accurately the available space is tracked to reduce futile trials. (We measure the timing in Table 1 via uniform random sampling as in classical dart throwing without such empty-space tracking.) We thus believe that χ could be better determined along with more definite future accelerations of our algorithms.

# classes	1	2	3	4	5	6	7	8
hard disk	9.85	3.80	2.74	2.24	1.36	1.15	0.99	0.65
soft disk	0.23	0.17	0.135	0.13	0.13	0.12	0.086	0.084

Table 1: Performance of our algorithms. All performance numbers are in K-samples/second, and measured on a laptop with a 2.50 GHz CPU + 2 GB RAM. The 1-class case serves as a reference for others.

5 Applications

Here, we show several applications for multi-class blue noise sampling. Depending on the particular application needs, either hard-disk or soft-disk sampling may be more suitable. Specifically, the former is better for applications that have strict sample spacing requirement and are flexible with regard to the number of samples, and the latter for applications that prefer more uniform spatial distribution and exact specification for the number of samples. Hard disk sampling is also more natural for continuous sample spaces whereas soft disk sampling suitable for both continuous and discrete settings. In addition, hard dart throwing is usually computationally faster as it is easier to accelerate.

5.1 Object distribution

Uniform object placement is often desirable for both scientific (e.g. biological distribution) and artistic (e.g. procedural texture [Lagae and Dutré 2005]) applications. Such a uniform distribution can be achieved by blue noise sampling [Cohen et al. 2003; Lagae and Dutré 2005; Kopf et al. 2006], but existing methods do not explicitly consider the presence of multiple classes of objects. We can apply our approach for this purpose. An example is shown Figure 1 for placing two classes of objects in either uniform or adaptive distribution. Due to the desire to keep minimum distances between objects, we opt for hard disk sampling for this application. Our method could also be applied to place 3D objects (e.g. flowers [Cohen et al. 2003]) for scene design or 2D motifs for pattern generation (e.g. [Lagae and Dutré 2005]).

5.2 Color stippling

In addition to object placement, blue noise sampling can also be employed for stippling with visually pleasing pointillism effects (see e.g. [Kopf et al. 2006; Kim et al. 2008; Balzer et al. 2009]). However, existing stippling results are mostly black-and-white since traditional blue noise sampling can handle only a single



Figure 9: Color stippling result. Using a color image (a corner from Seurat’s “A Sunday Afternoon on the Island of La Grande Jatte”) as the input importance field, our method produces an adaptive sample set with $\sim 290K$ color dots in 7 classes (red, green, blue, cyan, magenta, yellow, black) over a white background. (Note: this image might not show up well in print; try it on a computer display and vary the viewing distance.)

class of samples. We can apply our algorithms for multi-color stippling by using a color image as the input importance field, treating each color channel as a separate class and producing a multi-class output sample set accordingly. Unlike color halftoning which mainly targets discrete regular sample sets, our method allows samples to be placed anywhere and thus provides more of a free-style pointillism effect. As shown in Figure 9, our method can produce reasonably complex color stippling; the colored dots not only follow the input importance field but also maintain a blue noise distribution.

5.3 Sensor layout

The layout of a color sensor array determines the quality of the sampling results as well as subsequent reconstruction algorithms, such as super-resolution. The most widely used layouts usually deploy the RGB sensor elements in a regular grid (or variations thereof); as pointed out in [Ben Ezra et al. 2007], grid layouts are subject to a variety of sampling and reconstruction issues, and the authors recommended the use of Penrose pixels. However, as pointed out in [Kopf et al. 2006; Lagae and Dutré 2008], a Penrose sample layout, even after quality improvement via jittering [Ostromoukhov et al. 2004], still exhibits visible spectrum bias.

Following an analogous line of thinking, we wonder if it is possible to further improve the quality of Penrose pixel layout for color sensors [Ben Ezra et al. 2007] via our approach, treating the RGB sensors as three classes of samples. The comparison is shown in Figure 10. For [Ben Ezra et al. 2007], we use the randomized 3-coloring algorithm in [McClure 2002] to assign the RGB sensor locations. For our result, we use soft disk sampling to specify the exact number of samples. As shown, our result has no bias in the power spectrum as well as no aliasing in a spatial sampling for the zone-plate pattern, a commonly used stress test for evaluating sampling quality [Kopf et al. 2006; Ostromoukhov 2007; Wei 2008].

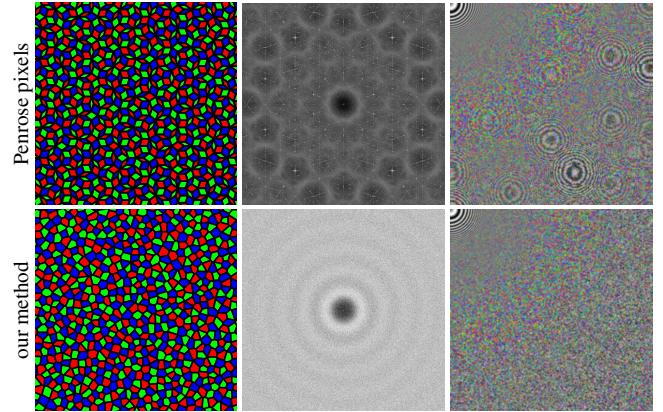


Figure 10: Comparison with Penrose pixels for RGB sensor layout. From left to right: spatial sensor layout, power spectrum, and spatial sampling via the zone-plate pattern. The spectrum results are produced by one of the classes while the spatial sampling via all 3 classes.

5.4 Color filter array design

Penrose pixels [Ben Ezra et al. 2007] and our method would produce better spectrum quality than traditional regular grid sensor layout. However, a regular layout is easier to fabricate, especially for wiring [Ben Ezra et al. 2007]. We have found it possible to maintain the regular layout for the sensor cells, but apply our technique to de-regularize the color filter placement so that the spectrum quality is still improved. This can be achieved by applying our multi-class soft disk sampling algorithm to a pre-determined regular set of samples. Due to the desire to specify the exact number of samples per (color channel) class, this is an application where soft-disk would be more suitable than hard-disk sampling.

As shown in Figure 11, a regular layout (Bayer mosaic in that particular case) causes significant aliasing as expected. One possible solution to reduce aliasing is to place the samples randomly. Even though randomization cannot remove aliasing caused by the underlying regular grid structure, the sampling quality is still improved by de-regularizing the color filter elements. However, this reduction in aliasing is achieved at the expense of more noise. Our soft disk approach, in contrast, reduces aliasing compared to a regular layout while introducing less noise than a random layout. We have also shown result produced by our hard disk approach with a discrete sample domain. To ensure that all sensor elements are utilized, we gradually decrease the r parameters throughout the sampling process similar to [McCool and Fiume 1992]. As shown, even though the hard disk result is better than random placement, it is still worse than the soft disk one. Furthermore, hard disk sampling cannot guarantee an exact number of samples per class.

6 Limitations and Future Work

We have mainly focused on the basic algorithms for multi-class sampling and only lightly touched on the issues of acceleration. Since our algorithms are extensions of dart throwing, we believe they can benefit from a repertoire of previous acceleration techniques, such as [Jones 2006; Dunbar and Humphreys 2006; White et al. 2007; Wei 2008]. Our method is also applicable for constructing multi-class sample tiles [Cohen et al. 2003; Ostromoukhov et al. 2004; Kopf et al. 2006; Lagae and Dutré 2006; Ostromoukhov 2007] as another way to save run-time computation.

Although we have only demonstrated results in 2D, our algorithm is directly applicable to higher dimensional spaces [Bridson 2007; Wei 2008] for scenarios like 3D object distribution. It would also be interesting to extend our approach to sample non-Euclidean do-

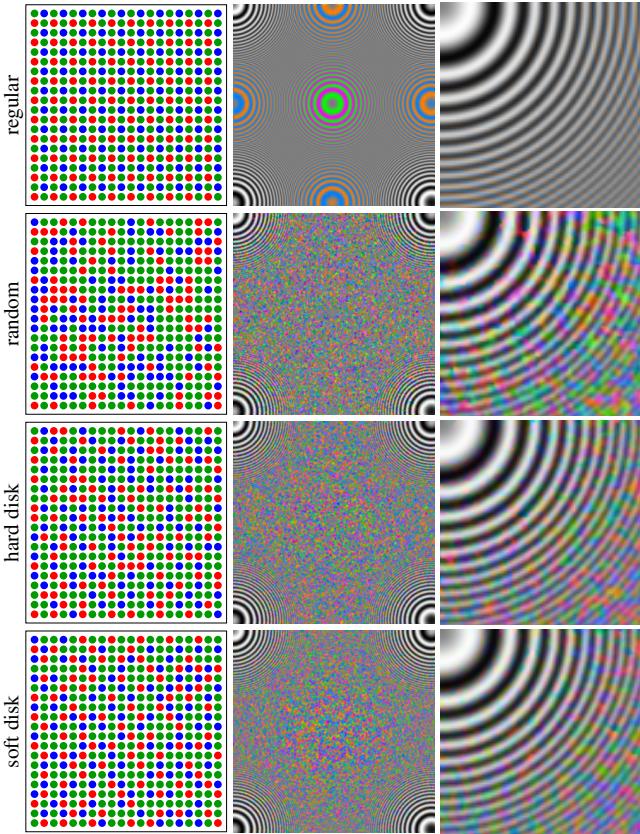


Figure 11: Color filter array design. From left to right: spatial filter array layout, zoneplate sampling, and zoom-in of the low frequency corner. The bottom 3 methods can remove aliasing caused by regular color layout, but not aliasing by the underlying sensor grid (i.e. near the zoneplate corners). Compared to other methods, our soft disk sampling produces more uniform spatial layouts, translating to less noisy sampling results (right column).

mains such as manifold surfaces [Turk 1992; Fu and Zhou 2008; Cline et al. 2009] for rendering and texturing applications.

Acknowledgements Yin Li inspired my pursuit of this project by asking if it is possible to extend [Wei 2008] to place multiple classes of objects. Kun Zhou, Xin Tong, Eric Stollnitz, Brandon Lloyd, Bill Baxter, and the anonymous reviewers provided valuable suggestions. The OPAL Group fixed a SIS bug that once made my submission disappear. I would like to thank Naga Govindaraju, John Manferdelli, Dan Reed, and other incubation team members for their support. Flower images courtesy of <http://www.gif-favicon.com/>. Grass texture courtesy of <http://store.got3d.com/>.

References

- ALLIEZ, P., DE VERDIÈRE, É. C., DEVILLERS, O., AND ISENBURG, M. 2003. Isotropic surface remeshing. In *Shape Modeling International*, 49–58.
- BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: A variant of Lloyd’s method. In *SIGGRAPH ’09*, 86:1–8.
- BAQAI, F., LEE, J.-H., AGAR, A., AND ALLEBACH, J. 2005. Digital color halftoning. *Signal Processing Magazine, IEEE* 22, 1 (Jan.), 87–96.
- BEN EZRA, M., LIN, Z., AND WILBURN, B. 2007. Penrose pixels super-resolution in the detector layout domain. In *ICCV ’07*, 1–8.
- BRIDSON, R. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *SIGGRAPH ’07 Sketches & Applications*.
- CLINE, D., JESCHKE, S., RAZDAN, A., WHITE, K., AND WONKA, P. 2009. Dart throwing on surfaces. In *EGSR ’09*, 1217–1226.
- COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *SIGGRAPH ’03*, 287–294.
- COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph.* 5, 1, 51–72.
- DUNBAR, D., AND HUMPHREYS, G. 2006. A spatial data structure for fast Poisson-disk sample generation. In *SIGGRAPH ’06*, 503–508.
- FU, Y., AND ZHOU, B. 2008. Direct sampling on surfaces for high quality remeshing. In *SPM ’08*, 115–124.
- JONES, T. R. 2006. Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools* 11, 2, 27–36.
- 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. In *SIGGRAPH ’06*, 509–518.
- LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph.* 24, 4, 1442–1461.
- 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.
- LLOYD, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2, 129–137.
- MCLURE, M. 2002. A stochastic cellular automaton for three-coloring penrose tiles. *Computers & Graphics* 26, 3, 519–524.
- MCCOOL, M., AND FIUME, E. 1992. Hierarchical Poisson disk sampling distributions. In *Graphics Interface ’92*, 94–105.
- MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *SIGGRAPH ’87*, 65–72.
- MITCHELL, D. P. 1991. Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph.* 25, 4, 157–164.
- OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH ’04*, 488–495.
- OSTROMOUKHOV, V. 2007. Sampling with polyominoes. In *SIGGRAPH ’07*, 78:1–6.
- PANG, W.-M., QU, Y., WONG, T.-T., COHEN-OR, D., AND HENG, P.-A. 2008. Structure-aware halftoning. In *SIGGRAPH ’08*, 89:1–8.
- TURK, G. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH ’92*, 55–64.
- WANG, M., AND PARKER, K. 1999. Properties of combined blue noise patterns. *ICIP* 4, 328–332.
- WEI, L.-Y. 2008. Parallel Poisson disk sampling. In *SIGGRAPH ’08*, 20:1–9.
- WHITE, K., CLINE, D., AND EGBERT, P. 2007. Poisson disk point sets by hierarchical dart throwing. In *Symposium on Interactive Ray Tracing*, 129–132.
- YELLOTT, J. I. J. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science* 221, 382–385.