Multi-Class Poisson Disk Sampling

Li-Yi Wei

Abstract

Sampling has been a core process for a variety of graphics applications including rendering, imaging, and geometry processing. Among the plethora of sampling patterns, Poisson disk distribution remains one of the most popular thanks to its spatial uniformity and blue noise spectrum. However, research so far has been mainly focused on Poisson disk distribution with one class of samples. This could be insufficient for common natural as well as man-made phenomenon requiring multiple classes of samples, such as object placement, imaging sensors, and stippling patterns. We extend Poisson disk sampling to multiple classes of samples where each individual class as well as their union exhibit Poisson disk properties. We propose algorithms to generate such multi-class Poisson disk samples, study their statistical characteristics, and demonstrate applications in object placement, sensor layout, and color stippling.

Keywords: Poisson disk, blue noise, sampling, multi-class

1 Introduction

Sampling is important for a variety of graphics applications, include rendering, imaging, and geometry processing. Although different applications may favor different sampling patterns, Poisson disk sampling [Cook 1986] remains one of the most popular and widely adopted. Inspired by the distribution of primate retina cells [Yellott 1983], a Poisson disk distribution contains samples that are randomly located but remain at least a minimum distance r away from each other. The resulting sample set has a blue noise power spectrum, replacing low frequency aliasing with high frequency noise, a visually less annoying artifact.

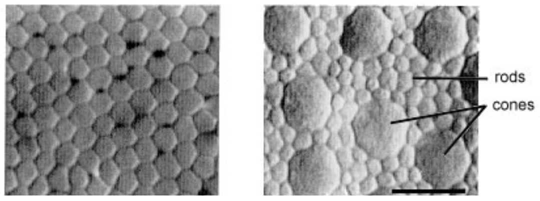


Figure 1: Retina mosaic. The fovea region is packed with cone cells (single class, left case). Further away from the fovea the cones become larger and sparser, with in-between space filled by rods (multiple class, right case).

Despite the nice properties of Poisson disk sampling, existing research has so far concerned about a single class of samples, i.e. all the samples are utilized together in the applications [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]. However, a variety of natural or man-made phenomenon may contain multiple classes of samples, such as the distribution of cone and rod cells in human retinas (Figure 1), the placement of multiple categories of objects, and the usage of multiple-colored dots for dithering or stippling. In these situations, it is often desirable to have each individual class of samples as well as their union to exhibit Poisson disk distribution. Unfortunately, previous methods for single-class Poisson disk sampling are not suitable for generating multiple classes of samples. Let us illustrate this issue via a simple 2-class example as in Figure 3. If we generate the individual classes separately via single-class Poisson disk sampling, their union might be highly non-uniform (top row in Figure 3). This can be undesirable for a variety of reasons depending on the specific applications, e.g. non-uniform distribution for object placement or under-packing for sensor layout. On the other hand, if we generate the entire collection via single-class Poisson disk sampling, the individual sets might be highly non-uniform (middle row in Figure 3). This can produce sub-optimal sampling quality for each individual class of samples.

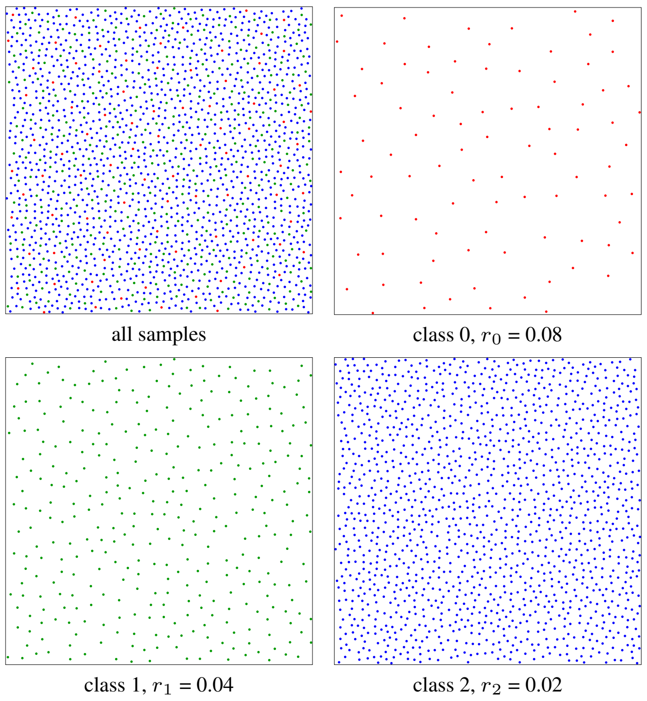


Figure 2: Multi-class Poisson disk sampling. Here we generate 3 classes of samples and visualize them in different colors. Each class could have its own density (controlled by class specific parameter r_i). Note that each individual class as well as their union exhibit Poisson disk sample distribution.

We present multi-class Poisson disk sampling, a technique to generate multiple classes of samples so that each individual class as well as their unions exhibit Poisson disk properties (Figure 2 & 3). Our technique is inspired by dart throwing [Cook 1986], but we propose algorithmic innovations so that it can generate multiple classes of Poisson disk samples. Our algorithm also provides a mechanism to control the relative number of samples across different classes, an issue non-existing in traditional single class sampling. Even though achieving these goals is challenging, we have strived to come up with an algorithm that is simple and elegant (as summarized in Program 1). As added benefits, our method also works in arbitrary dimensions [Bridson 2007; Wei 2008] and can be extended for adaptive sampling [Ostromoukhov et al. 2004; Kopf et al. 2006; Ostromoukhov 2007; Wei 2008].

We demonstrate a variety of applications of our technique, include object distribution [Cohen et al. 2003; Lagae and Dutré 2005; Kopf et al. 2006], sensor layout [Ben Ezra et al. 2007], and color stippling [Ostromoukhov and Hersch 1999; Pang et al. 2008].

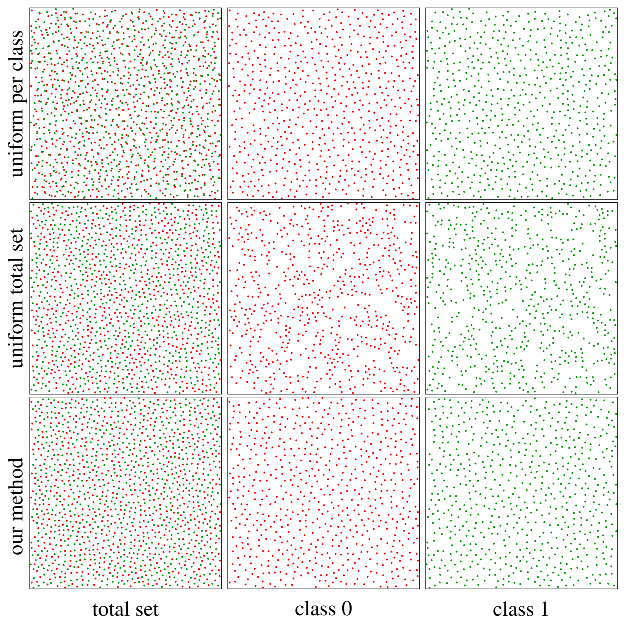


Figure 3: Comparisons between single- and multi-class Poisson disk sampling. The top case is produced by applying single-class Poisson disk sampling to individual classes, but the total set is highly non-uniform. The middle case is produced by applying single-class Poisson disk sampling to the total set, but the individual classes are highly non-uniform. Our approach produces samples that exhibit Poisson disk distribution for each class as well as the total set. Each class contains ∼650 samples generated with r = 0.02.

2 Uniform Sampling

The input to our algorithm consists of the sampling domain Ω as well as a set of user-specified intra-class distances $\{r_i\}_{i=0:c-1}$ for the c classes. Our goal is to produce c classes of samples so that each class has a similar statistical distribution to a single-class Poisson disk sample set with parameter r_i, whilst the total sample set exhibits Poisson disk distribution as well. In addition, the algorithm should be able to control the relative number of samples across different classes throughout the entire process. We summarize our algorithm in Program 1.

Our basic algorithm is an extension from traditional (single-class) dart throwing [Cook 1986]. In [Cook 1986], a trial sample is drawn uniformly from the entire domain. If the sample is not within a user-specified distance r from any other existing samples, it is accepted and added to the existing sample set. Otherwise, it is rejected. This process is repeated until a target number of samples are produced or a maximum number of trials are reached. Our algorithm also follows a similar approach, with the major difference being that we have to deal with multiple classes of samples. Specifically, for each trial sample, we need to decide: (1) which class to sample from, and (2) how to determine whether to accept or reject the sample. Details are as follows.

2.1 Sample class

For multi-class sampling, we have to decide which class to sample from 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 the under-filled-ness via FillRate (Program 1), 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, the target number of samples of class i, N_i, can be computed as follows:

```
function $S \leftarrow$ MultiClassDartThrowing($\Omega, \{r_i\}_{i=0:c-1}$)
    // $\Omega$: sampling domain
    // $\{r_i\}$: user specified parameters for intra-class sample spacing
    // $c$: number of classes
    // $\mathbf{r}$: $c \times c$ matrix controlling inter-class sample spacing
    $\mathbf{r} \leftarrow$ BuildRMatrix($\{r_i\}_{i=0:c-1}$) // see Program 2
    $S \leftarrow \emptyset$ // final set of samples
    while not enough trials attempted and not enough samples in $S$
        $s \leftarrow$ new sample uniform-random drawn from $\Omega$
        $c_s \leftarrow \arg\min_c$ FillRate($c$) // choose the most under-filled class
        if $\forall s' \in S \ \ |s - s'| \geq \mathbf{r}(c_s, c_{s'})$
            add s to $S$
        else if impossible to add another sample to $c_s$
            // try to remove the set of conflicting samples $N_s$
            $N_s \leftarrow \bigcup s' \in S$ where $|s - s'| < \mathbf{r}(c_s, c_{s'})$
            if Removable($N_s, s, \mathbf{r}$)
                remove $N_s$ from $S$
                add s to $S$
            end
        end
    end
    return $S$

function float FillRate($c$)
    return $\frac{\text{\# of existing samples} \in c}{\text{target \# of samples for } c}$ // see Equation 1

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

Program 1: Multi-class dart throwing for uniform sampling.

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

where N is the total number of target samples, n the sample space dimension, and $\{r_i\}$ the specified per-class minimum distances.

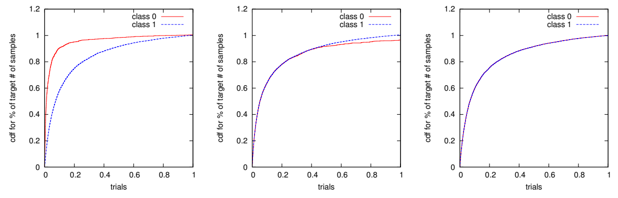
2.2 Conflict metric

In traditional single-class Poisson disk sampling, we can easily decide whether to accept or reject a trial sample based on whether it is at least distance r away from all existing samples. For multi-class sampling, the user would supply a set of intra-class distances $\{r_i\}_{i=0:c-1}$ for the c classes. This set of parameters are analogous to the r parameter in traditional single-class sampling, controlling the density of the resulting sample set. However, specifying the intra-class distances is not enough, as we need more information to decide whether two samples from different classes are too close to each other. To achieve this goal, we use a (symmetric) matrix $\mathbf{r}$ where $\mathbf{r}(k, j)$ specifies the minimum distance between samples from class k and j and $\mathbf{r}(i, i) = r_i$ for the diagnonal (intra-class) entries. Specifically, two samples s and s' may co-exist if their Euclidean distance $|s - s'| \geq \mathbf{r}(c_s, c_{s'})$ where $c_s/c_{s'}$ indicate the class for sample s/s'. We describe how to construct $\mathbf{r}$ in Section 2.4.

2.3 Sample removal

It is usually desirable to maintain a consistent fill rate among different classes throughout the sampling process, as this allows us to terminate the process at any time. However, drawing the next trial sample from the most under-filled class (Section 2.1) alone is not enough to achieve this goal. (See **discussion** below for details).

To overcome this issue, we allow the removal of existing samples 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 in the merit of [Dunbar and Humphreys 2006] or a simple timeout mechanism), (2) each $s' \in N_s$ 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 we sample from. (It can be easily shown that this sample removal process will never introduce infinite loops as it treats the classes hierarchically according to the r values.) 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.



240310.5 trials	247810.7 trials	157388.3 trials
0 killed	0 killed	1023.7 killed
229944.5 rejected	237444.7 rejected	145998.6 rejected
10366 accepted	10366 accepted	11389.7 accepted

Figure 4: 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 with a constant class probability $p_0 = 0.425$ and $p_1 = 0.575$, the middle case by always drawing a new sample from the most under-filled class, 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).

Discussion Here, we offer more detailed discussions on why we conclude that it is unavoidable to remove samples. We use a simple 2-class example as in Figure 4. In our first attempt (left case), we tried to control the relative probability to sample from each class, without considering their relative fill-rate and without removing samples. When the relative class probability is perfectly chosen, it will be possible to achieve the desired fill rates at the end of the sampling process (notice the two curves meet together at the end). However, there are two problems here. First, it is very tricky to come up with the right class probabilities (we estimated the numbers for Figure 4 left case by exhaustively trying a vast number of possible combinations until the two curves meet in the end). Second, even though the two classes achieve the same fill-rate at the end, it is not so during the middle of the process; this makes it hard to terminate the computation at any time.

In our second attempt, as shown in the middle case of Figure 4, we try to draw a sample from the most under-filled class (as described in Section 2.1) but still without removing samples. (To avoid deadlock, we accepts a trial sample s if it fails to be accepted for the most under-filled class but succeeds for another one.) As shown, even though the two classes maintain consistent fill-rates throughout the early stage of the process, eventually it becomes difficult for the class with a larger r value to compete with the other one.

In the final version of our algorithm, we allow samples to be removed. As shown in the right case of Figure 4, 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 is usually far below the number of accepted and (especially) rejected samples. In fact, as shown in Figure 4, the efficiency brought by the sample removal may actually reduce the total number of trials, making the process even more efficient.

2.4 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 3). On the other hand, if we treat the samples as geometric disks and define the off-diagonal entries $\mathbf{r}(k,j)$ as $\frac{r_k+r_j}{2}$, we will get results as in the middle row of Figure 3, where the individual classes can be highly non-uniform caused by samples in other classes "getting in the way".

```
function r ← 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
        $C \leftarrow C \bigcup \mathbf{P}_k$
        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 $i \neq j$
                    $\mathbf{r}(i,j) \leftarrow \mathbf{r}(j,i) \leftarrow \frac{1}{\sqrt[n]{D}}$ // r is symmetric
            end
        end
    end
    return r
```

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

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_o of $\mathbf{r}$ should be computed via the following formula so that the total set has the expected density $\sum_{i=0}^{c-1} \frac{1}{r_i^n}$:

$$\frac{1}{{r_o}^n} = \sum_{i=0}^{c-1} \frac{1}{r_i^n} \tag{2}$$

The bottom row in Figure 3 is produced by $\mathbf{r}$ constructed in this fashion. It can be seen, both experimentally and intuitively, that a r_o 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 3, whereas a larger value will produce less uniform individual classes as in the middle row of Figure 3.

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. The details are discussed in Section 3 (Figure 10), but here is a high level, intuitive explanation (Figure 5). 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 have very low energy. This is a main desirable feature for blue noise, pushing low frequency aliasing towards high frequency noise. However, in multi-class Poisson disk sampling, the power spectrum of a class c_i with parameter r_i could be intefered by any other class c_j 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.

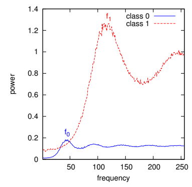


Figure 5: Inter-class spectrum inteference. Here we show the radial mean plots for 2 classes with different r values. Each class has its energy peak around frequency $f = \frac{1}{r}$. Note that the peak energy of c_0 falls into the inner ring of c_1, since $f_0 < f_1$ ($r_0 > r_1$).

We achieve this goal by the algorithm described in Program 2. 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 into priority groups $\{\mathbf{P}_k\}_{k=0:p-1}$ with decreasing r values, where each group $\mathbf{P}_k$ contains classes with identical r. We then visit these priority groups one by one as follows. We add each group $\mathbf{P}_k$ to the current aggregate group C, as well as the corresponding r values to the current aggregate accumulated density D via Equation 2. We then assign each off-diagonal entry $\mathbf{r}(i, j)$ with $\frac{1}{\sqrt[n]{D}}$, where $i \neq j$ with $i \in \mathbf{P}_k$ and $j \in C$ (or vice versa); this essentially applies Equation 2 to the off-diagonal entries in a sequential fashion so that each accumulated group C so far would be as uniform as possible as dedicated by our algorithm.

Discussion We are fortunate that a class c_i with r_i does not need to worry about any other class c_j with $r_j \leq r_i$, as c_j would impact c_i power spectrum outside its inner ring $\frac{1}{r_i}$. This allows us to sort all classes into a hierarchy according to their decreasing r value and construct $\mathbf{r}$-matrix accordingly (Program 2). Otherwise, we will have an impossible task of building an $\mathbf{r}$-matrix so that an arbitrary union of different classes has to be uniform.

Also note that even though we build $\mathbf{r}$-matrix in a priority order according to the r values of the classes, we generate samples across all classes together instead of in a prioritized order (as described above). See Section 3 (Figure 10) for a more detailed analysis.

3 Analysis

Here, we analyze the spectrum properties [Lagae and Dutré 2008] of the sample sets produced by our method. We also provide additional justifications for the design of our algorithm.

Number of classes We start our analysis with the simplest case where all the classes have the same parameters, including r as well as the target number of samples. Since samples in each class have to "get around" samples in other classes, one might concern that each individual class of samples produced by our approach might be less uniform than the one produced with traditional dart throwing [Cook 1986]. However, empirically we have found that this is not an issue, as long as we stick to Program 2 for computing the $\mathbf{r}$ matrix. As shown in Figure 6, we use a uniform value for the diagonal entries $\{r_i\}$ of the $\mathbf{r}$ matrix, produce sample sets with different number of classes c, and measure the mean and variance of the averaged power spectrum for one of the classes (the others have similar statistics). As shown, the sample statistics remain similar to traditional single-class dart throwing across a variety of c numbers.

Non-uniformity So far we have only analyzed situations where each class has the same r values. Here, we examine what happens if the classes have different r values. We start with the simplest case of only 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 values on the left with increasing disparity towards the right of Figure 7. Here, we can observe several interesting facts:

- Class 0 remains indistinguishable from single-class dart throwing. However, class 1 might deviate from the single-class results, as manifested by the small "humps" between their radial mean curves. These humps are obviously caused by the spectrum peaks of class 0, as they are centered around frequency $\frac{1}{r_0}$. The existence of these humps could be explained by inter-class interference as illustrated in Figure 5.
- The deviation between class 1 and ground truth is less obvious when r_0 and r_1 are either very similar or very dissimilar. When r_0 and r_1 are similar, the hump will happen around the existing peak of class 1, making it non-obvious. When r_0 and r_1 are dissimilar, class 0 simply has too few samples to have a major impact on the power spectrum of class 1. The hump could be visualized as a "traveling wave" across the 5 pairs of cases in Figure 7. The hump becomes most obvious at the center pair (r_0 = 0.02 and r_1 = 0.0076), but even there the deviation from the ground truth is not all that major. Our spatial sampling results also confirm this; see Figure 8.

Optimality Given a set of 2-class samples produced by our method, we would like to know how optimal the distribution of each class is. Specifically, we would like to know if there is a better way to partition the set of all samples into 2 classes with designated densities to reduce the humps in Figure 7. Here we compare our results with two extreme cases: (1) each class is randomly distributed among all samples and (2) each class is uniformly distributed by running a discrete version of the Lloyd relaxation over the total sample sets. (Note that we use a discrete instead of a continuous relaxation. The latter could settle into a regular tiling with highly biased spectrums, as reported in prior studies, e.g. [Lagae and Dutré 2008]. The discrete relaxation has no such issue as the output is constrained to lie on a set of Poisson disk samples.)

As shown in Figure 9, the random case shifts energy towards lower frequencies; this is undesirable since it would make the sampling noisier. The relaxation case does shift some energy away from the low frequency range compared to our results, but on the other hand the uniformity of the samples would cause a more spiky energy profile around frequency $\frac{1}{r_0}$. Even though this might be desirable for class 0, this may not be so for class 1 as a spikier hump is introduced within its inner ring $\frac{1}{r_1}$. This example illustrates one important difference between single- and multi-class sampling; for the former, we just need to optimize for one class but for the latter, an optimally uniform distribution of one class might actually harm another one.

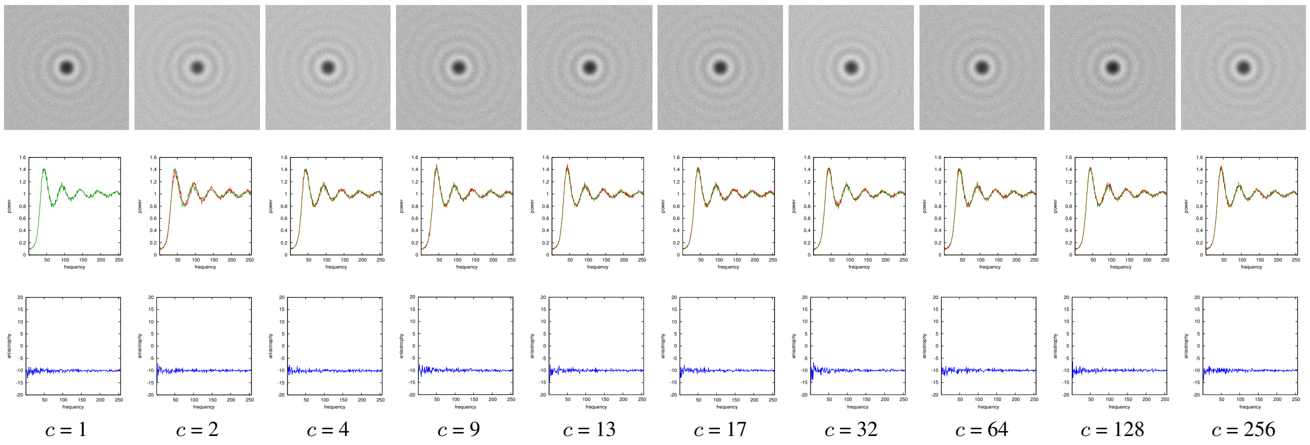


Figure 6: Spectrum results for different number of classes. From top to bottom: power spectrum averaged over 10 runs, radial mean power, and radial variance/anisotropy. Each column is produced with a different number of classes c as indicated. $r = 0.02$, # samples $\simeq$ 1300 per class for all cases. For easy comparison, we overlay the ground truth mean curve ($c = 1$) in green color with all other cases ($c > 1$).

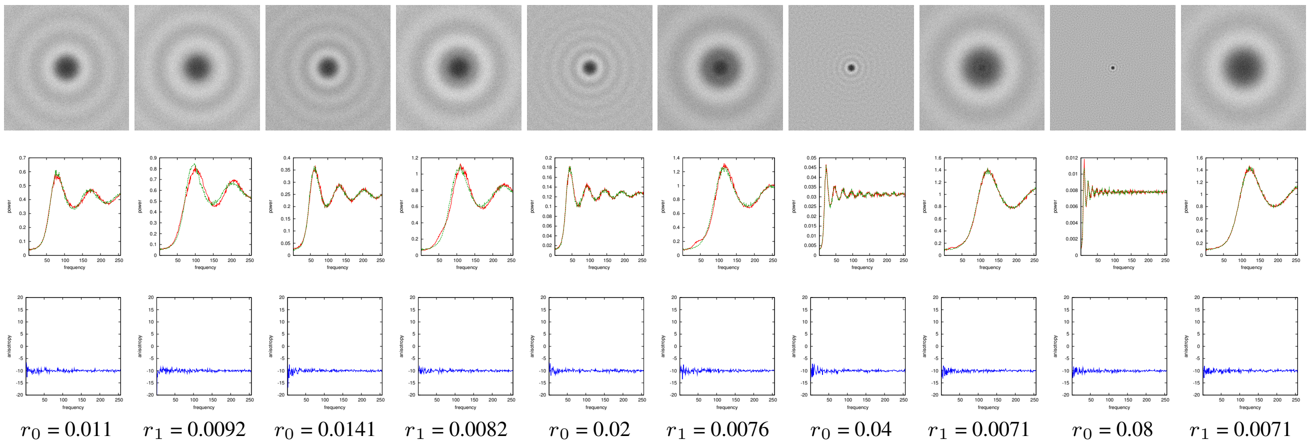


Figure 7: Spectrum results for two classes with different r values. From top to bottom: power spectrum averaged over 10 runs, radial mean power, and radial variance/anisotropy. Each pair of column is produced together with different r_0 and r_1 values so that their total is a sample set with $r = \frac{0.01}{\sqrt{2}}$. For the radial mean plots, we also overlay the ground truth produced by single-class dart throwing as green curves for easy comparison.

In general, we believe it is geometrically impossible to remove the small hump in the radial mean plot of the class with a smaller r value, and we have found that our approach distributes these extra energy better than other alternatives.

Another issue with discrete relaxation is that it will not necessarily produce a set of multi-class Poisson disk samples satisfying the set of user specified per-class parameters $\{r_i\}_{i=0:c-1}$. Specifically, there is no guarantee that two samples s and s' will satisfy the Poisson-disk requirement $|s - s'| \geq \mathbf{r}(c_s, c_{s'})$.

Similar to the discrete randomization and relaxation methods, we could also entertain the notion of *discrete dart throwing*, where we let our algorithm draw samples from a pre-existing sample set rather than a continuous domain. In general, this is a problematic approach, as the it tends to overshoot the r parameters (due to constraints imposed by discrete source), causing the output sample sets to be overly sparse.

r-matrix computation Figure 10 demonstrates the effects of the $\mathbf{r}$-matrix on sample quality for a 3-class scenario. In the left case, we compute the off-diagonal entries of $\mathbf{r}$ uniformly via Equation 2. In the middle case, we compute $\mathbf{r}$ via Program 2. (We have to use more than 2 classes because otherwise these two cases would be equivalent.) Note that in both cases class c_0 have very similar results to the ground truth; this is to be expected as c_0 has the largest r value, causing it to have the smallest inner ring in the power spectrum and thus immune from contaminations from other two classes. However, for classes c_1 and c_2, the results are quite different. Since the $\mathbf{r}$ is computed in a prioritized order in the middle case, it clearly has a better distribution for c_2 than the left case. The middle case does have a slightly worse class c_1 than the left case (because c_1 is more constrained), but the difference is quite minor.

Class priority One might also question why we have to generate all classes together instead of producing them sequentially. As we discussed in Section 2.3, generating all classes together (while maintaining the consistency of their fill rates) allows us the flexibility to terminate the computation at any time. Generating the classes sequentially certainly does not allow us to do this, as we will have to specify a certain criteria to stop the generation of one class and start 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 the right case of

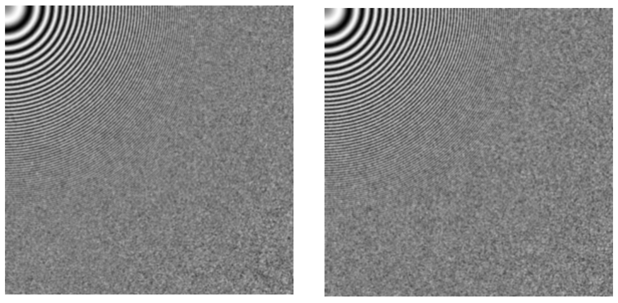


Figure 8: Spatial sampling quality comparison via the zone plate pattern $\sin(x^2 + y^2)$. Each image is produced with 166400 samples (roughly 1 sample per pixel) and filtering with a 3 pixel wide Gaussian kernel. The left image is produced from the single-class ground truth and the right image from our method as shown in the r_1 = 0.0076 case in Figure 7 where the two distributions deviate the most.

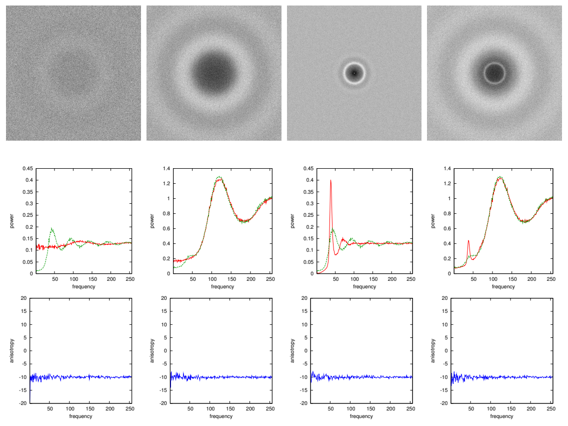


Figure 9: Optimality of our sample distribution. There are many possibilities for distributing 2 classes over a sample set; the two extreme cases are randomized (left) and discrete relaxation (right). Here we take the total sample sets from the third case (r_0 = 0.02 and r_1 = 0.0076) in Figure 7 where our sample distribution exhibits the most visible hump. For the radial mean plots, we also overlay our results as green curves for easy comparison.

Figure 10, since classes c_0 and c_1 are produced prior to c_2, it might have little rooms left and thus forced to distribute in a non-optimal way. This can be clearly seen by comparing the radial mean plot for c_2 between the middle and right cases.

Total sample set So far we have only shown the power spectrums of the individual classes but not the entire sample set. Since the entire set is not directly utilized for sampling, they just need to remain reasonably uniform to allow the maximum possible packing of samples. Figure 11 shows the power spectrums for a selected set of results from Figure 6 and Figure 7. As shown, the power spectrums of the entire sets stay pretty close to the ground truth blue noise profile, even though not identical. In particular, the maximum deviations on the radial mean curves happen around $\{\frac{1}{r_i}\}$ of the underlying classes. The deviation is most obvious when there are a small number of classes and/or when the classes have similar (but different) r values. When there are a larger number of classes, each individual sample would have a higher chance of being neighbor with a sample from a different class, thus geometrically making the entire sample set more similar to a single-class Poisson disk distribution; see the progression of the left five cases of Figure 11. When the classes have dissimilar r values, the class with larger r would have less samples to impact the overall power spectrum; see the progression of the right five cases of Figure 11.

Performance Using a simple grid-based data structure [Bridson 2007; Wei 2008] for storing samples and checking conflicts, our current implementation is able to achieve reasonable performance, as tabulated in Table 1. The performance decreases with the increasing number of classes since the sample placement is more constrained, resulting in more potential rejections during the generation process. We wish to emphasize that we have not attempted any further speed optimizations beyond the basic grid data structure (our current implementation is fast enough to produce results shown in the paper), and the performance is likely to increase significantly via more advanced sequential [Dunbar and Humphreys 2006; White et al. 2007] or parallel [Wei 2008] techniques.

# classes	1	2	3	4	5	6	7	8
performance	9.85	3.80	2.74	2.24	1.36	1.15	0.99	0.65

Table 1: Performance of our algorithm. 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.

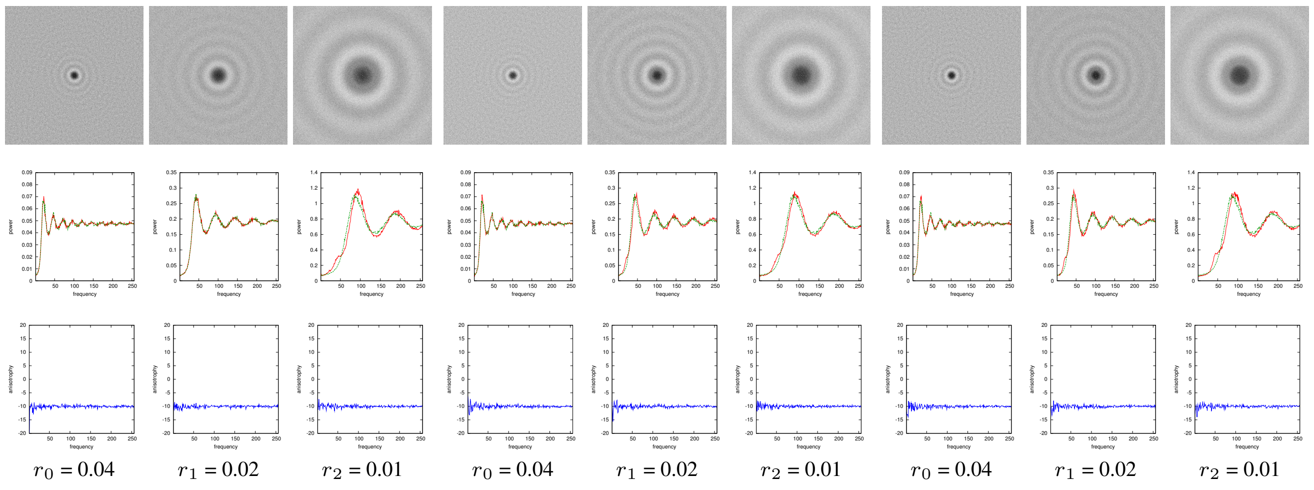
4 Adaptive Sampling

So far we have described multi-class dart throwing only for uniform sampling. Here we describe how to extend it for adaptive sampling. As summarized in Program 3, 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 $\{r_i(.)\}_{i=0:c-1}$ for the latter. This requires us to extend the definition of the $\mathbf{r}$-matrix to be spatially varying, as well as the conflict check metric in dart throwing and the hierarchy in determining if a sample s' is removable relative to s. See colored portions in Program 3:

- For building $\mathbf{r}$-matrix, we simply apply the algorithm in Program 2 for every sample location s, i.e. $\mathbf{r}(s)$ = BuildRMatrix($\{r_i(s)\}_{i=0:c-1}$).
- For conflict check, we use $\frac{\mathbf{r}(s,c_s,c_{s'})+\mathbf{r}(s',c_{s'},c_s)}{2}$ 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 hierarchizing the samples in Removable(), we use the sample location s in addition to its class number c_s utilized in Program 1. Similar to our uniform sampling algorithm, it can be easily shown that the tuples (s, c_s) are strictly ordered and thus will not introduce infinite loops for our algorithm.

Discussion Our adaptive sampling algorithm in Program 3 will reduce to our uniform sampling algorithm in Program 1 if the input parameters $\{r_i(.)\}_{i=0:c-1}$ happen to be constants.

One might also question why not simply treat each individual sample as a separate class and apply our uniform sampling algorithm. This is not doable not only from a computation point of view (e.g. a huge $\mathbf{r}$-matrix) but also it fundamentally makes no sense: in our definition, samples of each class could be used independently from other classes but this certainly does not apply to individual samples of an adaptive sampling.



classes thrown together
r has uniform off-diagonal entries via Eq 2

classes thrown together
r computed via Program 2

classes thrown in priority order ($c_0 \prec c_1 \prec c_2$)
r computed via Program 2

Figure 10: **r**-matrix computation and class priority. From top to bottom: power spectrum averaged over 10 runs, radial mean power, and radial variance/anisotropy. In the radial mean plots, the red curves correspond to results produced in each case, and the green curves the ground truth produced by single-class dart throwing with the same number of samples. Note that a **r**-matrix with uniform off-diagonal entries would cause noiser power spectrum for c_2 (compare the radial mean plots between the left and middle cases for c_2 in frequency range $[25\ 50]$). Generating the clases separately would introduce low frequency noise for the classes generated latter (compare the middle and right cases).

```
function $S \leftarrow$ MultiClassDartThrowing$(\Omega, \{r_i(.)\}_{i=0:c-1})$
    // $\Omega$: sampling domain
    // $\{r_i(.)\}$: spatially-varying parameters for intra-class sample spacing
    // c: number of classes
    // $\mathbf{r}(.)$: $c \times c$ spatially-varying matrix controlling inter-class sample spacing
    $\mathbf{r}(.) \leftarrow$ BuildRMatrix$(\{r_i(.)\}_{i=0:c-1})$ // see Program 2
    $S \leftarrow \emptyset$ // final set of samples
    while not enough trials attempted and not enough samples in $S$
        $s \leftarrow$ new sample uniform-random drawn from $\Omega$
        $c_s \leftarrow \arg\min_c$ FillRate $(c)$ // choose the most under-filled class
        if $\forall s' \in S \quad |s - s'| \geq \frac{\mathbf{r}(s, c_s, c_{s'}) + \mathbf{r}(s', c_{s'}, c_s)}{2}$
            add s to $S$
        else if impossible to add another sample to $c_s$
            // try to remove the set of conflicting samples $N_s$
            $N_s \leftarrow \bigcup s' \in S$ where $|s - s'| < \frac{\mathbf{r}(s, c_s, c_{s'}) + \mathbf{r}(s', c_{s'}, c_s)}{2}$
            if Removable$(N_s, s, \mathbf{r}(.))$
                remove $N_s$ from $S$
                add s to $S$
            end
        end
    end
    return $S$

function float FillRate$(c)$
    return $\frac{\text{\# of existing samples} \in c}{\text{target \# of samples for } c}$ // see Equation 1

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

Program 3: Multi-class dart throwing for adaptive sampling. The colored portions highlight the differences from the uniform sampling in Program 1.

5 Application

5.1 Object distribution

Uniform object placement is often desirable for both scientific (e.g. biological distribution) and artistic applications. Such a uniform distribution could be achieved by either geometry packing [Kim and Pellacini 2002] or Poisson disk sampling [Cohen et al. 2003; Lagae and Dutré 2005; Kopf et al. 2006]. We could apply our approach to place multiple classes of objects so that each individual class as well as their union exhibit Poisson disk distribution. An example is shown Figure 12 for placing two classes of objects: red and yellow flowers. Our approach can be applied for both uniform and adaptive object distribution.

5.2 Color stippling

In addition to object placement, Poisson disk sampling could also be employed for stippling with visually pleasing results (see e.g. [Kopf et al. 2006]). However, existing stippling results are mostly black-and-white since traditional Poisson disk sampling can handle only a single class of samples. We could apply our algorithm for multi-color stippling by using a color image as the input importance field, and simply treating each color channel as a separate class and producing a multi-class output sample set accordingly. As shown in Figure 13, our method can produce reasonably complex color stippling, and the colored dots not only follow the input importance field but also maintain a Poisson disk distribution. Note that since Poisson disk sampling forbids samples to overlap each other, it follows that on average each class can occupy no more than $\frac{1}{c}$ of the total sample area for a c-color input. This could limit our stippling result to achieve at most $\frac{1}{c}$ total intensity of the original color image under the same display gamut. This issue could be addressed by allowing the colored dots to overlap (similar to color dithering [Ostromoukhov and Hersch 1999; Pang et al. 2008]). We leave this as a potential future work.

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 certain variations from it); as pointed out in [Ben Ezra et al. 2007], such grid layouts

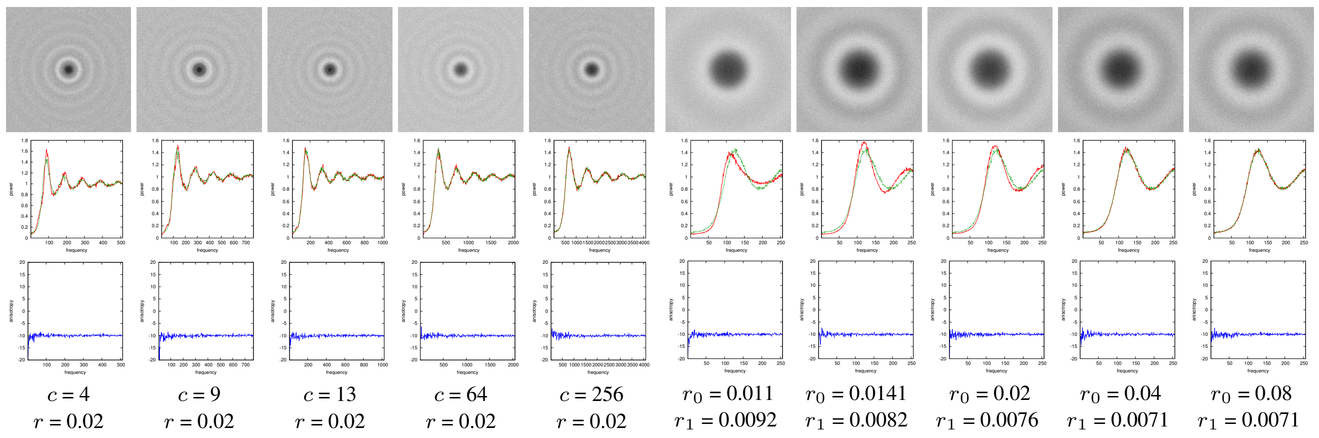


Figure 11: Total sample set distribution. The left 5 cases are from Figure 6 and the right 5 from Figure 7. For the radial mean plot, we also overlay the ground truth produced by single-class dart throwing as green curves for easy comparison.

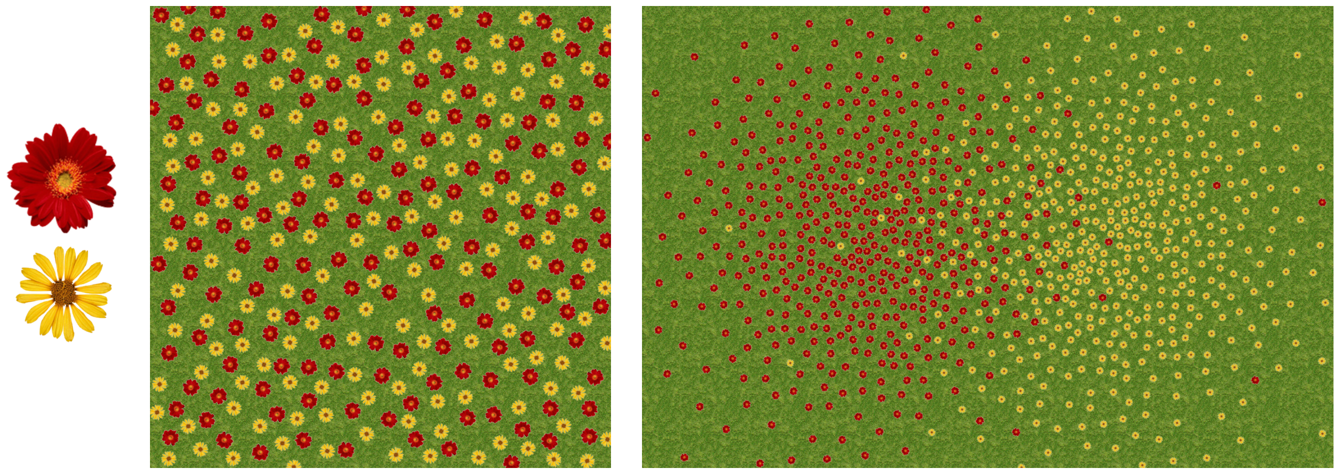


Figure 12: Object placement. Here we apply our algorithm to distribute two classes of objects: red and yellow flowers with inputs shown on the left. The middle is a case of uniform distribution. The right is a case of adaptive distribution with an input importance field consisting of two Gaussian blobs.

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], would still exhibit spectrum bias compared to one produced by Poisson disk sampling.

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 14. For the reference Penrose pixel algorithm, we use the randomized 3-coloring algorithm in [McClure 2002] to assign the RGB sensor locations. We have found this randomized algorithm improves the sampling quality for each individual color set, as their entire union is even more biased due to a deterministic layout.

Even so, we have found that our approach produces much better spectrum quality. Due to the fact that [Ben Ezra et al. 2007] could achieve 100% area utility (i.e. no gaps between sensor cells) whereas our approach could not (due to the nature of Poisson disk sampling), for fair comparison, we have produced two sets of results via our algorithm: one with similar number of samples to [Ben Ezra et al. 2007] (with larger total area) and another with similar total sensor area (with fewer samples). As shown in Figure 14, our former case 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 in prior publications (e.g. [Kopf et al. 2006; Ostromoukhov 2007; Wei 2008]). For our latter case, even though it produces a smaller inner ring in the power spectrum than [Ben Ezra et al. 2007] due to the use of fewer samples, the absence of aliasing and bias still makes our approach a potential better choice than [Ben Ezra et al. 2007].

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 layer so that the spectrum quality is still improved. This can be achieved by applying our multi-class dart throwing algorithm to an existing set of samples (regular cell layout in this particular case) rather than a continuous domain. (Or, equivalently, draw each sample from a continuous domain but snap it to the nearest cell center before drawing another one.)

As shown in Figure 15, even though our method cannot remove the

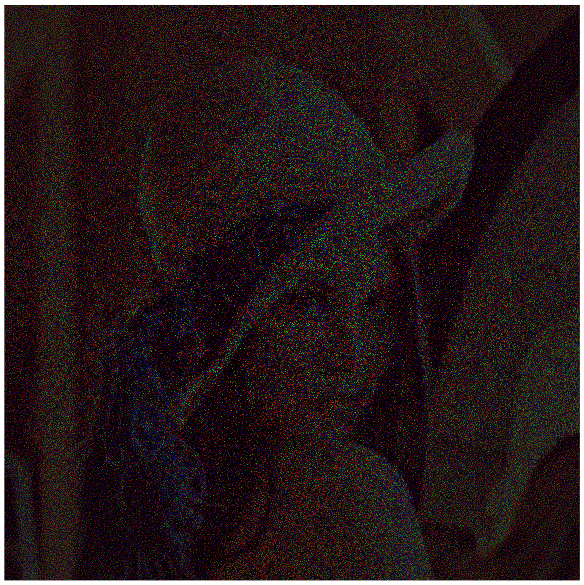


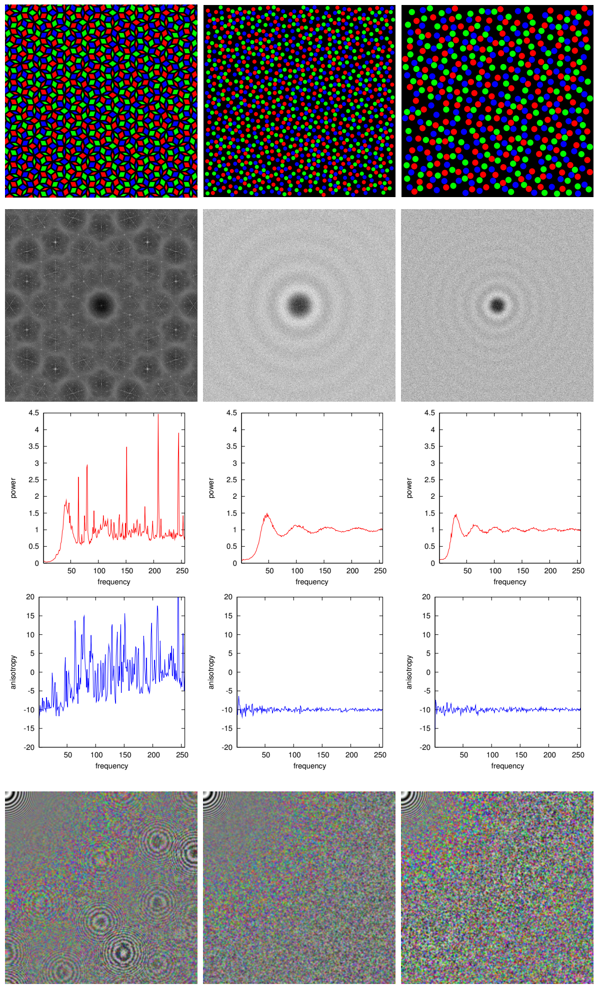
Figure 13: Color stippling result. Using the famous Lenna image as the input importance field, our method produces an adaptive sample set consisting of ∼322K RGB color dots. (Note: this image is best viewed on a computer display as it might not show up well on printing.)

bias caused by the underlying regular grid structure, the sampling quality is still improved by de-regularizing the color filter elements. (In this particular example we use the Bayer filter mosaic consisting of twice the number or G than R/B filters, but our algorithm applies to other layouts as well.) To ensure that all sensor elements are utilized, we gradually decrease the r parameters throughout the sampling process similar to [McCool and Fiume 1992]. Even though it is also possible to de-regularize the color filters via discrete randomization or relaxation (discussed in Section 3), we have found that the former tends to produce more noise while the latter might degenerate into a regular layout.

6 Limitations and Future Work

In addition to power spectrum analysis (see Section 3), another metric to evaluate Poisson disk sample quality is the relative radius $\rho = \frac{r}{r_{max}}$ as defined in [Lagae and Dutré 2008], where r_{max} is the maximum average inter-sample distance computed from the maximum packing of a given number of samples. As recommended by [Lagae and Dutré 2008], ρ should be in the range $[0.65\ 0.85]$ for traditional single-class Poisson disk sampling. However, for multi-class Poisson disk sampling, we have found it more difficult to achieve the upper end of that range due to the inherent more constraints in multi-class sampling; specifically, samples in each class not only have to keep a distance from peers with the same class but also from those in other classes. In our experience, we have found $\rho \in [0.65\ 0.70]$ achievable but beyond that might require excessive number of trials. Fortunately, this issue does not seem to worsen progressively with the increasing number of classes; due to our $\mathbf{r}$-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 $\rho = 0.67$.

We have mainly focused on the basic algorithms for multi-class sampling and only lightly touched on the issues of acceleration. Since our algorithm is extended from dart throwing, we believe it can benefit from a repertoire of previous acceleration techniques,



Penrose pixels [Ben Ezra et al. 2007] | our method same # samples | our method same total area

Figure 14: Comparison with Penrose pixels for RGB sensor layout. From top to bottom: spatial sensor layout, power spectrum averaged over 10 runs, radial mean, radial variance/anisotropy, 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. With respect to the left case, the middle one is produced by our technique with the same number of samples, while the right one has fewer samples (∼42%) occupying the same total area.

such as [Jones 2006; Dunbar and Humphreys 2006; White et al. 2007; Wei 2008]. Another related future direction is parallelization [Wei 2008]; right now our algorithm is not directly parallelizable due to the use of sample removals, but for applications that do not care about tight sample control, we could forgo both the fill-rate computation and the sample removal process, and simply draw samples according to a pre-determined probability (see discussions surrounding Figure 4); this would allow us to parallelize our algorithm similar to [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.

As discussed in Section 5.2, our color stippling results could be too dark due to the limitation of non-overlapping color dots. One possible solution is to enlarge the color dots and allow them to overlap each other, and it would be interesting to investigate the relationship between this possibility with multi-color dithering [Ostromoukhov

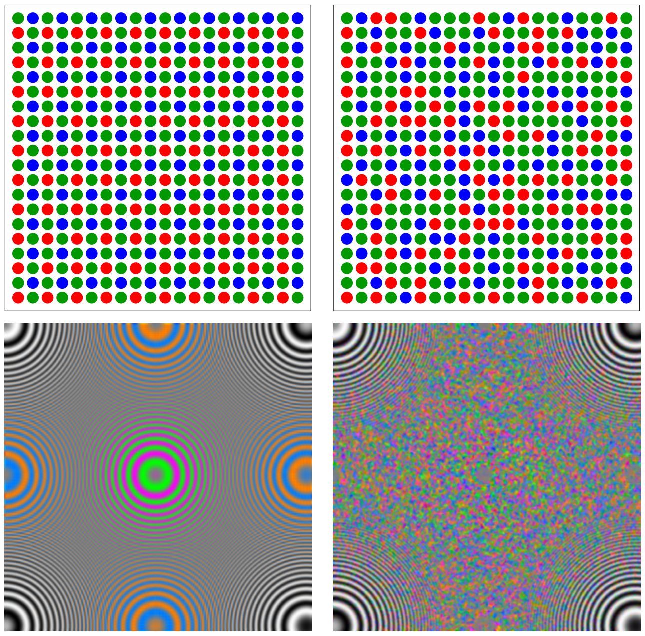


Figure 15: Color filter array design. Top: spatial filter array layout. Bottom: zoneplate sampling result. The color filter layout on the right is produced by applying our multi-class dart throwing technique to the regular grid on the left instead of a continuous domain. Note that our technique can remove biases caused by regular color filter layout, but not these caused by the underlying sensor array (i.e. aliasing near the zoneplate corners).

and Hersch 1999] or error diffusion [Pang et al. 2008].

Even though our algorithm is designed mainly for sampling continuous domains, it is also applicable to discrete domains such as color filter layout. We believe this multi-class discrete sample layout could have other interesting applications, such as stochastic rasterization [Akenine-Möller et al. 2007] where each group of time pixels could be considered as a separate class.

Although we have only demonstrated results in 2D, our algorithm is directly applicable to higher dimensional spaces [Bridson 2007; Wei 2008]. (The core component of our algorithm, dart throwing, as well as various grid/tree-based implementations are all applicable to arbitrary dimensions.) This could have potential high dimensional applications, e.g. distribution of 3D objects. It is also interesting to extend our approach to sample non-Euclidean domains such as manifold surfaces [Turk 1992; Fu and Zhou 2008]; this could have potential applications in geometry processing. Another potential future work is extensions for anisotropic sampling [Feng et al. 2008]. This could be useful for a variety of applications, e.g. anisotropic stippling [Kim et al. 2008].

References

AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. 2007. Stochastic rasterization using time-continuous triangles. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 7–16.

BEN EZRA, M., LIN, Z., AND WILBURN, B. 2007. Penrose pixels super-resolution in the detector layout domain. In *ICCV '07: Proceedings of the International Conference on Computer Vision*, 1–8.

BRIDSON, R. 2007. Fast poisson disk sampling in arbitrary dimensions. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Sketches & Applications*.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, 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: ACM SIGGRAPH 2006 Papers*, 503–508.

FENG, L., HOTZ, I., HAMANN, B., AND JOY, K. 2008. Anisotropic noise samples. *IEEE Transactions on Visualization and Computer Graphics 14*, 2, 342–354.

FU, Y., AND ZHOU, B. 2008. Direct sampling on surfaces for high quality remeshing. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, 115–124.

JONES, T. R. 2006. Efficient generation of poisson-disk sampling patterns. *journal of graphics tools 11*, 2, 27–36.

KIM, J., AND PELLACINI, F. 2002. Jigsaw image mosaics. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 657–664.

KIM, D., SON, M., LEE, Y., KANG, H., AND LEE, S. 2008. Feature-guided image stippling. In *Eurographics Symposium on Rendering*.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive wang tiles for real-time blue noise. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, 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.

MCCLURE, 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 *Proceedings of the conference on Graphics interface '92*, 94–105.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 65–72.

OSTROMOUKHOV, V., AND HERSCH, R. D. 1999. Multi-color and artistic dithering. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 425–432.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Papers*, 78.

PANG, W.-M., QU, Y., WONG, T.-T., COHEN-OR, D., AND HENG, P.-A. 2008. Structure-aware halftoning. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*.

TURK, G. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 55–64.

WEI, L.-Y. 2008. Parallel poisson disk sampling. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*.

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.

Supplementary Materials

A Implementation

So far we have only described how our algorithms work in an abstract, high level metaphor of dart throwing. Even though dart throwing could provide high quality, it is also well known for its slow computation speed. Here we provide more implementation details about acceleration and other implementation issues.

A.1 Single-resolution grid

One possibility to accelerate dart throwing is via a grid data structure as described in [Bridson 2007; White et al. 2007; Wei 2008]. The basic idea there is to subdivide the sample domain Ω into grid cells so that each cell can contain at most one sample. Thus, for uniform sampling, one only needs to examine a constant number of cells surrounding a new trial sample for conflict check. This grid data structure can be trivially extended for our multi-class uniform sampling algorithm, and we could also use the grid cells to track the available remaining space for each class of samples (for the purpose of estimating impossibility to add samples in Program 1). However, since the grid has to be built fine enough to accommodate the minimum value in our $\mathbf{r}$-matrix while the conflict check has to be conservative enough for the maximum value in the $\mathbf{r}$-matrix, the computation cost of a single-resolution grid implementation of our algorithm will increase linearly with respect to the ratio of the $\max$ and $\min$ values of $\mathbf{r}$. This situation is further exacerbated in adaptive sampling.

A.2 Multi-resolution tree

The aforementioned issues for a single-resolution grid could be addressed by a multi-resolution tree structure as described in [Wei 2008]. The basic idea there is to store larger samples (in terms of $r(.)$) at a lower resolution of the tree while the smaller ones at higher resolutions, so that the conflict check can be performed by looking at a constant number of tree nodes at each resolution.

The multi-resolution single-class algorithm in [Wei 2008] could be combined with our single-resolution multi-class algorithm in Program 3. The hybrid algorithm for multi-resolution multi-class sampling is summarized in Program 4. Our main idea is to use c individual trees to store each class of samples, and for each new trial sample s we perform conflict check across multiple trees for all existing samples that have potential for conflict. (The algorithm could be visualized as the spatial overlay of c individual trees for each class of samples.) Since the number of samples stored in each tree can be quite different, we also subdivide the trees on demand instead of all together. This means that at any time the trees could have different number of levels. We traverse the lead level nodes of any tree in a randomized order to avoid bias (as discussed in [Wei 2008]).

The main difference between our algorithm and [Wei 2008] is on cross conflict check between a trial sample s and a tree T for different classes. Specifically, when s is generated from its own (same class) tree T_s, s must be drawn from a highest resolution leaf cell $\square_s$ and thus all the math properties of [Wei 2008] hold. However, when s is conflict-checked against a different tree T, $\square_s$ might not even have a twin cell in T, or the twin cell exists but is not on the highest resolution of T. Thus, the math and algorithm in [Wei 2008] cannot be directly applied. To handle these situations, we extend the approach in [Wei 2008] as follows. Let the trial sample s be generated at level l from T_s. In [Wei 2008], the conflict check is performed by looking at, for each resolution l' from l to 0, a set of cells within $3\sqrt{n}\mu(l')$ distance from the ancestor cell $\square(l')$ containing s. In our approach, we simply look at cells within the same vicinity of the ancestor cells not only within the same tree T_s but also at all other tree T. To accommodate for the aforementioned situations, we make the following modifications: (1) we only examine cells that actually exist in T and (2) if $l_{max}(T) > l$, for any non-leaf cell $\square$ at level l of T, we have to examine all samples contained within its sub-tree. (This situation never happens in T_s as $l_{max}(T_s) = l$.)

To mathematically prove that the algorithm is correct, simply follow the math proofs in [Wei 2008] and take into account the fact that $\forall s \in \Omega$, the off-diagonal entries in $\mathbf{r}(s, ., .)$ are all smaller than its diagonal entries according to our $\mathbf{r}$-matrix construction algorithm in Program 2. See Appendix B for math details.

A.3 Impossibility Estimation

A core component of our algorithm is to estimate when it is impossible to add samples to a specific class. For uniform sampling, this can be precisely estimated by tracking the remaining available space for each class in the merit of [Dunbar and Humphreys 2006]: for each newly added sample s in class i, it will strike out a spherical region centered at s with radius $\mathbf{r}(i, j)$ out of the remaining available space for class j. However, it is not clear how to extend this strategy for adaptive sampling where $\mathbf{r}(.)$ can be spatially-varying. In addition, it can be quite complex to implement, especially for high dimensional spaces [Bridson 2007; Wei 2008].

Fortunately, for our algorithm, all we need is a rough estimation for impossibility rather than a 100% precise measurement. In our current implementation, we simply use the grid/tree cells to track available regions: for each newly added sample s in class i, we strike out cells for class j that are entirely within the spherical region centered at s with radius $\mathbf{r}(s, i, j)$. (Note that this works for both uniform and adaptive sampling, and we strike out cells only for the sake of impossibility estimation, not really removing them from the candidate list $\{\square\}_j^{l_j}$ in Program 4.) We have found this strategy work well in practice.

B Math Details

Claim B.1 *The hyper-sphere radius utilized in Program 4 is conservative enough to check all potential conflicts (for both the* mean *and* max *metrics).*

Proof For clarity, for the discussion below we use the mainly the $\max$ metric (as in [Wei 2008]); the mean metric could be proved analogously.

Let's consider the conflict check between a new sample s in class c_s and an existing sample s' in class $c_{s'}$. When $c_s = c_{s'}$, the situation reduces to the single-class algorithm in [Wei 2008]. When $c_s \neq c_{s'}$, note that (1) Claim A.1 in [Wei 2008], which states that for any existing sample s generated in tree level l we have $\mathbf{r}(s, c_s, c_s) \leq 2\sqrt{n}\mu(l)$, still holds true for each individual class, and (2) for each trial sample s produced in level l, $\mathbf{r}(s, c_s, c_s)$ must be $\leq 2\sqrt{n}\mu(l')$ for each l' = 0 to l, to survive the conflict check within its own tree T_{c_s}, as in the last statement of Claim A.2 in [Wei 2008]. These two properties, together with the fact that for each s', $\mathbf{r}(s', c_{s'}, c_s) < \mathbf{r}(s', c_{s'}, c_{s'})$, (i.e. the inter-class distances are smaller than intra-class ones due to our construction algorithm in Program 2), inform us that to check conflict between a trial sample s and a node $\square(l')$ $\in$ level l' ($\leq l$ of s) of tree $T_{c_{s'}}$ of a different class $c_{s'}$, the distance $3\sqrt{n}\mu(l')$ would be enough to conflict-check s with all samples $\in \Omega\left(\square(l')\right)$. This, in turn, allows us to check all samples contained in non-leaf cells at level l as well as all leaf cells at level $l' < l$ of tree $T_{c_{s'}}$. ∎

C Basic Formulation

Let $\{\mathbf{p}^k\}_{k=0 \text{ to } N-1}$ be a set of N samples in a n dimensional space. Their Fourier transform can be computed as follows:

$$F(\mathbf{f}) = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i (\mathbf{f}.\mathbf{p}^k)} \tag{3}$$

where $\mathbf{f}$ is the frequency.

For the purpose of analyzing sample patterns, we are usually interested in the power spectrum $F(\mathbf{f})$ [Lagae and Dutré 2008], which can be computed as follows:

$$\begin{aligned} P(\mathbf{f}) &= |F(\mathbf{f})|^2 = P_r(\mathbf{f}) + P_i(\mathbf{f}) \\ P_r(\mathbf{f}) &= \left(\frac{1}{N} \sum_{k=0}^{N-1} \cos(2\pi \mathbf{f}.\mathbf{p}^k) \right)^2 \\ P_i(\mathbf{f}) &= \left(\frac{1}{N} \sum_{k=0}^{N-1} \sin(2\pi \mathbf{f}.\mathbf{p}^k) \right)^2 \end{aligned} \tag{4}$$

We can reformulating Equation 4 via basic trigonometry into the following equivalent form:

$$\begin{aligned} P(\mathbf{f}) &= \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} \cos\left(2\pi \mathbf{f}.(\mathbf{p}^k - \mathbf{p}^j)\right) & (5) \\ &= \frac{1}{N} + \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{j=0, j \neq k}^{N-1} \cos\left(2\pi \mathbf{f}.(\mathbf{p}^k - \mathbf{p}^j)\right) & (6) \end{aligned}$$

Note that unlike Equation 4 which depends on explicit sample locations, Equation 6 depends on only pair-wise sample location differences. This property provides us with an intuitive explanation of the blue noise profile for Poisson disk sampling; see Appendix D.

D Explanation for Blue Noise Shape

We could use Equation 6 to help explain the famous blue noise profile (i.e. radial mean of the power spectrum $P(\mathbf{f})$) consisting of two main signature characteristics: lack of energy in the inner ring ($\mathbf{f} < \frac{1}{r}$) and the undulations. See Figure 16.

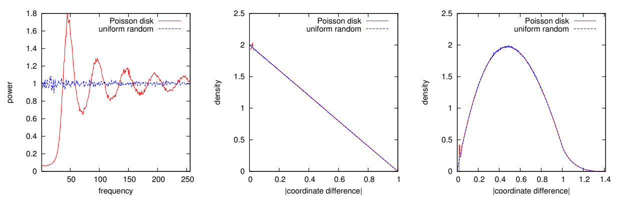


Figure 16: Power spectrum analysis for Poisson disk and uniform random distributions. Left: average mean power averaged over 10 runs. Middle: probability density functions of the random variable $\left|\mathbf{p}_x^k - \mathbf{p}_x^j\right|$, $k \neq j$. Right: probability density functions of the random variable $\left|\mathbf{p}^k - \mathbf{p}^j\right|$, $k \neq j$. Notice the spike at r (and the groove at 0) for the Poisson disk relative to the uniform distribution. Dimension $n = 2$ and $r = 0.02$.

We start with white noise and use that as a reference to explain the differences with Poisson disk sampling. For a white noise, the sample set $\{\mathbf{p}^k\}$ is uniformly sampled from $[0\ 1]^n$, thus by basic probability analysis we have $\overline{P(\mathbf{f})} = \frac{1.0}{N}$ for all $\mathbf{f}$. The term white noise comes from the fact that this profile contains roughly equal energy at all frequencies, as shown in Figure 16. (We normalized the plots with respect to the number of samples N.)

Compared to the flat white noise radial profile, a Poisson disk sample set has lower energy for frequency ($\mathbf{f} < \frac{1}{r}$ as well as the intriguing undulations. These two main deviations could be explained by the histogram distribution of the random variable $\mathbf{p}^k - \mathbf{p}^j$. In Figure 16, we plot the histograms for both $\left|\mathbf{p}^k - \mathbf{p}^j\right|$ in the original 2D space as well as the 1D projection $\left|\mathbf{p}_x^k - \mathbf{p}_x^j\right|$. As shown, the white noise has the standard linear profile while the Poisson disk sample set has higher concentration around r as well as a lack of values $< r$. This is to be expected as $\left|\mathbf{p}^k - \mathbf{p}^j\right| < r$ is completely forbidden by the sample generation process. However, the 1D profile will have some values $< r$ due to the nature of projection. We can use the histogram profiles combined with Equation 6 to explain the formation of the signature blue noise profile as follows:

Reduced low frequency energy For a small $\mathbf{f}$, removing a small distance pair $\mathbf{p}^k - \mathbf{p}^j$ from Equation 6 is going to reduce its energy, as these small values will tend to concentrate around the 0 argument for cos. This removal of smaller distance pairs is a nature consequence of Poisson disk sampling, as evident from the histograms in Figure 16. However, as $\mathbf{f}$ increases, the removal of energy will start to spread across all ranges (and possibly multiple lobes) of cos, diminishing the effect. (Try to imagine a sequence starting with $\mathbf{f} = 1$ and gradually increases to 2, 3, etc.)

Undulations The cos terms in Equation 6 serve essentially as detectors for any particular value of pair-wise sample differences $\mathbf{p}^k - \mathbf{p}^j$. For Poisson disk sampling, since the differences have an usual higher concentration around r (again due to the nature of Poisson disk sampling process; see histograms in Figure 16), it will cause excessive undulations for multiples of $\mathbf{f} = \frac{1}{r}$. Specifically, for $\mathbf{f} = \frac{m}{r}$ where m is any integer, $\overline{P(\mathbf{f})}$ will have local maximums as these matches the peaks of the cos. Conversely, for $\mathbf{f} = \frac{m+0.5}{r}$, $\overline{P(\mathbf{f})}$ will have local minimums as these matches the valleys of the cos. And the undulation amplitudes diminishes with respect to increasing frequency as the clustering around r essentially get dispersed.

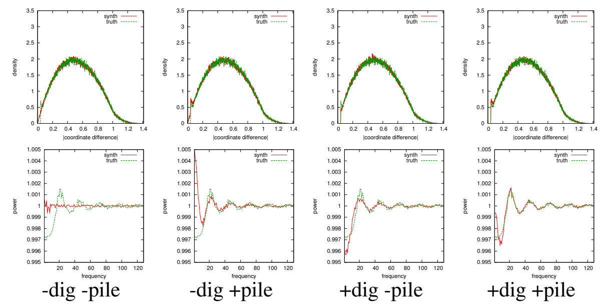


Figure 17: Synthetic sample differences. Here, we generate the random variables $\mathbf{p}^k - \mathbf{p}^j$ directly instead of from real sample sets. The red curves are our synthetic results while the green curves ground truth from real Poisson disk sample sets. Our baseline generator is a white noise, and we provide 2 orthogonal twists, dig and pile.

To further verify these observations, we have performed more experiments as in Figure 17. There, instead of from real sample sets, we directly manipulate the pair-wise difference random variable $\mathbf{p}^k - \mathbf{p}^j$; this flexibility allows us to gain further insights. Our baseline generator is a white noise, and on top of that we provide 2 orthogonal twists, dig and pile. The former option allows us to remove pairs with distance $< r$, while the latter allows us to spawn any pair $< r$ into a pair with identical direction but length $\sim r$. These two orthogonal options allow four possible combinations. As shown in Figure 17, the no-dig-no-pile option is essentially the baseline white noise, while the yes-dig-yes-pile option essentially simulates a Poisson disk sampling. The other two options have no physical counter parts in real sample distributions but allow us to explore further insights. In particular, the yes-dig-no-pile option clearly shows that, after the removal of pairs $< r$, the power spectrum radial profile has the expected reduction of low frequency energy. However, when the pile option is turned on (second and fourth case), we get enhanced undulations and also higher low frequency energy due to the spillover effects caused by projection from 2D onto the 1D profile where the produce $2\pi\mathbf{f}.(\mathbf{p}^k - \mathbf{p}^j)$ inside Equation 6 occurs.

E Analysis for Multi-class Sampling

Once we appreciate the reasons behind the signature blue noise profile shape for single-class sampling (Appendix D), we could apply similar machineries for multi-class sampling. Essentially, the central mystery for multi-class sampling beyond single-class sampling is the fact that a class with a larger r value could affect another with a smaller r, as the spectrum peak of the former could fall into the inner ring of the latter (Figure 5). Specifically, consider the 2-class scenario where $r_0 > r_1$. For class 0/1, the sample location differences will cluster around r_0/r_1 just as analyzed in Appendix D. However, a strange phenomenon is that for class 1, the location difference histogram also has a higher value around r_0. (See Figure 18.) We have yet any theoretical explanation for this, but the effect of this secondary peak is manifested in the spectrum peaks around frequency $\frac{1}{r_0}$. And as shown in Figure 9, the spikiness of the power spectrum radial mean around $\frac{1}{r_0}$ for class 1 is proportional to the one in class 0. Our algorithm mitigates this inter-class spike interference by generating all classes together.

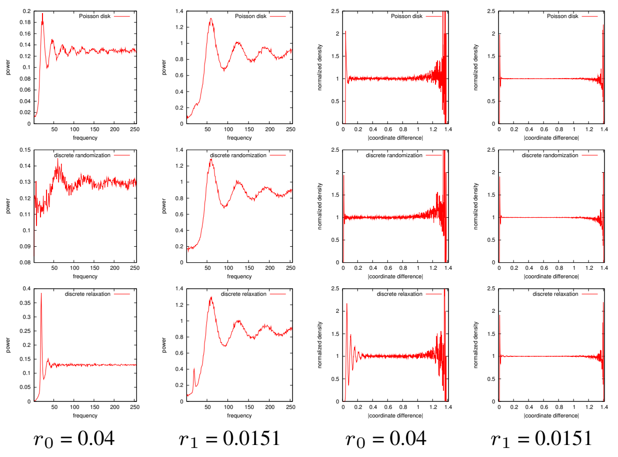


Figure 18: Histogram analysis for 2-class distributions. From top to bottom: our method, discrete randomization over our results, and discrete relaxation over our results. From left to right: radial mean plots, and histogram of the random variable $\left|\mathbf{p}^k - \mathbf{p}^j\right|, \ k \neq j$, normalized with respect to a white noise (values between [1 1.4] are noiser due to reduced sampling density constrained around the domain corners). The hump effect is most pronounced for the discrete relaxation case; notice the peaks around frequency 19 in the radial mean plots for both classes. This corresponds to higher 1D histogram (not shown) value of $\frac{1}{19}$ as a result of the projection from the 2D histogram (shown on the right).

```
function MultiClassAdaptiveSampling($\Omega$, $\mathbf{r}(.)$, $k$)
  // $\Omega$: sampling domain in $n$-dimension
  // $\mathbf{r}(.)$: $c \times c$ $\mathbf{r}$-matrix defined over $\Omega$; see Program 3
  // $k$: maximum number of trials per node
  // use separate trees to track each class of samples
  $\{T_i(0)\}_{i=0:c-1}$ ← BuildNDTreeRoots($\Omega$) // hypercubes covering $\Omega$
  foreach class i   $l_i \leftarrow 0$   // track the leaf level number for each $T_i$
  foreach class i   $\{\square\}_i^{l_i}$ ← randomized list of (leaf) nodes $\in T_i(l_i)$
  while not enough trials attempted and not enough samples in $\{T_i\}$
    $\jmath \leftarrow \arg\min_c$ FillRate $(c)$ // choose the most under-filled class
    if $\{\square\}_\jmath^{l_\jmath} = \emptyset$ // no more leaf nodes to sample from; try subidivide $T_\jmath$
      $T_\jmath(l_\jmath + 1)$ ← Subdivide($\Omega$, $\mathbf{r}(.)$, $T_\jmath(l_\jmath)$)
      if $T_\jmath(l_\jmath + 1) = \emptyset$   break   // impossible to add another sample
      $l_\jmath \leftarrow l_\jmath + 1$
      $\{\square\}_\jmath^{l_\jmath}$ ← randomized list of (leaf) nodes $\in T_\jmath(l_\jmath)$
    end
    $\square$ ← PopFront($\{\square\}_\jmath^{l_\jmath}$) // take the head of the randomized list
    $s$ ← ThrowSample($\{T_i\}$, $\Omega(\square)$, $\mathbf{r}(.)$, $k$, $l_\jmath$)
    if $s$ is not null
      add $s$ to $\square$
    else if impossible to add another sample to class $\jmath$
      // try to remove the set of conflicting samples $N_s$
      $N_s \leftarrow \bigcup s' \in S$ where $s$ and $s'$ are in conflict
      if Removable($N_s, s, \mathbf{r}(.)$) // see Program 3
        remove $N_s$ from $\{T_i\}$
        add s to $\square$
      end
    end
  end

function $T(l+1)$ ← Subdivide($\Omega$, $\mathbf{r}(.)$, $T(l)$)
  $i$ ← class number for $T$
  foreach node $\square$ of $T(l)$
    if $\exists s \in \square$ and $\sqrt{n}\mu(\square) > \mathbf{r}(s, i, i)$
    // subdivide $\square$ only if likely to add another sample
    // $\mu(\square)$ is the cell size of $\square$
      subdivide $\square$ into $2^n$ child nodes // $n$ is the dimension of $\Omega$
      migrate $s$ into the child $\square'$ where $s \in \Omega(\square')$
    end
  end
  $T(l+1)$ ← newly created nodes
  return $T(l+1)$

function $s$ ← ThrowSample($\{T_i\}$, $\Omega(\square)$, $\mathbf{r}(.)$, $k$, $l$)
  foreach trial = 1 to $k$
    $s$ ← sample uniformly drawn from $\Omega(\square)$
    if $\forall s' \in T \ \ |s - s'| \geq \max\left(\mathbf{r}(s, c_s, c_{s'}), \mathbf{r}(s', c_{s'}, c_s)\right)$
    // this can be done by examining only $s' \in$ neighbor nodes in $T_{c_{s'}}$
    // within hyper-sphere of radius $3\sqrt{n}\mu(l')$ at level $l'$ = 0 to $l$
    // for each node in level $l$ of $T_{c_{s'}}$, look at samples under its subtree
    // can also use the mean metric $\frac{\mathbf{r}(s,c_s,c_{s'})+\mathbf{r}(s',c_{s'},c_s)}{2}$ in Program 3
    // in this case, use hyper-sphere radius $5\sqrt{n}\mu(l')$ above
    // and $\frac{1}{2}\mathbf{r}(s, i, i)$ in Subdivide()
      return $s$
  end
  return null

function bool Removable($N_s, s, \mathbf{r}(.)$)
  foreach $s' \in N_s$
    if $\mathbf{r}(s', \ c_{s'}, c_{s'}) \geq \mathbf{r}(s, \ c_s, c_s)$ or FillRate$(c_{s'})$ < FillRate$(c_s)$
     or level(s') < level(s) // add on beyond Program 3
      return false
  return true
```

Program 4: Multi-resolution adaptive sampling using separate trees to track each class of samples. The main code is a hybrid of the multi-resolution algorithm in [Wei 2008] and our adaptive sampling algorithm in Program 3. The functions Subdivide() and ThrowSample() resemble the ones in [Wei 2008]; for easy comparison, we highlight the main differences.