Eurographics Symposium on Rendering 2012
Fredo Durand and Diego Gutierrez
(Guest Editors)

Volume 31 (*2012*), *Number 4*

Fast Generation of Approximate Blue Noise Point Sets

Nima Khademi Kalantari and Pradeep Sen

Advanced Graphics Lab, University of New Mexico

Abstract

Poisson-disk sampling is a popular sampling method because of its blue noise power spectrum, but generation of these samples is computationally very expensive. In this paper, we propose an efficient method for fast generation of a large number of blue noise samples using a small initial patch of Poisson-disk samples that can be generated with any existing approach. Our main idea is to convolve this set of samples with another to generate our final set of samples. We use the convolution theorem from signal processing to show that the spectrum of the resulting sample set preserves the blue noise properties. Since our method is approximate, we have error with respect to the true Poisson-disk samples, but we show both mathematically and practically that this error is only a function of the number of samples in the small initial patch and is therefore bounded. Our method is parallelizable and we demonstrate an implementation of it on a GPU, running more than 10 times faster than any previous method and generating more than 49 million 2D samples per second. We can also use the proposed approach to generate multidimensional blue noise samples.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Antialiasing; Image Processing and Computer Vision [I.4.1]: Digitization and Image Capture—Sampling

1. Introduction

Poisson-disk sampling is widely used in various applications in computer graphics. It is particularly useful for antialiasing because of its blue noise spectrum due to its low energy annulus around the DC spike in the Fourier domain. The basic goal of uniform Poisson-disk sampling is to uniformly cover the space with samples, with the condition that two samples cannot be closer than a specified minimum distance.

The brute-force method to generate Poisson-disk samples is dart throwing [DW85, Coo86, MF92], where the sample is thrown uniformly in the space and kept only if it is farther than a minimum radius to samples already placed. Several attempts have been made to accelerate the process of dart throwing [Jon06,DH06,WCE07,GM09,KS11] by using special data structures to keep track of the available regions. More recently, Ebeida et al. [EDP*11] proposed a method for accelerating dart throwing which can be run on a GPU. This is probably the fastest method for accurate Poisson-disk sample generation known today, and they report it can generate up to 240K samples/second.

While all the above methods generate samples on the fly, there are several other methods which use precomputed data sets to generate Poisson-disk samples [HDK01, CSHD03, ODJ04, LD05a, KCODL06, Ost07, LD05b]. In nearly all of these methods, a set of sample tiles is carefully generated with certain properties (e.g., samples at the boundary of one tile respect the minimum distance criterion of samples at the boundaries of other tiles). This optimization usually takes a lot of time and is performed offline, and at run-time the tiles are placed randomly to fill the space. These methods are fast at placing tiles, but they do not have the desired blue noise spectrum (see [LD08] for detailed comparison). Among these, the method of Kopf et al. [KCODL06] is the fastest and can generate more than 2 million samples/second.

Wei [Wei08] proposed a parallel implementation of dart throwing for a GPU which, while not exactly accurate, produces results with similar quality to dart throwing. Unlike the tiling methods, Wei's technique generates samples on the fly. To our knowledge, this is the fastest method for generation of Poisson-disk samples available today, generating around 4 million samples/second as quoted in Wei's paper. For comparison, the GPU implementation of our method can generate more than 49 million samples per second.

In this paper, we propose a fast way of generating blue noise samples. The basic idea is to generate an initial set of Poisson-disk samples using any conventional technique, and then replicate this set at various locations in the final space using a process similar to convolution. We use the convolution theorem to show that the spectrum of the final samples has blue noise properties. This approach is parallelizable, and we demonstrate an implementation of it running on the GPU.

Although the method introduces errors at the boundaries between replicated blocks, we characterize the error mathematically and show that it is analytically bounded and verify the accuracy of analysis by some practical examples. This allows the user to generate fast Poisson-disk sets with specific quality. In the limit, if no error can be tolerated, our algorithm does not provide any improvement over known methods. The samples generated using our method are "blue noise" samples since their spectrum approximates the blue noise spectrum of Poisson-disk samples, although they do not preserve the minimum distance criteria. Therefore, our method is designed for applications such as Monte Carlo integration, rather than random object placement since the minimum distance criteria is not maintained.

Others have proposed related algorithms for generating a large number of samples using a replicated patch (or patches) of Poisson-disk samples, such as the work of Dippé and Wold [DW85] and Lagae and Dutré [LD05b]. However, as shown in Fig. 1 these simple methods have artifacts in their spectrum that do not make them suitable for practical use. To demonstrate this, we show a comparison of our method against [LD05b] in Section 3.

While our method is conceptually similar to tile-based methods, our goal is completely different. Tile-based methods generate large sets of Poisson-disk samples by precomputing a specific set of tiles. This tile set is very important for these approaches, and they cannot use an arbitrary tile set. The goal of our method, on the other hand, is to accelerate any existing approach for generating Poisson-disk samples. Unlike tile-based methods, we do not need to compute a special set of tiles and can use any method to generate our initial patch. In fact, our goal is to produce a large point set that has a quality similar to those produced by the input algorithm. We show we can do this to within an error that is based on the number of samples in the initial patch.

To be effective, our algorithm should have three qualities: 1) the error between our generated point set and that which would have been produced by the input algorithm should be bounded and not dependent on the final number of samples, 2) our method should be faster than the fastest known approach, and 3) our method should work with a variety of input algorithms and produce plausible results. We show our method meets these three criteria in the results section.

2. Our Proposed Algorithm

We begin by observing that the process of tiling a patch over a space is similar to convolution. For example, if we simply replicate the patch on a uniform grid, we are effectively convolving the original samples with the samples on the grid, as shown in the first row of Fig. 2. To understand why this would not work for Poisson-disk sampling, we use the convolution theorem from signal processing, which states that the Fourier transform of the convolution of two signals is the multiplication of their Fourier transforms, i.e., $\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$. In this case, one signal is the patch of samples and the other is the locations of these patches in the final space. The Fourier transform of the small patch will have the desired blue noise properties, but the spectrum of the uniform grid will be a set of uniform spikes. This means that the resulting spectrum will be a set of uniform spikes with variable magnitudes. Some previous tiling methods (e.g., [LD05a]) placed the tiles on a regular grid in this fashion, which is why they required several tiles matched with each other at the boundaries to avoid problems.

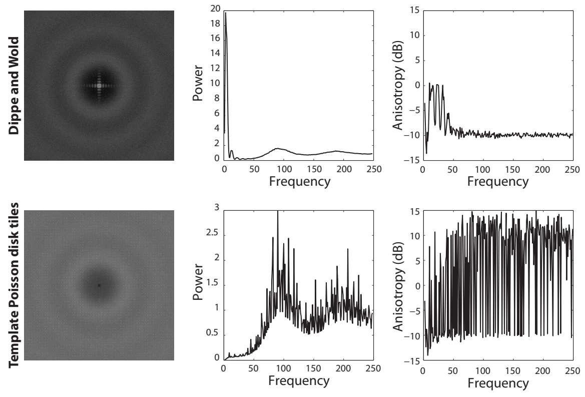

Figure 1: *(top row) The periodogram, radial power, and anisotropy of the method suggested by Dippé and Wold [DW85]. Here we rotate, replicate, and crop a small patch of samples on a uniform grid. (bottom row) the method proposed in [LD05b]. We generated a tile set containing 1024 tiles each with 80 samples and the final set of samples was obtained by randomly placing these tiles next to each other. For both methods, the spectrum results are obtained by averaging the results over 10 sets of 6,400 samples. There are severe artifacts in the spectrum results of both approaches and therefore these methods are not suitable to be used in practice.*

Therefore, the point sets we generate with our approach should satisfy three conditions: 1) the samples should be uniformly spread, 2) the samples should have a blue noise spectrum, and 3) their spectrum should match that of a set of Poisson-disk samples with a comparable number of samples. Formally, we call our initial small set of Poisson-disk samples $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{N_x}\}$, which can be generated with any previous method. We denote the set of locations to replicate this patch as $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{N_y}\}$, and the resulting set of samples as $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_{N_z}\}$.

A possible choice for $\mathbf{Y}$ is to randomly place its samples (the positions for replicating $\mathbf{X}$) with uniform distribution. This would have a constant spectrum, which would allow our resulting samples to satisfy our second condition since it would be the multiplication of a constant spectrum with a blue noise spectrum. Unfortunately, this approach does not satisfy the first condition since the random samples may be clumped together in some parts. The second row of Fig. 2 shows the spectrum results of this sampling pattern.

Based on this observation, a good choice for $\mathbf{Y}$ should be to randomly perturb a set of uniform grid samples, as in a jittered grid. These samples have a spectrum similar to random samples (constant almost everywhere), except that it has a low energy annulus at the center which makes it suitable for our application. In order to use the jittered grid, our patch size should be twice as large as the block size in each dimension to fill the whole space. As seen in the third row of Fig. 2, the jittered grid approach satisfies the first and sec-

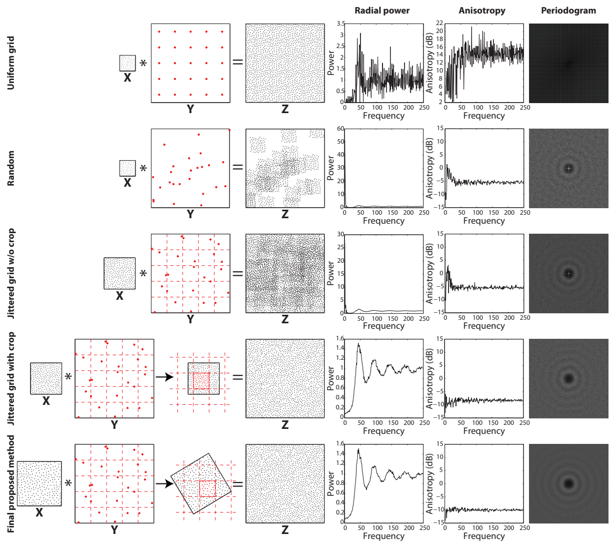

Figure 2: *In all cases $N_y = 25$ and around 1,600 samples are generated except for the third row which has around 6,400 samples. All the spectrum results are obtained by averaging over 10 sets of samples.*

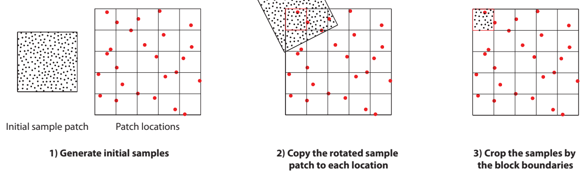
Initial sample patch
Patch locations
1) Generate initial samples
2) Copy the rotated sample patch to each location
3) Crop the samples by the block boundaries

Figure 3: *Overview of the proposed method.*

ond conditions to some extent. However, the third condition is not satisfied since there are many samples close together because of the overlap between patches. Although the resulting spectrum has blue noise properties, it is possible to generate a similar spectrum more efficiently with less samples. For example, the jittered approach has a total number of samples $N_z = 6,400$ but a similar spectra can be obtained with 1,600 samples as seen in the fifth row.

To avoid conflicts in the overlapping regions, we should crop the samples outside each block after placing the patch. Unfortunately, the cropping process breaks our convolution analogy and introduces some artifacts in the result, which look like low-frequency noise in the periodogram in the fourth row of Fig. 2. To remove these artifacts, we propose to rotate the sample patch with a random angle for each block before placing it at each patch location and cropping it by the boundaries of the block. This gives us the improved spectrum shown in the fifth row of Fig. 2. Because of this rotation, the patch size should be $2\sqrt{2}$ times bigger in each dimension than the block size for a 2D set of samples. If we extend this to n-dimensions, this value is $2\sqrt{n}$ where $\sqrt{n}$ is the length of the diagonal of a hypercube with side length equal to 1. In summary the proposed method has the three following main steps, also shown in Fig. 3:

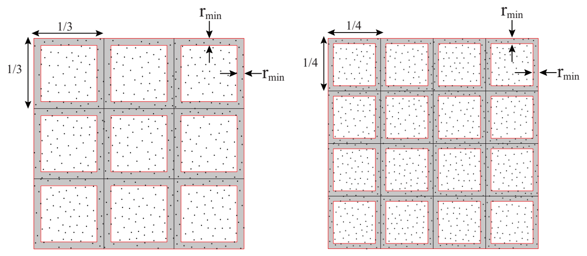

Figure 4: *On the left we have 9 blocks with size $1/3 \times 1/3$ and on the right there are 16 blocks with size $1/4 \times 1/4$. In both cases N_x is the same since we are using the same patch to tile the space. The ratio of the area of the conflict region (shown in gray) to the area of the whole space (our error metric) is constant for both cases, because the minimum radius (r_{min}) has changed between the two.*

Step 1 – Generate initial samples: We generate a patch with N_x Poisson-disk samples using any existing approach, as well as a jittered samples as the final patch locations.

Step 2 – Copy the rotated sample patch to each location: For each block, we rotate the sample patch by a random angle and put it at the jittered location in the block.

Step 3 – Crop the samples by the block boundaries: We remove the extra samples outside the block.

Steps 2 and 3 continue until all the blocks are filled with samples.

We now discuss the different parameters in our algorithm in the general n-dimensional case. The user will want to specify N_z, the total number of samples in the final set. If we are given a patch with N_x samples, we need to know how many blocks in $\mathbf{Y}$ will produce this result. If the space is tiled with m blocks in every dimension, then $N_y = m^n$. Given N_x and N_z it is easy to show that m should be calculated as such:

$$m = \lceil (2\sqrt{n}) \sqrt[n]{N_z/N_x} \rceil, \tag{1}$$

We must also choose a value for N_x. Our algorithm has error with respect to the true Poisson-disk samples because we do not test for conflicts along the block boundaries (which is what makes our algorithm so fast). It turns out that this error is only dependent on N_x. As shown in Fig. 4, the error in our approach occurs only in the boundary regions that are within r_{min} from the edge of the block. Samples in the interior of each block are correct, assuming a proper Poisson-disk method was used to generate the initial patch. We quantify the amount of error in our approximation by taking the ratio of the volume of the regions in which conflict can happen to that of the whole space. To calculate the volume of the conflict region, we examine a single block, which is of size $(1/m)^n$. In this block, the middle region which will not have

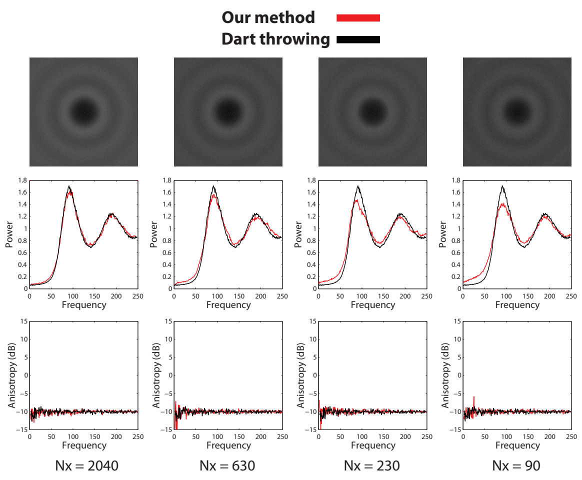

Figure 5: *The number of total samples N_z in all cases is around 6,400. As expected by our metric in Eq. 4, decreasing N_x results in a reduction in quality (more error) of the generated samples.*

any conflicts is of size $(1/m - 2r_{min})^n$, which means that the conflict region is of size $1/m^n - (1/m - 2r_{min})$. Since there are m^n blocks, our error metric gives us:

$$\varepsilon = m^n \left[1/m^n - (1/m - 2r_{min})^n\right], \tag{2}$$

where we have set the volume of the whole space to 1. Note that in Eq. 2, r_{min} is proportional to $1/\sqrt[n]{N_z}$. By substituting this in and doing some simplifications, we get:

$$\varepsilon = a(1 - (1 - 2m/\sqrt[n]{N_z})^n), \tag{3}$$

where the a is a constant to account for the proportionality between r_{min} and $1/\sqrt[n]{N_z}$. By replacing m from Eq. 1 and canceling out the $\sqrt[n]{N_z}$ terms, we get:

$$\varepsilon = a(1 - (1 - 4\sqrt{n}\sqrt[n]{1/N_x})^n), \tag{4}$$

Therefore, our error is only a function of N_x, which means that it is bounded for a fixed N_x. This means that for a fixed N_x, the user can generate an arbitrary number of samples in a space with a fixed error, or for a desired quality determine the N_x that would produce this result. We examine this effect in Fig. 5, where we vary the value of N_x for a fixed total number of samples. We see that when the N_x is reduced, the quality of the generated samples is reduced as compared to dart throwing.

Furthermore, if we keep the value of N_x constant but vary the number of blocks per dimension m to generate more samples, the quality of our results will stay the same. This can be seen in Fig. 6. For our results, we use a value of $N_x = 630$ because it maintains a sufficient level of quality.

3. Results

We implemented our method in C++ and since the method is parallelizable we used OpenMP. All timings are obtained on an Intel dual quad-core Xeon X5570 3.06GHz machine with 16GB of memory. We also implemented our algorithm on a GPU and we used OpenGL and all the timings are obtained on an NVIDIA GeForce Tesla C2070 with 6 GB of memory. Since our method aims to accelerate existing Poisson-disk sampling methods, most of our results will compare the existing method for generating all the samples to our combination of using the existing method for only generating the initial patch and then using our approach for replicating it over the entire space. This allows to demonstrate that our method produces results that are comparable to existing methods.

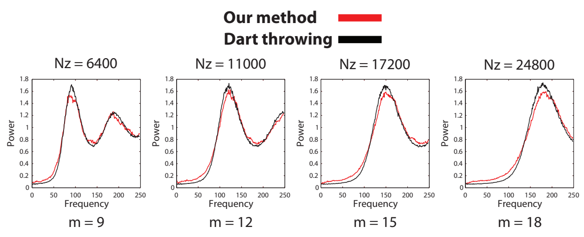

Figure 6: *We set $N_x = 630$ and generate samples using different values for m, which changes the number of samples N_z. The initial set for our method is generated using dart throwing. The radial power of dart throwing and our method are similar, regardless of the value of m.*

We start by studying the spectrum of our generated samples with different input approaches in Fig. 7. We compare with dart throwing, boundary sampling [DH06] (the fastest method among all the variations), Gamito and Maddock [GM09], and the tile-based method of Kopf et al. [KCODL06]. The total number of samples N_z generated by the boundary method and Gamito and Maddock is around 8,600 and 6,700, and for dart throwing and Kopf et al. method is around 6,400. The quality of our spectrum is comparable to those of the original methods, but our method is much faster. This shows that any existing method can be used to generate the input patch, which our method then uses to generate a large set of samples with comparable quality to the original method.

To compare against approaches such as the template Poisson disk tiles [LD05b], we can also generate several initial small patches and select them randomly and use them to fill the space, as shown in Fig. 8. For both methods, we generated the initial tiles using dart throwing. Unlike our approach, template Poisson disk tiles need processing to preserve the minimum distance criterion between all tiles before they can be used. However, despite the fact that they make an effort to respect the minimum distance criterion, the spectrum of the samples they produce is not good – it suffers from regular grid spikes. When these points are used to sample the zone plate pattern (bottom row of Fig. 8), the result has visible aliasing artifacts. Our approach, on the other hand, does not have these artifacts and the quality is comparable to dart throwing. Our result shows that if the goal is

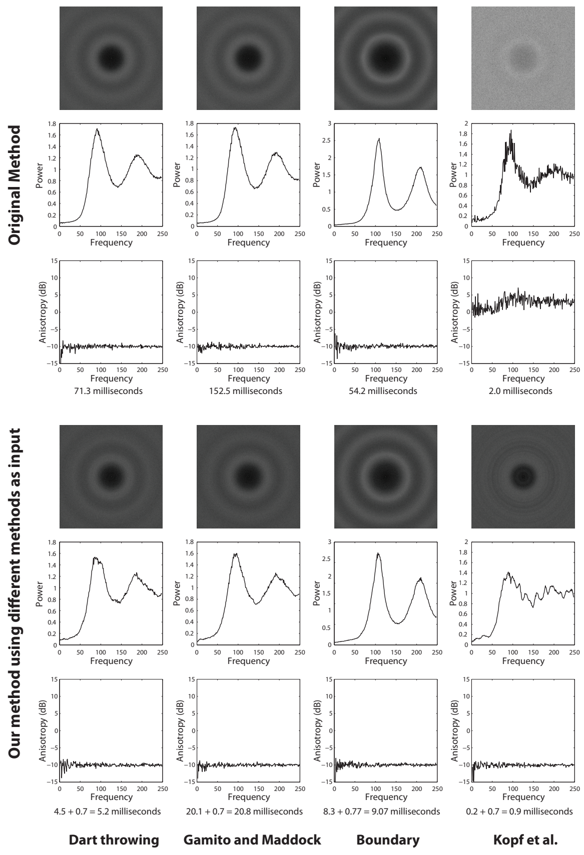

Figure 7: *The group on top shows the spectrum result for different methods while the bottom shows the result of our method by generating the input patch using the different approaches. For the boundary method [DH06], $N_x = 680$ and for all the other methods $N_x = 630$. We express the total time of our method as the sum of two numbers: the first is the time to generate the initial sample patch and the second is the time for generating the final samples using our method. These timings are obtained with our CPU implementation using 2 cores. Use of more cores in this case does not lead to speed up because of the overhead of creating the threads. The implementation codes for the original methods (except for the dart throwing algorithm) were provided by the authors and they were single threaded. As shown, our method has comparable quality and is much faster than the other methods.*

to have a good blue noise spectrum for purposes of Monte Carlo integration, then the minimum distance criterion can be ignored to some extent and that there is more value in the simple rotation and jittering of our approach.

We also show that our method is faster than any existing approach in generating a large number of samples from an initial patch to prove that it can be used to accelerate any method. For this, we generated sample sets of different sizes with our CPU and GPU implementations

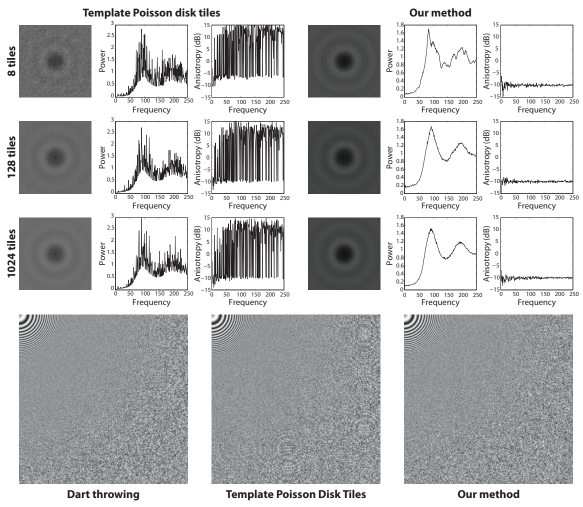

Figure 8: *We compare our method against template Poisson disk tiles [LD05b] when the total number of samples in the entire tile set is equal. Since the template Poisson disk tiles are 8× smaller than ours (they are not jittered or rotated), each of their tiles has 8× less samples (80 vs. 640 for our method). Each row shows the results with a different number of template Poisson disk tiles (8 for the top, 128 for the middle, 1024 for the bottom). Because the total number of samples in the entire set should be equal, our method used 1/8 the number of tiles in each row. The total number of samples in all the figures was equal to 6,400. The images on the bottom show the sampling of a zone plate pattern with approximately 1 sample/pixel. The template Poisson disk tiles used 1024 tiles (the best in quality of the ones shown here), while our result is generated using only one tile. As can be seen, the template Poisson disk tiles have artifacts while our method is close to the dart throwing result.*

and compared our performance to the tile-based method of Kopf et al. [KCODL06] (the fastest available). Note that the Kopf et al. method is also approximate and as Lagae and Dutré [LD08] mentioned, they can have two samples close together, like ours. We would have liked to compare with Wei's [Wei08] method as well, but the implementation is not available. However, this method has a performance which is in the ballpark of Kopf et al.'s, as mentioned in their paper. In Fig. 9 we can see that our method has linear time complexity for both CPU and GPU implementations. For a low number of samples, our GPU implementation is slower than the CPU version due to the large overhead of reading back the data, but this overhead is negligible as the number of samples increases. Our CPU and GPU implementations can generate more than 29M and 49M samples/second respectively. For comparison, Kopf et al.'s prototype implementation generates around 3.3M samples/second, although it might be possible to accelerate it and they also can produce variable den-

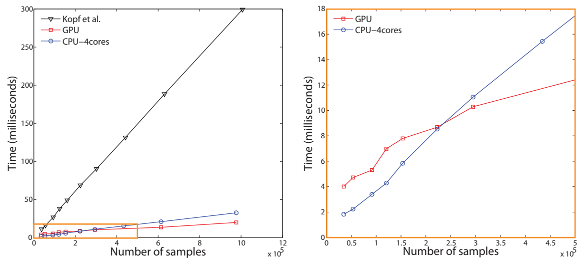

Figure 9: *(left) Time vs. number of samples for our CPU/GPU implementations compared to Kopf et al. [KCODL06] (not including the time to generate the initial samples for our approach). (right) Zoom in of the inset, only comparing our two implementations.*

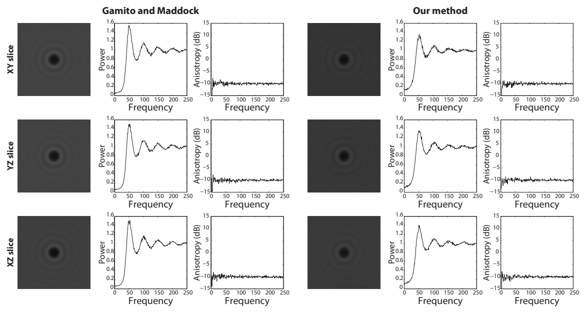

Figure 10: *(left) 2D slices of the 3D spectrum for Gamito and Maddock's [GM09] method with 81,000 3D samples, generated in 28.2 seconds. (right) spectrum result of our method when the Gamito and Maddock's method is used for generation of initial sample patch. Here, $N_x = 15,600$ and the total number of samples N_z is around 81,000. The time to generate initial sample patch is 5.6 seconds, and the time for our method to generate the final samples is 6.35 milliseconds. As in the 2D case, our method has comparable quality but is much faster.*

sity sets. This shows that we can use any method as the input to our algorithm and still enjoy a significant speed up.

We also studied the performance of our method for the generation of 3D samples. Fig. 10 shows the spectrum comparison and Fig. 11 shows the timing. Our result using the Gamito and Maddock [GM09] method for generating the initial patch is comparable to their full result. Moreover, the CPU and GPU implementations of our method can generate around 9.1M and 17.3M 3D samples/second.

Since the purpose of generating blue noise point sets is to use them in applications such as Monte Carlo integration, we also conducted experiments to investigate the performance of our method for antialiasing as shown in Figs. 12 to 14. We begin by verifying that our method's error only depends on the number of samples in the initial patch. To do this, we rendered a checkerboard scene with approximately 4 samples/pixel at three different resolutions (thereby changing the total number of samples for each image). Fig. 12 shows the

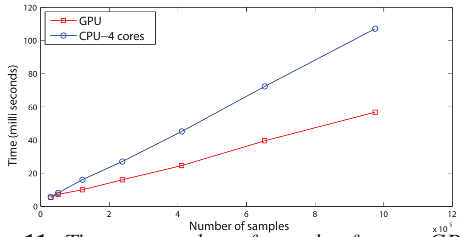

Figure 11: *Time vs. number of samples for our GPU and CPU implementations for generation of samples in 3D. $N_x = 15,600$ in this experiment.*

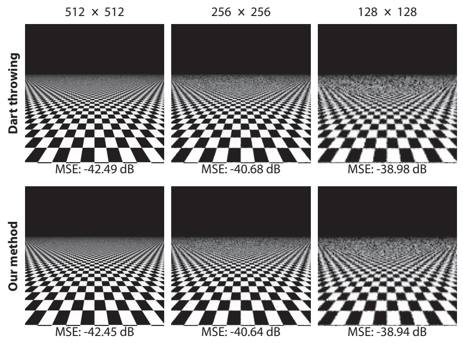

Figure 12: *These scenes are all rendered with approximately 4 samples/pixel, but their sizes are different so they have a different total number of samples. Here $N_x = 630$. All the MSE values are obtained with respect to the groundtruth image rendered with 2048 samples per pixel and are in dB. Since they are all negative, values with larger magnitude are better. Our algorithm has a constant error with respect to dart throwing which verifies our analysis in Section 2.*

error with respect to dart throwing does not depend on the total number of samples and is constant.

Fig. 13 and 14 show the results for sampling the zone plate test pattern and rendering a checkerboard scene using different methods. The top row in both figures show the results when *all* the samples are generated using an existing methods while the bottom row shows the result of using these methods only to generate the initial patch and then using our algorithm to generate the final point set. We can see subjectively and objectively that the quality is comparable and the error is small.

The time complexity of our method is $O(n^2 N_z)$ where the n^2 factor is the complexity of rotation. Note that the more accurate complexity is $O(n^2 N_z + \sqrt{N_z} n^5)$ where the second term is the complexity for multiplication of $n(n-1)/2$ n-dimensional rotation matrix for each block. Therefore by assuming $\sqrt{N_z}$ to be larger than n^3 which is a correct assumption in practice, we can neglect the second term in complexity. Also the space complexity of our method is $O(nN_z)$ which is needed for writing N_z n-dimensional samples.

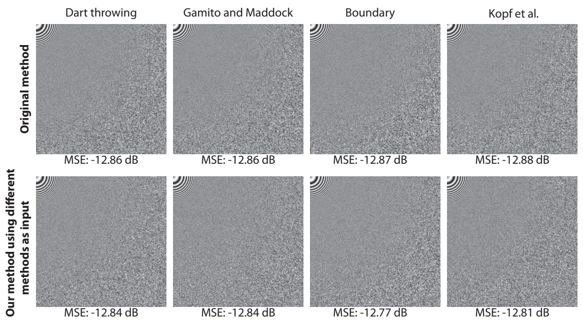

Figure 13: *This zone plate test pattern is sampled with approximately 1 sample/pixel and smoothed with a 3 pixel-wide Gaussian filter. MSEs are obtained with respect to the reference zone plate image with 10,000 samples per pixel. Here, our method uses ($N_x = 630$). Although our algorithm is faster than any other method, we see here the error introduced is small.*

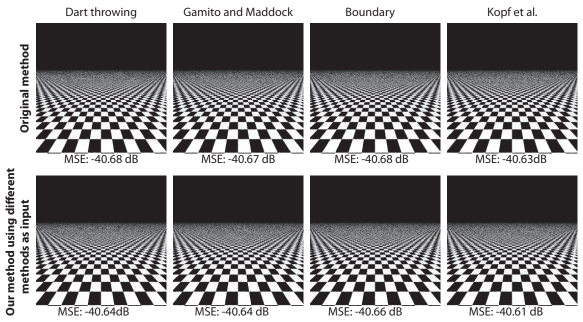

Figure 14: *In all cases the images are of size 256×256 and the scene is rendered with approximately 4 samples/pixel. Again $N_x = 630$ and the error of our method is small.*

4. Limitations and Conclusion

Although our algorithm is intended to produce samples suitable for Monte Carlo integration, it cannot be used in some applications like object placement, because of close samples at block boundaries. We leave the investigation of how to address this for future work. Moreover, our method is not able to directly do variable-density sampling, which can also be investigated in the future.

We have presented a fast algorithm for generating blue noise samples which is linear in time and space. We use a small patch of samples generated with any existing methods and replicate it to generate a large set of blue noise samples. Since the operations on each sample are independent of the other samples, our method is parallelizable. We demonstrate a GPU implementation of our method which is significantly faster than existing approaches.

Acknowledgements

The implementations for the algorithms of Gamito et al. [GM09], Dunbar and Humphreys [DH06], and Kopf et al. [KCODL06] were provided by the respective authors. This work was supported by the National Science Foundation under the NSF CAREER award #0845396.

References

[Coo86] Cook R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5* (January 1986), 51–72. 1

[CSHD03] Cohen M. F., Shade J., Hiller S., Deussen O.: Wang tiles for image and texture generation. *ACM Trans. Graph. 22* (July 2003), 287–294. 1

[DH06] Dunbar D., Humphreys G.: A spatial data structure for fast Poisson-disk sample generation. *ACM Trans. Graph. 25* (July 2006), 503–508. 1, 4, 5, 7

[DW85] Dippé M. A. Z., Wold E. H.: Antialiasing through stochastic sampling. *SIGGRAPH Comput. Graph. 19* (July 1985), 69–78. 1, 2

[EDP*11] Ebeida M. S., Davidson A. A., Patney A., Knupp P. M., Mitchell S. A., Owens J. D.: Efficient maximal poisson-disk sampling. *ACM Trans. Graph. 30*, 4 (Aug. 2011), 49:1–49:12. 1

[GM09] Gamito M. N., Maddock S. C.: Accurate multidimensional Poisson-disk sampling. *ACM Trans. Graph. 29* (December 2009), 8:1–8:19. 1, 4, 6, 7

[HDK01] Hiller S., Deussen O., Keller A.: Tiled blue noise samples. In *Proceedings of the Vision Modeling and Visualization Conference 2001* (2001), VMV '01, pp. 265–272. 1

[Jon06] Jones T. R.: Efficient generation of Poisson-disk sampling patterns. *journal of graphics, gpu, and game tools 11*, 2 (2006), 27–36. 1

[KCODL06] Kopf J., Cohen-Or D., Deussen O., Lischinski D.: Recursive wang tiles for real-time blue noise. *ACM Trans. Graph. 25* (July 2006), 509–518. 1, 4, 5, 6, 7

[KS11] Kalantari N. K., Sen P.: Efficient computation of blue noise point sets through importance sampling. *Computer Graphics Forum 30*, 4 (2011), 1215–1221. 1

[LD05a] Lagae A., Dutré P.: A procedural object distribution function. *ACM Trans. Graph. 24* (October 2005), 1442–1461. 1, 2

[LD05b] Lagae A., Dutré P.: *Template Poisson Disk Tiles*. Report CW 413, Departement Computerwetenschappen, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium, May 2005. 1, 2, 4, 5

[LD08] Lagae A., Dutré P.: A Comparison of Methods for Generating Poisson Disk Distributions. *Computer Graphics Forum 27*, 1 (2008), 114–129. 1, 5

[MF92] McCool M., Fiume E.: Hierarchical Poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92* (1992), Morgan Kaufmann Publishers Inc., pp. 94–105. 1

[ODJ04] Ostromoukhov V., Donohue C., Jodoin P.-M.: Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23* (August 2004), 488–495. 1

[Ost07] Ostromoukhov V.: Sampling with polyominoes. In *ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 1

[WCE07] White K. B., Cline D., Egbert P. K.: Poisson disk point sets by hierarchical dart throwing. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing* (2007), IEEE Computer Society, pp. 129–132. 1

[Wei08] Wei L.-Y.: Parallel Poisson disk sampling. *ACM Trans. Graph. 27* (August 2008), 20:1–20:9. 1, 5