

Deterministic Blue Noise Sampling by Solving Largest Empty Circle Problems

Yoshihiro KANAMORI[†], Zoltan SZEGO^{††}, Tomoyuki NISHITA^{††} (Member)

[†] University of Tsukuba

^{††} The University of Tokyo

Summary Sampling patterns with a blue noise distribution are widely used in many areas of computer graphics, yet their efficient generation remains a difficult problem. We propose a method to generate point sets with a blue noise distribution using a deterministic algorithm with no preprocessing. We insert each new sample at the center of the largest empty circle in the point set, which is obtained by calculating the Delaunay-triangulation of the set and finding the triangle with the largest circumcircle. Our method supports adaptive sampling according to a user-specified density function, as well as specifying the exact number of required samples. It can also be extended to perform sampling on a three-dimensional curved surface.

Key words: Blue noise sampling, Delaunay triangulation, halftoning

1. Introduction

Sampling is a fundamental part of various graphics applications such as image processing or rendering. With the number of samples allowed often being limited in these applications, getting the best-quality results out of as few samples as possible is essential. For this purpose, one of the most often used sampling patterns are Poisson Disk patterns, defined by a minimum distance between all neighboring sample points. These have the spectral properties of blue noise, defined as having a spectrum with low energy in the low frequencies. This property has been shown to be very effective at providing good sampling with a low amount of noise or aliasing, as evidenced by the fact that it also occurs naturally, for example, in the placement of photoreceptor cells in the eye¹⁾.

This paper presents a novel method that produces point sets deterministically, with high-quality blue noise distributions without any precomputation, using a geometrically-based algorithm to determine the exact position for each new sample. While slower

than some of the alternatives, the proposed method results in patterns with better spectral properties. The proposed method supports adaptive sampling with a simple modification to the algorithm. Furthermore, it can be naturally extended to handle arbitrary sampling domains, including volumes and three-dimensional curved surfaces.

2. Related Work

In this section, we first introduce the basic framework for evaluating the quality of a point set based on its spectral properties, and then describe the two categories of methods for generating point sets with blue noise properties: methods based on dart throwing and tile-based methods.

2.1 Quality evaluation framework

Here we briefly explain how the quality of a blue noise distributed point set is evaluated, as detailed in an in-depth survey⁵⁾. We assume the samples are generated on a two-dimensional domain $[0, 1]^2$, and consider N point samples x_i as a signal given by the sum of Dirac-deltas (δ) located at the position of each

point:

$$\sum_{i=1}^N \delta(x - x_i). \quad (1)$$

Taking the Fourier-transform of this signal gives the spectrum for a single point set, and averaging many of these gives the characteristic power spectrum $P(f)$ for the method used, with f being the frequency. The typical blue noise spectrum has a low-energy low-frequency band surrounding the center, followed by a sharp transition to higher energies, finally flattening out for high frequencies. The spectrum is radially symmetric, therefore it can be represented by a one-dimensional *radially averaged* power spectrum by averaging $P(f)$ in concentric rings, each corresponding to a frequency:

$$P_r(f_r) = \frac{1}{N(f_r)} \sum_{i=1}^{N(f_r)} P(C(i)), \quad (2)$$

where $N(f_r)$ is the number of samples that are taken from the spectrum in each ring with radius f_r . $C(i)$ gives the position for the i^{th} sample in the ring. The variance of the power for each frequency, $s^2(f_r)$, is used to derive the *anisotropy* of the spectrum:

$$A_r(f_r) = \frac{s^2(f_r)}{P_r^2(f_r)}, \quad (3)$$

where

$$s^2(f_r) = \frac{1}{N(f_r)} \sum_{i=1}^{N(f_r)} (P(C(i)) - P_r(f_r))^2. \quad (4)$$

High-quality blue noise spectra are characterized by a wide central low-energy ring and low anisotropy (around -10dB).

2.2 Methods based on dart throwing

Dart throwing, introduced by Cook²⁾, generates Poisson disk distributions by first generating uniformly distributed points, then rejecting points that are too close to each other given a minimum separation distance. While easy to implement, this algorithm is slow and difficult to control because a minimum distance needs to be given instead of the required number of samples. Dart throwing has been optimized and extended in a number of ways, one of which is called relaxation dart throwing⁷⁾, where the minimum separation distance starts out large and is gradually reduced, producing hierarchical samples somewhat faster than traditional dart throwing. A method called Lloyd's relaxation⁶⁾ is often applied to these results as a post-process, which is a rather

costly iterative process that constructs the Voronoi-diagram for the entire point set repeatedly.

In 2008, Wei¹⁰⁾ proposed a method to generate blue noise distributed samples at high speed on programmable GPUs based on the observation that areas of the sampling domain that are sufficiently far apart do not affect each other, and can therefore be handled independently in parallel. The method produces good quality results at high speed thanks to the high performance provided by GPUs; however, it is difficult to control the exact number of produced samples due to the random nature of the algorithm. Our method can deterministically generate any given number of samples, which is useful for several applications including object placement. Additionally, while extending Wei's method to non-Euclidean domains is not trivial, our method can naturally handle sampling on spherical or arbitrary polygonal surfaces.

2.3 Tile-based methods

Methods based on various tiling schemes use a different approach than dart throwing. Instead of relying on randomness, they generally precalculate distributions for a given number of predefined tiles, and then arrange those on the sampling domain, allowing for better performance at the expense of quality and precomputation time.

For example, Wang tiles are used by various methods. Hiller *et al.*³⁾ introduced the use of Wang tiles for generating a point set. However, the power spectrum of the method is of low quality. Kopf *et al.*⁴⁾ used Wang tiles recursively to produce hierarchical sampling patterns at high speed from a large precomputed dataset, but the spectrum still has unwanted spikes and noise.

Ostromoukhov⁹⁾ proposed a different tiling scheme based on hierarchically subdivided polyominoes. The method produces reasonably good results at high speed, however, it still requires a rather complicated and expensive precomputing step. The variety and randomness of the results are limited by the number of pregenerated tiling variations.

Our method produces point sets with high-quality spectra, and is not limited in the variety of outcomes by any kind of precomputed data. Note that our method does not fit into the two categories presented in Sections 2.3 and 2.2, it should instead be classi-

fied as a geometrically-based method that selects the exact position for each sample point.

3. Proposed Algorithm

In this section, we present the algorithm used to generate blue noise distributed point samples in detail, going into specifics for both the two-dimensional and three-dimensional cases.

3.1 Overview

Our method generates samples deterministically and sequentially. Each new sample is placed at the center of the largest empty circle in the already existing set of samples. The intuitive reasoning for the algorithm used is that given a set of samples, the next one should go in the most sparsely sampled area so far, and should be equidistant from the other samples surrounding it. Finding the largest empty circle⁸⁾ is equivalent to finding the triangle with the largest circumcircle in the *Delaunay triangulation* of the point set, a triangulation which guarantees by definition that no other point lies within any triangle’s circumcircle. The process is repeated, gradually filling up the sampling domain with points.

Initially, the input point set consists of a few seed points, which serve as a means of providing deterministic control over the pseudorandom outcome of the process. The procedure to generate new points is as follows:

```

GeneratePoints(N, Points):
  // N : the total number of samples required
  // Points : the vector of sample points
  // (initialized with a small arbitrary set of seed points)
  Delaunay := InitializeDelaunayTriangulation(Points);
  while (Points.length < N)
    C := Delaunay.findLargestCircumcircle();
    Points.append(C.center);
    Delaunay.insertPointAndUpdateTriangulation(C.center)
  end

```

See Figure 1 for a graphical overview of one step in the inner loop of the algorithm described above.

In order to find the largest circumcircle quickly, in addition to the Delaunay triangulation, we maintain a heap data structure containing all of the triangles, sorted by the radius of their circumcircle. The largest triangle can therefore be obtained in constant time, and updates to the triangulation (such as inserting and modifying triangles) happen in $O(\log N)$. Trian-

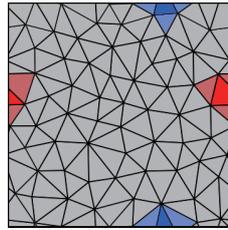


Fig. 2 Illustration of a completely triangulated toroidal domain. A few triangles and their neighbors are highlighted to emphasize the wrap-around effect.

gles with a circumcircle whose center lies outside of the domain $[0, 1)^2$ are excluded from the search.

3.2 Using a toroidal domain

Using the above algorithm as-is results in a point set whose Delaunay triangulation is constrained by the boundaries of the 2D domain $[0, 1)^2$. When a triangle’s circumcenter lies outside this domain, it is ignored and the next largest triangle is used instead. As a consequence, elongated triangles tend to cluster up near the edges, resulting in undesirable gaps around boundaries.

To alleviate this problem, the sampling domain can be changed from a simple two-dimensional domain to a toroidal domain. That is, the opposite boundaries of the domain “wrap around”, the top boundary being connected to the bottom boundary, and similarly for the left and right boundaries, making the domain topologically equivalent to the surface of a torus (Figure 2).

Once the domain wraps around, there are no more “boundaries” to speak of, and therefore the elongated triangles also disappear, resulting in much more balanced shapes for all generated triangles. Such well-shaped triangles allow us to omit very time-consuming searches; in the case of an unbalanced triangle, the circumcenter tends to lie outside the triangle in question, and thus, when inserting a new point, we must specify which triangles should be updated by searching for the triangle that the new point belongs to. In contrast, well-shaped triangles contains their own circumcenters in a large majority of cases (99% of all triangles in our experiments), and thus we can find the triangle containing the point to be inserted in $O(1)$. In case that a circumcenter lies slightly outside the triangle in question, we clamp the point to

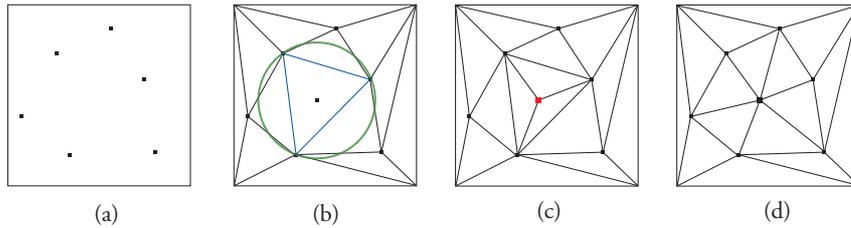


Fig. 1 Overview of a single step of the proposed algorithm. In order: (a) an initial set of points, (b) their Delaunay triangulation with the largest empty circle shown, (c) the new sample point immediately after being added, and (d) the updated triangulation. Note that in this example, only circles with centers within the area bounds were considered. For a more correct approach, see Section 3.2.

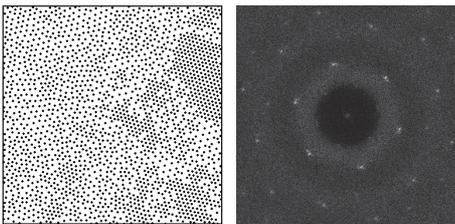


Fig. 3 Example of the appearance of the undesirable regular patterns mentioned in Section 3.3. The pattern in the generated points (left) causes hexagonally arranged spikes in the spectrum (right).

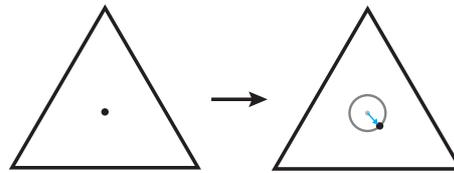


Fig. 4 Adjusting the position of a sample point inside an equilateral triangle.

be inserted within the triangle.

3.3 Avoiding regular patterns

While triangles with elongated edges cause loss of sampling quality and speed, triangles that are too regular and close to equilateral are also undesirable. During the execution of the proposed algorithm, it produces results that are often uneven, with noticeable large areas of regular patterns that show up as spikes on the power spectrum. See Figure 3 for an example.

The reason these patterns emerge is because of the way the algorithm behaves for equilateral triangles. If the Delaunay triangulation happens to include a cluster of equilateral triangles sometime during the process, and one of them is selected as the one with the largest empty circle, the new point will form smaller equilateral triangles, with further subdivision of the area only preserving the pattern. The result is a regular triangular grid, obviously unsuitable for blue noise sampling. To prevent this, an additional step is included in the algorithm: when inserting a new point into the Delaunay triangulation, we first check if the

triangle containing the point is close to being equilateral up to a certain threshold. If it is, the new point is first offset by a small amount $(\delta x, \delta y)$, such that it stays within its triangle, in a pseudorandom direction depending only on the point's original position, in this case, to keep our algorithm deterministic (see Figure 4);

$$\theta = (x + y) C\pi, \quad \delta x = d \cos(\theta), \quad \delta y = d \sin(\theta),$$

where x and y are the original coordinates, C is an arbitrary constant and d is the distance to move the point. For the distance, we used 0.2 times the triangle's edge length (determined empirically). This modification eliminates the formation of regular patterns and fixes the occasional spikes in the power spectrum.

3.4 Adaptive sampling

With a simple modification, the proposed method can also perform adaptive sampling based on a user-specified importance function $f : [0, 1]^2 \rightarrow [0, 1]$, that specifies how dense the samples should be in an area. For all of the triangles in the heap data structure mentioned in Section 3.1, the radius of the circumcircle that acts as the sort key is weighted by a sample from the importance function, taken at the center of the circumcircle. This way, triangles that lie in areas requiring more thorough sampling are given more pri-

ority, resulting in sample points that are distributed according to the importance function.

3.5 Sampling on a 3D surface

Our method can be extended to generate samples on three-dimensional surfaces. The basic process is mostly the same: starting from a sparse set of points and their points are added in triangles with the largest circumcircle. The new points are projected onto the surface and the triangulation is updated.

In case the surface is specified by a complex polygonal mesh, the initial set of points and their connectivity can be obtained with a mesh simplification method such as edge decimation. Adaptive sampling is also possible if an importance function is defined on the surface.

3.6 Improving performance by parallelization

In order to further improve the performance of the proposed method, we can parallelize the process using modern multicore CPUs. One observation when testing our algorithm was that the updating of the Delaunay property generally does not propagate very far in the triangulation. In fact, the number of triangle flips after a point insertion was always less than 10. In other words, when the number of points is sufficiently large, individual steps of the algorithm operate fairly locally. A similar observation is also exploited by Wei¹⁰⁾ in his highly parallelized sample generation technique. We can also observe that the performance of our algorithm decreases as the number of generated points get bigger, due to the $O(N \log N)$ runtime of repeatedly updating a heap for N iterations.

We therefore employ a divide-and-conquer strategy, by cutting up the original $[0, 1]^2$ domain into subregions. Theoretically, if we divide the region into K subregions, the cost becomes $O(K \times N/K \log N/K) = O(N \log N/K)$, which is smaller than $O(N \log N)$. Note that no guarantees are made about the Poisson Disk property of samples at the edges of neighboring subregions this way, since the subtasks do not take constraints due to other regions into account. However, we found that in the actual results the borders are barely noticeable, and the resulting spectra are indistinguishable from the single-threaded version. For details on the scaling achievable, see Section 4.

Table 1 Time to generate a given number of samples (msec)

Method	Number of samples		
	20,000	50,000	100,000
Recursive Wang tiles	0.59	1.35	2.66
Bridson <i>et al.</i> 's dart throwing	165	420	829
Proposed method (1 thread)	132	378	853
Proposed method (4 threads)	92	241	502

Table 2 Scaling by parallelization for 100,000 samples

Subdivision	Time	Scaling
none	853 ms	1.00×
2 × 2	559 ms	1.52×
4 × 4	502 ms	1.69×
8 × 8	502 ms	1.69×

4. Results

We implemented a prototype system that uses our algorithm in C++. The environment used for our experiments was a standard desktop PC with an Intel Core 2 Quad Q6700 CPU at 2.66GHz and 2GB of RAM. We used the Intel Compiler version 10.1 and the OpenMP extension for parallelization.

We compared the performance and quality of the proposed method to two other related techniques, Recursive Wang tiles⁴⁾ and an optimized variation of the dart throwing algorithm utilized by Bridson *et al.*¹⁾ We performed experiments for a varying number of samples, namely, uniform sampling with 20,000, 50,000 and 100,000 samples generated. Table 1 compares the speed measurements for each technique. Figures 5 and 6 show the spectra for each method, as well as the radially averaged energy and anisotropy graphs, for 20,000 and 50,000 samples, respectively.

Performance-wise, Recursive Wang tiles performed several orders of magnitude faster than the alternatives, however, it did so at the expense of quality. The larger the number of samples required, the noisier the output becomes, as is evident from the spike patterns and noise in the spectrum and anisotropy graphs. Our method, while having a speed faster than the optimized dart throwing variation, produced slightly better quality results, based on the wider inner ring of the power spectrum.

Table 2 shows the scaling achieved by parallelization. The scaling does not increase as the theory due

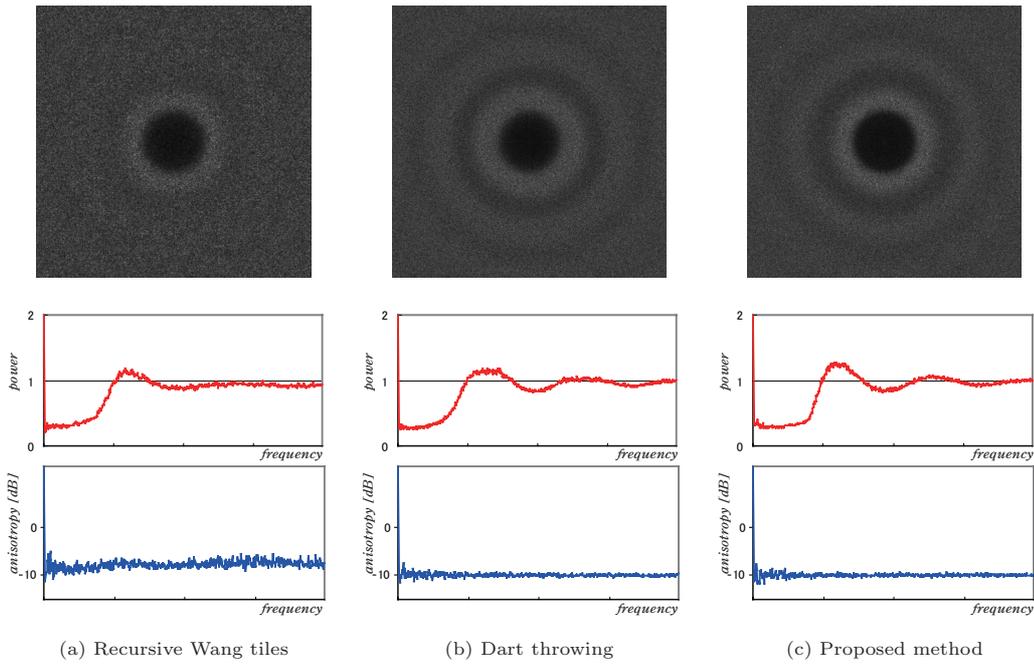


Fig. 5 Spectral quality comparison chart for 20,000 generated samples.

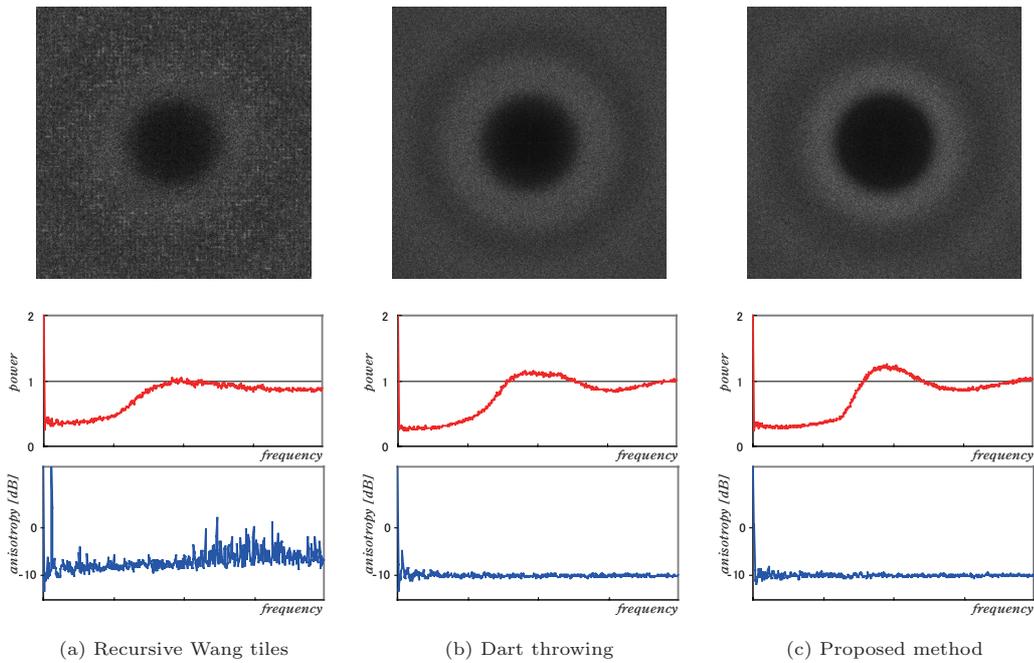


Fig. 6 Spectral quality comparison chart for 50,000 generated samples.

to the overheads of the process. Because the subdivision slightly causes quality loss at boundaries, we conclude 4×4 subdivision is the optimal.

Figure 7 shows the examples of adaptive sampling on a 2D domain, demonstrated by point-based

halftoning of a grayscale input image.

Figure 8 shows the results of producing samples using our method on spherical HDR light maps. The importance function was given by converting the color of each pixel into a grayscale value and biasing the

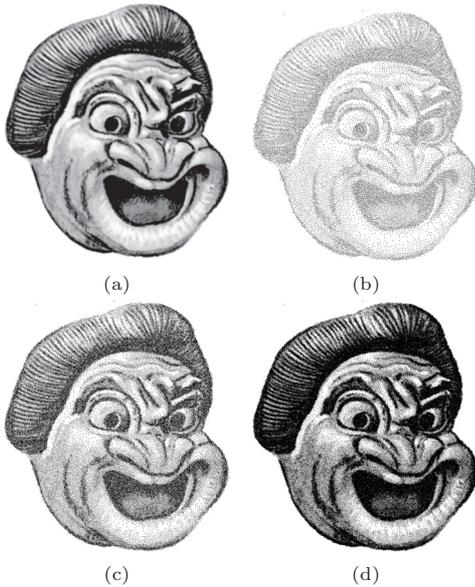


Fig. 7 Results of halftoning the image of a mask. (a) The grayscale original. (b)–(d) Halftoned results created with the proposed method consisting of 20,000, 50,000 and 100,000 points, respectively.

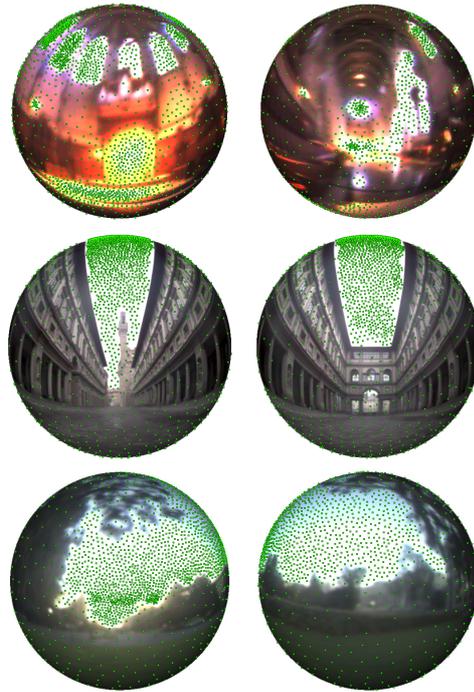


Fig. 8 Results of adaptively sampling a variety of spherical HDR radiance environment maps using our method. In each case, 3,000 samples are placed on the unit sphere, displayed as dots.

generated samples toward bright regions only.

Finally, Figure 9 demonstrates that our method can be extended to three dimension via Delaunay tetrahedralization. The three-dimensional spectrum shows the samples exhibit blue noise property.

5. Conclusion and Future Work

We described a method to generate sampling patterns with a high quality blue noise distribution. Our method is deterministic, produces the exact number of required samples, requires no precomputation, and handles adaptive sampling based on a user-specified importance function. Furthermore, our method can be easily extended higher dimensions than two thanks to the theoretical simplicity. For future work, we would like to gain further speed up of our method using CUDA.

References

- 1) Robert Bridson, Jim Houriham, and Marcus Nordensam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3):46, 2007.
- 2) Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.
- 3) Stefan Hiller, Oliver Deussen, and Alexander Keller. Tiled blue noise samples. In *VMV '01: Proceedings of the*

- Vision Modeling and Visualization Conference 2001*, pages 265–272, 2001.
- 4) Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 509–518, New York, NY, USA, 2006.
- 5) Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. Report CW 459, Department of Computer Science, K.U.Leuven, Leuven, Belgium, August 2006.
- 6) S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.
- 7) Michael McCool and Eugene Fiume. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, pages 94–105, San Francisco, CA, USA, 1992.
- 8) Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 1992.
- 9) Victor Ostromoukhov. Sampling with polyominoes. *ACM Transactions on Graphics*, 26(3):78, 2007.
- 10) Li-Yi Wei. Parallel poisson disk sampling. *ACM Transactions on Graphics*, 27(3):1–9, 2008.
- 11) J. I. Yellott Jr. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, 221(4608):382–385, 1983.

(Submitted July 16, 2010)

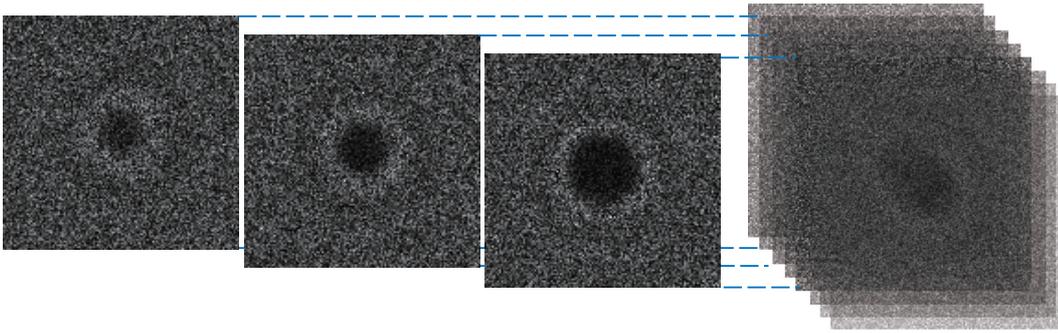


Fig. 9 The spectrum of 10,000 3D samples in the unit cube. The three-dimensional spectrum is shown as a series of 2D slices. The low-energy spherical region in the middle indicates a blue noise distribution.

(Received November 5, 2010)