Efficient Generation of Poisson-Disk Sampling Patterns

Thouis R. Jones

April 24, 2006

Abstract

Poisson Disk sampling patterns are of interest to the graphics community because their blue-noise properties are desirable in sampling patterns for rendering, illumination, and other computations. Until now, such patterns could only be generated by slow methods with poor convergence, or could only be approximated by jittering individual samples or tiling sets of samples.

We present a simple and efficient randomized algorithm for generating true Poisson Disk sampling patterns in a square domain, given a minimum radius R between samples. The algorithm runs in $O(N \log N)$ time for a pattern of N points. The method is based on the Voronoi diagram. Source code is available online.

1 Introduction

Much of computer graphics is concerned directly or indirectly with sampling. At the most basic level, the location of point samples and how they are combined is fundamental to many rendering tasks. Accordingly, much effort has been expended in the exploration of different sampling strategies, primarily in the area of generating two-dimensional sampling patterns.

Sampling patterns are often characterized by their behavior in frequency space. For many tasks, sampling patterns with a blue noise Fourier spectrum are preferred. Blue noise spectra in two dimensions are characterized by a concentration of energy beyond some radius from the origin in frequency space. It is also desirable that the pattern have low radial anisotropy in frequency space (see figure 6). One way to produce patterns having these properties is by using a Poisson Disk process to generate points ([Yel83, Coo86, Mit87]).

We present a randomized algorithm for generating Poisson Disk patterns, given the minimum separation between points. The algorithm runs in $O(N \log N)$ time where N is the number of points in the resulting pattern. The patterns produced are complete, in the sense that no further points could be added. Our method makes use of Delaunay triangulations along with two common data structures, a balanced binary tree and a hash table. Although not as straightforward as the naïve algorithm, our algorithm can be easily implemented using off-the-shelf and widely available software libraries. Source code is available online at the address listed at the end of this paper.

2 Background

The Poisson Disk distribution can be defined as the limit of a uniform sampling process with a minimum-distance rejection criterion. Successive points are independently drawn from the uniform distribution on $[0, 1]^2$. If a point is at least distance R from all points in the set of accepted points, it is added to that set. Otherwise, it is rejected. The choice of R controls the minimum allowable distance between points and, indirectly, the density of the Poisson Disk pattern.

A direct implementation of the description above (also known as "dart-throwing", [Mit87, MF92]) is inefficient. This is in part because of the expense of testing each new sample against all samples. More importantly, as more samples pass the minimum distance test, the open area in $[0, 1]^2$ from which samples could be accepted becomes arbitrarily small. Similarly, the probability of a uniform sample landing in this open area drops, and an increasingly large number of samples from the uniform distribution must be generated and tested to find one that passes the minimum distance criterion. Moreover, it is unclear when to stop trying to generate samples, i.e., there is no obvious stopping condition. An $O(N^2)$ method for generating samples that gives a distribution empirically similar to the Poisson Disk distribution was introduced by Mitchell [Mit91], as well as a discussion of why patterns with blue-noise properties are preferred for sampling in computer graphics.

Relaxation methods have also been proposed. Lloyd's relaxation [Llo83] generates points uniformly at random and then repeatedly moves them to their Voronoi regions' centroids. The convergence rate of this method is also quite poor, though better than dart throwing. This method trades off isotropy of the final pattern for speed [HDK01].

Recently, a method based on aperiodic tiling was proposed by Ostromoukhov et al. [ODJ04]. This method is able to produce sets of samples with blue-noise properties very quickly, but the algorithm is somewhat complex to understand and implement.

This paper demonstrates an algorithm for creating true (non-approximate) Poisson Disk sampling patterns efficiently (in $O(N \log N)$ time and linear storage). The algorithm is conceptually simple, and is straightforward to implement. Its implementation can make use of widely available libraries for computing Delaunay triangulations.

```
Given: R

Initialize D = Delaunay triangulation with single point
Initialize W = Weighted tree of Voronoi regions

while W contains free space:
    1: Search W for Voronoi region V containing free space
    2: Generate new point p in free region of V
    3: Insert p in D
    4: Compute Voronoi region of p and insert in W
    For each neighbor q of p in D:
        5: Recompute Voronoi region of q and update W
```

Figure 1: Pseudo-code for our algorithm. Here “free” means that the Voronoi region of a point is not entirely covered by a disk of radius R.

3 Method

Our method can be described concisely in pseudo-code, given in figure 1. Each step in the algorithms takes $O(\log N)$ time with high probability (*w.h.p.*), and the entire algorithm terminates in $O(N \log N)$ time, *w.h.p.* We will explore each step below.

Our algorithm maintains a Delaunay triangulation D of the generated sample points, updating it incrementally as points are added. For each point p in D, our algorithm also computes its Voronoi region V,, along with the total “free” area in V that is more than R from p. Points and their free areas are stored in a weighted tree, W. The weighted tree W stores at each node the sum of the weights in its left and right branches. Terminal nodes of the tree correspond to vertices in the Delaunay triangulation, and are weighted by the free area of their corresponding vertices' Voronoi regions. See Figures 2 and 3.

Finding a Voronoi region V with free space in the tree (step 1) is done by randomly traversing the tree from the root with each left/right decision weighted according to the total weight in the two subtrees. The tree has depth $O(\log N)$ *w.h.p.* (see the discussion of step 4 below and Figure 2), so this step is $O(\log N)$.

We discuss the generation of a new point in the free region of V (step 2) in Section 3.1, below, noting only that it can be done in $O(\log N)$ *w.h.p.*

The code to maintain and update the Delaunay triangulation (step 3) can be any of several incremental construction algorithms. The only requirement is that the time to add a new point be $O(\log N)$ (amortized, if necessary). Given foreknowledge of the closest point in the Delaunay triangulation (in our case, the central point of V), the time to insert a new point is $O(1)$ in a randomized incremental construction [LDM92]. Note that this implies the number of Delaunay neighbors of the new point is $O(1)$, amortized.

Computation of the Voronoi regions and their free space for the new point

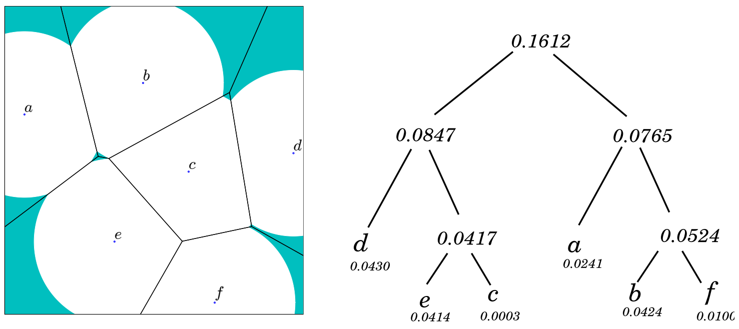

Figure 2: Voronoi diagram of a group of points, and their weighted tree W. Weights for the leaf nodes correspond to the free (shaded) area in the Voronoi diagram (see Figure 3), and the interior nodes store the total weight of their children. The points are inserted into the tree randomly (without regard to weights). When creating a new point in the Poisson-disk pattern, the tree is traversed randomly, with probabilities proportional to subtree weight, to create a uniform sampling of the free area.

and its neighbors in D (steps 4 and 5) is effectively $O(1)$ because the amount of work in updating the Voronoi cells is proportional to the amount of work to update the Delaunay triangulation. (The Delaunay neighbors of the inserted point are the only ones whose Voronoi regions change when the new point is inserted.)

Inserting the new point into the tree W (step 4) is done by randomly traversing the tree with equal probability of taking the left or right branch, until the chosen branch is empty. The new point is inserted in W to fill the missing branch. This keeps the tree of depth $O(\log N)$ *w.h.p.*, by an N-balls-in-N-bins argument [MR95], as follows. At a depth of $\log_2 N$ in the tree, there are N nodes. A new node inserted into the tree randomly will end up parented by one of those nodes with equal probability. By the standard argument, *w.h.p.* there is no node at the $\log_2 N$ level with more than $O(\log N)$ children.

Updating the tree to reflect the changes in area of the Delaunay neighbors of the new point (step 5) takes $O(\log N)$ time, as each neighbor must update its parents in the tree to reflect the change in free area, recursively. In order to make such updates straightforward, in our implementation we maintain parent pointers in the tree and a hash table from points in the Delaunay triangulation to nodes in the tree[1]

[1]It would be feasible to remove the hash table by using a balanced-binary tree ordered by a random per-point index.

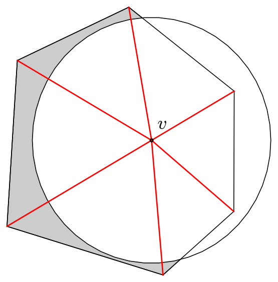

Figure 3: The Voronoi region for a point v. The circle shows the radius by which points must be separated in the Poisson Disk pattern. New points would be allowed in the shaded area, which we call the free space for the Voronoi region. The point v would receive weight in the binary tree W proportional to the shaded area. The area is calculated by decomposing the Voronoi area into a triangle fan and computing the shaded area in each triangle.

3.1 Generation of a new point

All of the described steps thus far are $O(\log N)$. It remains to be shown that a new point in the free region of V can be generated in $O(\log N)$ time. This is the key insight behind our algorithm.

We treat the Voronoi region of a point as a triangle fan centered at that point. See figure 3. We first select one of the triangles in the fan at random weighted by their respective free areas. Our strategy for uniformly generating a random point in the triangle's free area relies on a case analysis. See figure 4. Label the vertices of the triangle v_c, v_1, v_2 with v_c the central point of the Voronoi region. We refer the reader to figure 4.

Case 1: One of v_1, v_2 is within R of v_c. In this case, we compute the intersection v_i of $\overline{v_1 v_2}$ and the circle centered at v_c. The triangle formed by v_c, v_i and the farther of v_1, v_2 falls into case 4 below.

Case 2: $\overline{v_1 v_2}$ intersects the circle of radius R centered at v_c in two places. In this case, both intersections are computed, and two triangles that fall into case 4 occur.

Case 3: The point of closest approach on $\overline{v_1 v_2}$ to v_c is interior to $\overline{v_1 v_2}$. In this case, the point of closest approach is computed, and two triangles that fall into case 4 are formed.

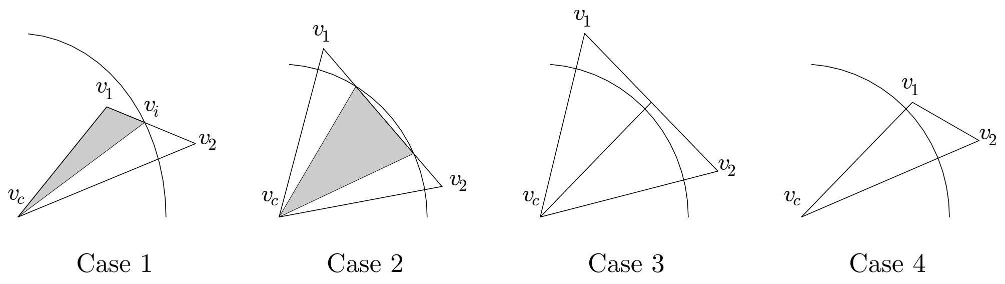

Figure 4: The four cases for a triangle in the Voronoi region. Cases 1, 2, and 3 all reduce to case 4 after subdivision. Shaded areas are trivial rejections after subdivision.

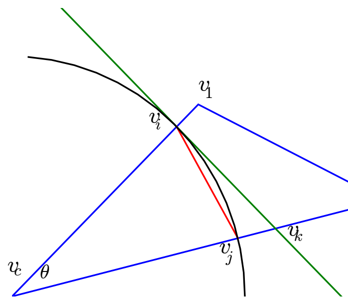

Figure 5: Sampling from the quadrilateral $\overline{v_1 v_2 v_j v_i}$ has a better than $\frac{1}{2}$ chance of being outside the circle centered at v_c. The line through v_i, v_k is the tangent at the intersection point v_i. v_1 is the point of closest approach to v_c on $\overline{v_1 v_2}$.

Case 4: The point of closest approach on $\overline{v_1 v_2}$ is one of v_1, v_2. In this case, the intersections v_i, v_j of $\overline{v_c v_1}$ and $\overline{v_c v_2}$ with the circle are computed. A random point is repeatedly generated uniformly within the quadrilateral formed by these intersections and v_1, v_2, until such a point is at least R away from v_c. See also figure 5.

We must now show that case 4 terminates in $O(\log N)$ time with high probability. It suffices to show that the probability that a random point uniformly drawn from the quadrilateral is at least R distance from v_c is $> \frac{1}{2}$.

We refer to the labels in figure 5. If a point is drawn uniformly from the triangle defined by v_i, v_j, v_k, the probability of it being more than R away from v_c is at least $\frac{2}{3}$, by the following argument. This probability is the ratio of the area outside the arc $\widehat{v_i v_j}$ to that of the triangle v_i, v_j, v_k. Straightforward trigonometry shows that this ratio is $\frac{\tan\theta - \theta}{\tan\theta - \sin\theta}$, which is $\frac{2}{3}$ when θ approaches zero, and grows to unity at $\theta = \frac{\pi}{2}$.

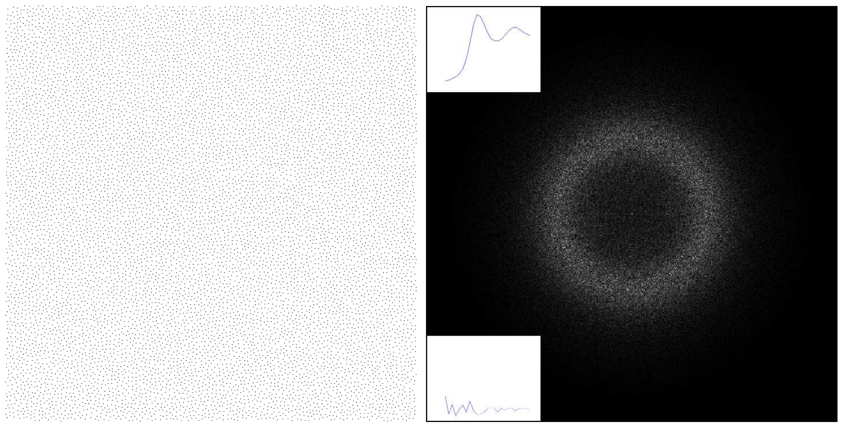

Figure 6: A set of points generated by our algorithm, and their Fourier transform. The spectrum reveals the typical blue noise properties, with an annulus of low energy around the origin. The insets show mean radial power and radial anisotropy, respectively.

Thus, case 4 above terminates in $O(\log N)$ time *w.h.p.*, since the probability of requiring more than $\log_2 N$ trials is less than $\frac{1}{N}$.

4 Results

A set of points output from our algorithm is shown in figure 6, along with the Fourier transform of those points. The spectrum shows the typical blue-noise profile, with a large spike at the origin and an annulus of low energy surrounding it.

We show the empirical performance of our implementation in figure 7, in which we plot the ratio of time taken to generate N points and $N \log N$, during a run generating a pattern with 1,000,000 points. The performance matches the theoretical expectations. The sigmoidal behavior of the timing curve is probably due to tradeoffs between cache and memory behavior, as well as fewer valid choices in the binary tree traversal as large areas of $[0, 1]^2$ become covered by samples.

5 Discussion

Any implementation of geometric operations such as distance calculations and intersection of primitives must deal with the possibility of machine precision. Our algorithm is no exception. We have not used robust predicates in our implementation, but have taken some care to gracefully handle any inaccuracies due to precision. Our implementation is able to generate patterns of millions of

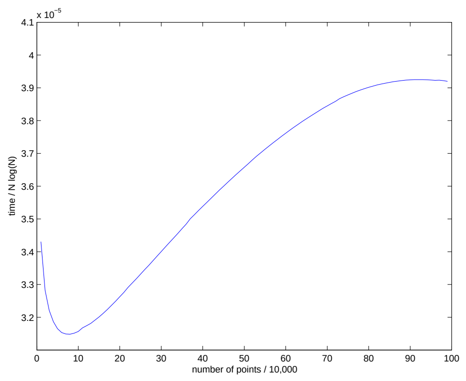

Figure 7: Time taken to generate N points divided by $N \log N$, plotted against the number of points generated.

points without roundoff becoming an issue. We have made use of GTS [Pop00] for its incremental Delaunay construction code and geometric predicates.

It is sometimes advantageous to vary the density of points in space [ODJ04]. It is possible to adapt our algorithm to generate patterns with spatially-varying densities, though it would require using a weighted Voronoi diagram [OBS92].

Similarly, hierarchical patterns are sometimes useful to allow refinement in sampling processes [MF92]. Our method allows for a complete pattern to be generated at a particular R, and then a more dense set to be generated by reducing R and continuing to add points.

When generating patterns with a large number of points, the density of points around the boundary of the $[0, 1]^2$ square is different than that in the middle, due to boundary effects. This can be overcome by inserting copies of every point with $+1$ or -1 offsets into the Voronoi diagram, or more simply though with less theoretical justification, generating a slightly denser pattern and discarding the edges.

One difficulty with any Poisson-Disk method is choosing the correct radius of exclusion to achieve a particular density of points. For very dense patterns, theoretical results on Poisson-disk patterns (AKA “random sequential adsorption” and “hard-core processes”) can be used to guide the choice of R. It is known, for example, that the coverage density, $\pi R^2 N/4$ approaches 0.548 in the limit [DWJ91]. For less dense patterns, since generation is not particularly expensive, we generally use a binary search to find a reasonable setting for R yielding a value close but just above the exact N we want, and scale the points up slightly around $(0.5, 0.5)$ to eliminate the few extra points.

Our method also extends to higher dimensions. Though it is not known how the complexity of the Voronoi diagram for Poisson Disk patterns behaves in higher dimensions, for uniform samples the results are promising [Dwy89].

Addendum

A similar $O(N \log N)$ algorithm has been proposed simultaneously with this work by Dunbar & Humphries [DH06]. They represent the free space where new points can be generated with a "scalloped sector" data structure, and avoid the rejection sampling discussed in section 3.1. They also give a linear-time algorithm for generating patterns that, while not Poisson-disk, are similar in their spectral properties. Our algorithm could be analogously modified to produce such patterns in linear time by randomly choosing a Voronoi region V with free space, rather than maintaining the weighted tree W, and placing a new sample on the circle of radius R centered at v_c in the portion where it intersects V, instead of using rejection sampling. We refer the reader to their work [DH06] for analysis of the properties of these approximate Poisson-disk patterns.

Acknowledgements

The author would like to thank Frédo Durand, Bob Sumner, Matthias Zwicker, Peter-Pike Sloan, and Don Mitchell for their helpful discussion of this paper, as well as the reviewers for their comments.

References

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

[DH06] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast Poisson-disk sample generation. To appear in Transaction on Graphics (SIGGRAPH 2006), 2006.

[DWJ91] R. Dickman, J.-S. Wang, and I. Jensen. Random sequential adsorption: Series and virial expansions. *J. Chem. Phys.*, pages 8252–8257, 1991.

[Dwy89] R. A. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 326–333, 1989.

[HDK01] Stefan Hiller, Oliver Deussen, and Alexander Keller. Tiled blue noise samples. In *VMV*, pages 265–272, 2001.

[LDM92] L.J.Guibas, D.E.Knuth, and M.Sharir. Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica 7*, pages 381–413, 1992.

[Llo83] S. Lloyd. An optimization approach to relaxation labeling algorithms. *Image and Vision Computing*, 1(2):85–91, 1983.

[MF92] Michael McCool and Eugene Fiume. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, pages 94–105, 1992.

[Mit87] Don P. Mitchell. Generating antialiased images at low sampling densities. In *Siggraph '87 Conference Proceedings*, pages 65–72, 1987.

[Mit91] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. In *Computer Graphics (Siggraph '91 Conference Proceedings)*, 1991.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.

[OBS92] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial tessellations: concepts and applications of Voronoi diagrams.* John Wiley & Sons, Inc., 1992.

[ODJ04] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.*, 23(3):488–495, 2004.

[Pop00] S. Popinet. Gts: The gnu triangulated surface library, 2000.

[Yel83] John I. Jr. Yellot. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, pages 382–385, 1983.