Using Scalloped Sectors to Generate Poisson-Disk Sampling Patterns

Daniel Dunbar
University of Virginia

Greg Humphreys
University of Virginia

Abstract

Sampling distributions with blue noise characteristics are widely used in computer graphics. Although Poisson-disk distributions are known to have excellent blue noise characteristics, they are generally regarded as too computationally expensive to generate in real time. We present a new data structure that alllows sampling by dart-throwing in $O(N \log N)$ time. We also show how a novel and efficient variation on this algorithm can be used to generate Poisson-disk distributions in $O(N)$ time and space.

1 Introduction and Background

Almost all problems in computer graphics involve sampling. It is well known that the properties of the sampling distribution can greatly affect the quality of the final result. In particular, blue-noise patterns perform especially well in this setting because of the low-energy annulus around the DC spike in their frequency spectrum. High quality sampling patterns are especially important when sampling the image plane in a raytracer, not only because they do a better job of capturing the continuous function being sampled, but also because in this setting the function being reconstructed is displayed directly, so any sampling errors will be especially apparent to a viewer.

Poisson-disk distributions have excellent blue noise spectra and also mimic the distribution of photoreceptors in a primate eye [Yellot 1983]. These distributions have proven difficult to generate directly, so many alternate approaches have been developed, few of which can guarantee the Poisson-disk property. In this paper, we describe an $O(N \log N)$ algorithm for directly generating maximal Poisson-disk distributions identical to those produced by a dart-throwing technique. We then present a variation of this algorithm that both yields better spectral distributions and runs in linear time and space. This algorithm generates point sets with excellent blue noise characteristics very quickly; it can generate over 200,000 points per second on a modern CPU.

1.1 Previous Work

Sampling theory is a well researched area of computer graphics, and it has even deeper roots in the signal processing and information theory literature. Stochastic sampling was first introduced to computer graphics by Dippé and Wold [1985]. Cook analyzed the spectral properties of various stochastic point processes [Cook 1986]. In that paper, he extols the virtues of Poisson-disk distributions because of their blue noise properties and relationship to photoreceptor distributions, but ultimately advocates the use of jittered grids because the straightforward dart-throwing algorithm for generating Poisson-disk distributions is prohibitively expensive. Since then, many algorithms have been proposed for generating point distributions; e.g. [Ulichney 1988; Shirley 1991; Hiller et al. 2001; Kollig and Keller 2002; Kollig and Keller 2003].

Mitchell's $O(N^2)$ "best-candidate" algorithm attempts to mimic dart-throwing while providing a termination guarantee [Mitchell 1991]. Whenever a new sample is to be drawn, a number of candidate samples are randomly generated, and the candidate that is farthest from the existing point set is accepted. This algorithm cannot guarantee the Poisson-disk property, but in practice it generates excellent point sets if enough candidates are drawn. The primary drawback of this technique is its long running time.

McCool and Fiume generated high-quality tile sets with a toroidal distance function so that they could be repeatedly tiled across the plane [McCool and Fiume 1992] and introduced the use of Lloyd's relaxation to improve the blue noise properties of the set. Lloyd's relaxation transforms a point set by moving each point to the center of its associated Voronoi region and is typically applied iteratively or used to generate distributions directly [Hiller et al. 2001]. Once the tiles have been generated, this method generates large point sets very efficiently. However, the tiling process introduces easily recognizable structures. This is acceptable for most sampling applications as long as the tile size is large enough, but is disastrous if the points are being used to distribute objects or as part of a texture basis function. Several schemes have been proposed to solve this problem.

Multiple authors have proposed using Wang tiles to solve this problem. Cohen et al. populated a set of Wang tiles with small point distributions that were intended to tile the plane [Cohen et al. 2003]. Lagae et al. showed that correctly applying this technique requires careful attention to how points are placed at the boundary of a tile and presented *Poisson-disk tiles* to address these issues [Lagae and Dutré 2005]. Most recently, Kopf et al. have extended these techniques to allow generation of point sets with blue noise properties satisfying an arbitrary density function [Kopf et al. 2006]. Their technique produces high quality point sets very efficiently once the tile set has been computed.

Ostromoukhov et al. described a fast technique for importance sampling a provided density function [Ostromoukhov et al. 2004]. Their algorithm generates point sets with local blue noise characteristics by using a clever modification of Penrose tiles and exploiting the tilings' aperiodic nature. However, when used to generate many points from a constant density function, the resulting point sets have larger angular anisotropy than techniques based on randomness.

Jones presents an algorithm for generating 2D Poisson-disk distributions in $O(N \log N)$ time [Jones 2006]. Like our technique, Jones' method builds a point set incrementally by storing neighboring regions of points in a balanced tree and inserting points into these regions one by one. However, neighbor regions are represented as Voronoi cells and Jones' method requires an incremental Delaunay triangulation algorithm with $O(\log N)$ performance when adding a point. This makes the implementation more complicated than that of our algorithm. Furthermore, in this paper we show a variation of our algorithm that runs in linear time.

2 Dart-Throwing in $O(N \log N)$

The dart-throwing method for computing Poisson-disk distributions iteratively refines an existing point set by generating a series of random candidate points in the sample domain and keeping only the first such point that is farther than the minimum distance $2r$ from all other points. Each sample effectively invalidates a disk of radius $2r$ centered around that point.

This algorithm is simple to implement and extends naturally to any domain with a well-defined and computable distance metric. However, the algorithm may not terminate, so in practice the algorithm is stopped after some fixed number of consecutive candidates have failed to be accepted.

One side effect of this approximation is that the generated point set is usually not maximal (there may be regions where a point could be placed without violating the distance criterion), so some regions of the domain may be undersampled. This problem is usually ignored, although Jones' technique is guaranteed to generate maximal distributions [Jones 2006]. Furthermore, since a large number of sample points is typically required, the algorithm is too slow to use directly. Consequently, several schemes for precomputing small distributions and tiling them have been proposed [Hiller et al. 2001; Cohen et al. 2003; Lagae and Dutré 2005].

We call the subdomain within which it is legal to add a point the *available subdomain*. Let $D(x, r)$ be the disk of radius r around a point x. For a domain X and existing point set P, the available subdomain is given by

$$A_X = X - \bigcup_{p \in P} D(p, 2r).$$

The key to emulating dart-throwing efficiently is the observation that we do not need to sample the entire available subdomain. Consider the annulus between radii $2r$ and $4r$ around some point. Every point in this annulus must be unavailable in any maximal distribution and therefore within a distance of $2r$ from some other point. This means that there must be at least one point that lies inside the annulus, and hence the union of all such annuli, intersected with the available subdomain, must contain at least one point. Therefore, it is possible to emulate dart-throwing by sampling from only this region, which we call the *available neighborhood* of a point set P. By carefully choosing the representation for this region, dart-throwing can be implemented in $O(N \log N)$ time.

2.1 Representing the Available Neighborhood

The problem of representing the available neighborhood can be divided into two parts: developing a spatial structure for the available region of the annulus, and partitioning the available neighborhood so that these structures can be efficiently updated upon the insertion of a new point. In addition, it must be possible to quickly generate a uniformly distributed random point inside the region.

Our solution to the first part of this problem involves a new data structure, the *scalloped region*. This data structure can be used to efficiently represent arbitrary boolean operations on 2D disks. We divide scalloped regions into a disjoint union of *scalloped sectors* (Figure 1). A scalloped sector is

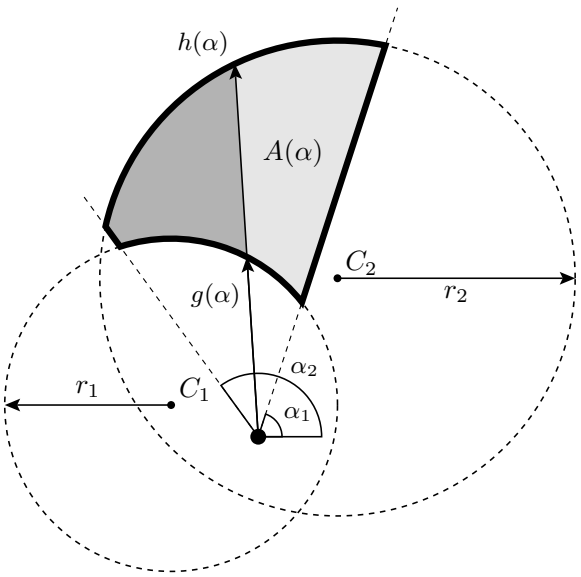


Figure 1: A *scalloped sector* is a sector bounded above and below by circular arcs. In the above diagram, α ranges from α_1 to α_2, and $g(\alpha)$ and $h(\alpha)$ are the distance functions from the sector's apex to the near and far arcs, respectively. $A(\alpha)$ is the partial area of the sector up to α, represented by the light gray area in the figure.

defined as the sector lying between angles α_1 and α_2 and bounded by near and far circular arcs. For convenience, the circular arcs are required to be non-intersecting, although they are allowed to meet at the edge of the sector. For the purposes of exposition the scalloped sector is assumed to be placed with the sector apex at the origin; in practice the apex is at some point P and the other coordinates are stored relative to this point.

The circular arcs are each described by a center C given in polar coordinates (d, γ), a radius r, and a sign $k \in \{-1, 1\}$ which selects either the near or far side of the circle. We refer to the near circular arc as (C_1, r_1, k_1) and the far circular arc as (C_2, r_2, k_2). For some angle α from the sector apex then we label the distance functions to the near and far bounding arcs as $g(\alpha)$ and $h(\alpha)$, respectively.

2.1.1 Distance to Bounding Arcs

Let $f(\alpha)$ be the distance along a ray from the sector apex at angle α to a particular circular arc. If the ray intersects the circle at a point Q then the points C, Q, and the sector apex form a triangle with sides d, r, and $f(\alpha)$ as in Figure 2. Additionally, the angle opposite the side of length r is $\alpha - \gamma$.

By the Law of Cosines,

$$r^2 = f^2(\alpha) + d^2 - 2f(\alpha)d\cos(\alpha - \gamma)$$

and solving for $f(\alpha)$ gives

$$f(\alpha) = d\cos(\alpha - \gamma) \pm \sqrt{r^2 - d^2\sin^2(\alpha - \gamma)}.$$

The parameter k is used to select either the near or far side of the circle, resulting in the equation

$$f(\alpha) = d\cos(\alpha - \gamma) + k\sqrt{r^2 - d^2\sin^2(\alpha - \gamma)}.$$

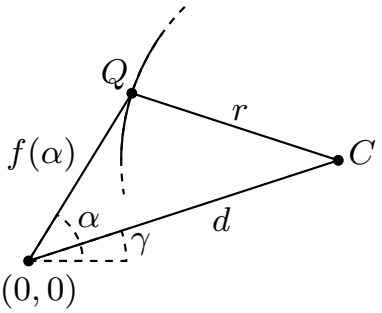


Figure 2: The distance $f(\alpha)$ from the sector apex at the origin to a bounding arc described by the center of a circle C and a radius r can be solved using the Law of Cosines.

2.1.2 Uniform Sampling in Scalloped Sectors

The area of a scalloped sector up to some angle α is given by

$$\begin{aligned} A(\alpha) &= \int_{\alpha_1}^{\alpha} \int_{g(\theta)}^{h(\theta)} r \, dr \, d\theta \\ &= \int_{\alpha_1}^{\alpha} \frac{h^2(\theta) - g^2(\theta)}{2} \, d\theta \\ &= \int_{\alpha_1}^{\alpha} \frac{h^2(\theta)}{2} \, d\theta - \int_{\alpha_1}^{\alpha} \frac{g^2(\theta)}{2} \, d\theta \end{aligned}$$

and the area of the full sector is $A(\alpha_2)$. Since g and h are simply variations of f we omit the full formula for $A(\alpha)$ and only note that the indefinite integral

$$\begin{aligned} \int f^2(\alpha) \, d\alpha = r^2(\alpha - \gamma) \\ + kr^2 \sin^{-1}(\sin(\alpha - \gamma)\frac{d}{r}) \\ + rkd \cos(\sin^{-1}(\sin(\alpha - \gamma)\frac{d}{r})) \sin(\alpha - \gamma) \\ + d^2 \cos(\alpha - \gamma) \sin(\alpha - \gamma). \end{aligned}$$

In order to uniformly sample from within a scalloped sector we would like to have a transformation from a uniformly distributed point in $[0,1]^2$ to the sector. If this transformation preserves relative areas then the point in the sector will also be uniformly distributed. The following theorem establishes the transformation which we use.

Theorem 1. *Let S be a 2D surface described in polar coordinates by an angular interval (α_1, α_2) and functions $g(\alpha)$ and $h(\alpha)$ with $0 < g(\alpha) < h(\alpha)$, where the functions f and g give the distance to the near and far boundary of the surface. That is, S is the image by $[0,1]^2$ of $\overline{\mathbf{x}}$ where*

$$\begin{aligned} \overline{\mathbf{x}}(u,v) &= (\overline{r}(u,v)\cos(\overline{\theta}(v)), \overline{r}(u,v)\sin(\overline{\theta}(v))) \\ \overline{\theta}(v) &= \alpha_1 + (\alpha_2 - \alpha_1)v \\ \overline{r}(u,v) &= g(\overline{\theta}(v)) + (h(\overline{\theta}(v)) - g(\overline{\theta}(v)))u. \end{aligned}$$

Let

$$A(\alpha) = \int_{\alpha_1}^{\alpha} \int_{g(\theta)}^{h(\theta)} r \, dr \, d\theta$$

be the area of S up to some angle α, then A^{-1} exists and

$$\begin{aligned} \mathbf{x}(u,v) &= (r(u,v)\cos(\theta(v)), r(u,v)\sin(\theta(v))) \\ \theta(v) &= A^{-1}(vA(\alpha_2)) \\ r(u,v) &= \sqrt{g^2(\theta(v)) + (h^2(\theta(v)) - g^2(\theta(v)))u} \end{aligned}$$

is a map from $[0,1]^2$ to S that preserves area up to multiplication by a constant.

Proof. Observe that $\mathbf{x}([0,1]^2) = \overline{\mathbf{x}}([0,1]^2) = S$ and that A is the integral of a strictly positive function and therefore is monotonic and invertable.

The partial derivatives of $\mathbf{x}$ are

$$\begin{aligned} \mathbf{x}_u &= (r_u \cos(\theta), r_u \sin(\theta)) \\ \mathbf{x}_v &= (r_v \cos(\theta) - r\theta' \sin(\theta), r_v \sin(\theta) + r\theta' \cos(\theta)) \end{aligned}$$

and

$$\begin{aligned} E &= \langle \mathbf{x}_u, \mathbf{x}_u \rangle = r_u^2 \\ F &= \langle \mathbf{x}_u, \mathbf{x}_v \rangle = r_u r_v \\ G &= \langle \mathbf{x}_v, \mathbf{x}_v \rangle = r_v^2 + r^2(\theta')^2 \end{aligned}$$

are the coefficients of the first fundamental form. The area element of the surface

$$\begin{aligned} dA &= \sqrt{EG - F^2} \, du \wedge dv \\ &= r r_u \theta' \, du \wedge dv \end{aligned}$$

and so it is only necessary to verify that $r r_u \theta'$ is constant.

Since $A(A^{-1}(x)) = x$ then taking the derivative of both sides yields

$$(A^{-1})'(x) A'(A^{-1}(x)) = 1.$$

The derivative of A,

$$\begin{aligned} A' &= \int_{g(\theta)}^{h(\theta)} r \, dr \\ &= \frac{h^2(\theta) - g^2(\theta)}{2} \end{aligned}$$

is such that $A'(\theta) \neq 0$, thus

$$(A^{-1})'(x) = \frac{1}{A'(A^{-1}(x))},$$

and

$$\begin{aligned} \theta' &= A(\alpha_2)(A^{-1})'(vA(\alpha_2)) \\ &= A(\alpha_2) \frac{1}{A'(A^{-1}(vA(\alpha_2)))} \\ &= A(\alpha_2) \frac{1}{A'(\theta)}. \end{aligned}$$

Finally,

$$\begin{aligned} r_u &= \frac{h^2(\theta) - g^2(\theta)}{2r} \\ &= \frac{A'(\theta)}{r} \end{aligned}$$

and so $dA = r r_u \theta' \, du \wedge dv = A(\alpha_2) \, du \wedge dv$. □

Corollary 1. *Given a surface S as in Theorem 1 and a point (ξ_1, ξ_2) selected from $[0,1]^2$ with a uniform distribution, then $\mathbf{x}(\xi_1, \xi_2)$ is a uniformly distributed point on S.*

Generating the point inside the sector is done by transforming a uniformly distributed random point (ξ_1, ξ_2) in $[0,1]^2$ to a point (d, θ) in polar coordinates using the transformation $\mathbf{x}$ in Theorem 1. Although we do not have a closed form equation for A^{-1}, it is known to exist and in practice binary search is sufficient to evaluate A^{-1} efficiently.

2.1.3 Uniform Sampling in the Available Neighborhood

The available neighborhoods is represented as a set of scalloped regions. In order to uniformly sample from the available neighborhood we first select a scalloped region in the neighborhood using the regions' areas as a probability distribution. Subsequently we select a scalloped sector from the scalloped region using the sectors' areas as the probability distribution. Finally we generate a point in the sector uses the methods of the preceeding section.

The maximum number of scalloped regions depends on the given radius r and the set of regions varies dynamically during the sampling process. In order to efficiently sample from the available neighborhood the scalloped regions are stored in a binary tree ordered by the regions area. We also store the sum of all areas below a given tree node with that node. This allows selecting a scalloped region according to weight in $O(\log N)$ time. Listing 1 shows how to select a region using this tree structure for some random value $p \in (0,1)$.

Listing 1: Selecting Weighted Nodes

```
def choose(root, p):
  weight = p*root.sumOfWeights

  node = root
  while 1:
    if node.left:
      if weight<node.left.sumOfWeights:
        node = node.left
      else:
        weight -= node.left.sumOfWeights
    if weight<node.weight:
      break
    else:
      weight -= node.weight
      node = node->right
  return node
```

2.1.4 Disk Subtraction

Scalloped regions contain disjoint scalloped sectors stored in a list and are typically initialized to contain a single scalloped sector representing a disk or annulus. Subsequent modifications are performed by applying the requisite boolean operation on each individual sector and substituting the resulting sectors for the old one.

We will only describe the method for disk subtraction; this is the only operation necessary for Poisson-disk sampling. There are a large number of cases that can occur during disk subtraction due to the necessity of differentiating between the near and far sides of the intersecting disk, tangent points of the disk, and intersections of the disk with the sector boundary. For simplicity we have chosen a method that restricts the number of cases that need to be considered.

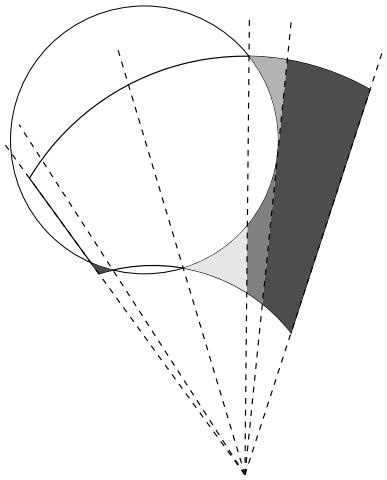


Figure 3: The subtraction of a scalloped sector by a disk results in a number of new sectors. The rays corresponding to the angles that need to be considered by our algorithm are shown as dash lines and new sectors in shades of gray.

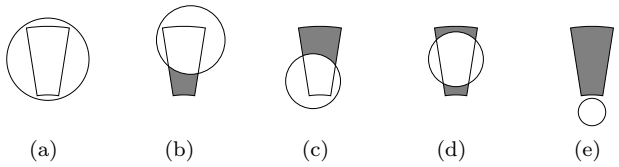


Figure 4: Cases for subdivided sectors occurring during subtraction by a disk.

Subtraction of a scalloped sector by a disk is performed by computing the angles relative to the sector center of any intersection points between the disk boundary and the near and far bounding arcs. Additionally, the angles at which the disk is tangent to a ray emanating from the sector center are recorded. These angles are inserted into a list along with the angular endpoints α_1 and α_2. Any angles outside the interval (α_1, α_2) are discarded; the maximum number of angles in the list is eight, four for intersections with the bounding arcs, two for the angles of tangency, and two for the angular endpoints. Finally, the resulting list is sorted. Figure 3 shows the angles that will be consider for an example disk subtraction operation.

For every adjacent pair of angles in the list the resulting sectors for that range are then output in order. Adjacent angles have the property that the result of the subtraction restricted to that angular range can result in only zero, one, or two new sectors. The possible cases that can occur within an angular range are shown in Figure 4.

Classification is straightforward, for an angular range (α, β) some interior angle $\theta \in (\alpha, \beta)$ is chosen. Let s_1 (s_2) be the distances along θ from the sector center to the near (far) edge of the disk, or $-\infty$ ($+\infty$) if this distance does not exist. The following sectors for the angular interval (α, β) are then output:

- $s_1 \geq h(\theta)$ **or** $s_2 \leq g(\theta)$ – a sector with the same near

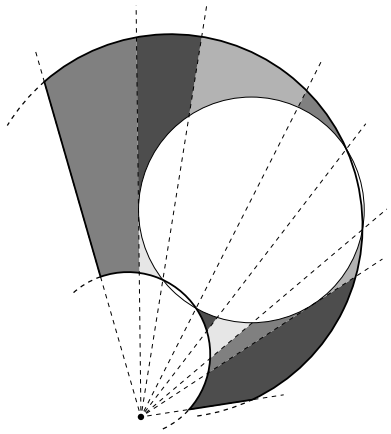


Figure 5: The maximum number of scalloped sectors that can be generated from a single subtraction is ten, as is shown here. If sector merging is used then the maximum is six sectors.

and far bounding arcs as the original sector. This corresponds to a disk falling completely outside the subrange (Case (e)).

- $s_1 > g(\theta)$ – a sector with the near bounding arc of the original sector and the far bounding arc being the near side of the disk. This occurs when the near edge of the disk is above the near bounding arc (Cases (b), (d)).
- $s_2 < h(\theta)$ – a sector with the near bounding arc being the far edge of the disk and the far bounding arc of the original sector. This occurs when the far edge of the disk is below the far bounding arc (Cases (c), (d)).

The method may output more sectors than are necessary due to the manner of dividing into angular ranges. Since sectors are output in angular order this problem can be solved by comparing newly output sectors to the previous two sectors. If the bounding arcs are the same and the new sector simply extends the angular range then the previous sector can be updated to cover the full range and the new sector can be discarded.

Since the maximum number of angles is eight the maximum numbers of angle pairs considered is seven. If the full seven are present then the first and last must be between the sector bounding angles α_1 or α_2 and an angle of tangency. For these pairs only one sector will be output. Further, two pairs must represent the range where the disk intersects the sector bounding arcs, and so only one sector either above or below the disk can be output. Therefore the maximum number of sectors that can be output by this method is ten, one for each of the four pairs mentioned and two for each of the other three pairs, which may output at most two sectors. Figure 5 demonstrates a subtraction operation that achieves this maximum. If sector merging is used then the maximum is six, corresponding to one sector left and right of the angles tangent to the disk and two sectors above and below the disk.

2.1.5 Available Neighborhoods of Points

The available neighborhood is partitioned into scalloped sectors of outer radius $4r$ around each point in order to restrict the number of sectors that must be updated after point insertion to a small constant. Efficient sampling, however, requires that all of these neighborhoods be disjoint. In general, if an ordering relation is defined for a set S of sets it is possible to derive a new set S' of *disjoint* sets where

$$\bigcup_{s' \in S'} s' = \bigcup_{s \in S} s,$$

by subtracting from each set the union of all members of S that are less than it in the relation.

We use the generation order of the points as an ordering relation and then define the available neighborhood of a point $p \in P$ as

$$N_p = D(p, 4r) - \bigcup_{p' \in P} \begin{cases} D(p', 4r), & p' < p \\ D(p', 2r), & p' \geq p \end{cases}.$$

The available neighborhood is $N = \bigcup_{p \in P} N_p$ (Figure 6). Each disjoint N_p is computed using boolean disk subtraction.

2.2 Algorithm Details and Complexity

Our algorithm A_1 for efficient dart-throwing begins with an initial set consisting of a single point randomly chosen in the domain. During sample generation, we maintain an associative map from candidate points (points with non-empty available neighborhoods) to their associated neighborhoods.

A candidate point is then randomly chosen (using neighborhood areas as a probability distribution) and a random point within its neighborhood is added to the point set. The available neighborhood for the new point is an annulus from radii $2r$ to $4r$, minus a disk of radius $4r$ around the nearby points. The maximum distance required to search for neighbors is $8r$ since the scalloped region and neighbor disk are both bounded by $4r$. All nearby neighborhoods are then updated by subtracting a disk of radius $2r$ around the newly inserted point. This process continues until no candidate points remain.

The maximum number of scalloped sectors in an available neighborhood is bounded by a constant. Furthermore, the Poisson-disk distance condition bounds the number of neighbors within a fixed radius. We can therefore use a uniform grid to implement the neighbor search and update of the available neighborhoods in $O(1)$ time. Similarly, picking an individual scalloped sector within an available neighborhood and generating a point in that sector can be done in $O(1)$ time. By using the balanced tree data structure described in Section 2.1.3, we can choose an available neighborhood of a point according to its area and update the tree within $O(\log N)$ time, so the time complexity of the entire algorithm is $O(N \log N)$ where N is the number of generated points. The space complexity is $O(N)$.

If we drop the requirement that the available neighborhoods be sampled according to an area-weighted probability density function then this new algorithm A_2 runs in linear time. In practice the cost of maintaining the sectors, intersecting them with disks, and updating data structures dominates the running time and this does not result in a performance increase even for large point sets.

Pseudo code for the algorithm is given in Listing 2 and empirical results confirm that the generated point sets exhibits spectral properties matching those of dart-throwing.

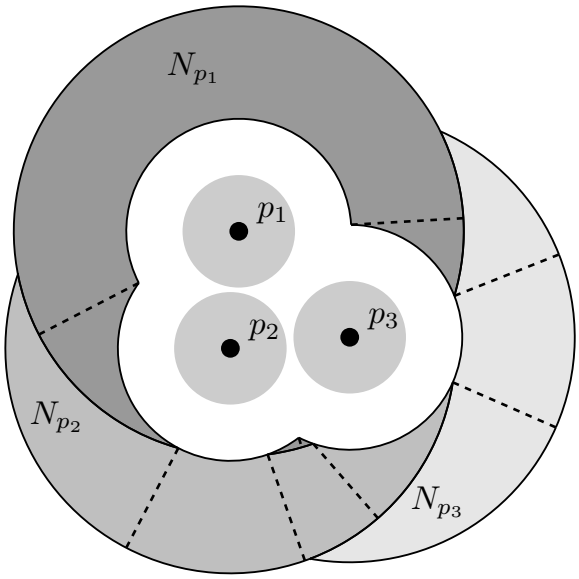


Figure 6: A partial point set and its neighborhoods. The dashed lines represent scalloped sector boundaries within a region.

3 Boundary Sampling

In this section, we show how the algorithms described in the previous section can be modified to avoid the complexity of sector operations, thereby generating Poisson-disk point sets in linear time extremely quickly. First, notice that it is possible to generalize either A_1 or A_2 by introducing a *annulus radius factor* c_a to vary the outer radius of the annulus defining the available neighborhoods, where $2 < c_a r \leq 4$. This modification does not change the structure or asymptotic performance of either algorithm, and we will assume that they are parameterized by c_a for the rest of the paper.

With this change, the available neighborhood is defined as

$$N_p = D(p, c_a r) - \bigcup_{p' \in P} \begin{cases} D(p', c_a r), & p' < p \\ D(p', 2r), & p' \geq p \end{cases} .$$

Notice that the overall density of the generated point set tends roughly to be inversely proportional to c_a; this can be exploited for applications such as randomized object placement, in which it is desirable to tune the density of the point set. Figure 7 graphs the density of a sample point set as c_a is varied.

A special case arises if c_a is taken to be the minimum value 2. In this case, a point's available neighborhood collapses to a collection of circular arcs centered at the point. We call these arcs the *available boundary*. By directly implementing boundary sampling, we no longer need to represent the available neighborhood as scalloped regions; instead, the available boundary is represented as a set of per-point angular ranges at which a point can be placed on the boundary.

Additionally, if we select the new candidate point at random instead of according to the length of its available boundary (similarly to how we obtained the linear algorithm A_2), it is no longer necessary to explicitly store the neighborhoods for every point already in the set. Once a candidate has been chosen, its available boundary can be quickly computed by intersecting the boundary circles of the candidate with its immediate neighbors. If the candidate point is P and Q is some neighbor with polar coordinates (d, θ) then the angle range about P that will be occluded is

$$(\theta - \cos^{-1}(\frac{d}{4r}), \theta + \cos^{-1}(\frac{d}{4r})),$$

(Figure 8).

After the legal ranges have been determined, new points can be repeatedly placed at available locations on the boundary until the available boundary is empty. The addition of a new point only requires subtracting a single angular range from the candidate's boundary.

The resulting algorithm A_3, which we call *boundary sampling*, is simple to implement and runs in $O(N)$ time and space. Pseudo code for the algorithm is given in Listing 3 and an implementation is available from our website. Our implementation is approximately 200 lines of C++ code and can generate over 200,000 points per second on a 3 GHz Pentium 4.

Listing 2: Algorithm 1

```
def sample(radius):
  pt = (random(),random())
  Npt = Annulus(pt, 2*radius, 4*radius)

  P = [pt]
  C = {pt : Npt}
  T = WeightedTree()
  T.insert(pt, Npt.area)

  while C:
      # choose a random point in the
      # available neighborhood
    Np = choose(T, random())
    pt = Np.randomPoint()
    P.append(pt)

      # update the available neighborhoods
      # of the new point and its neighbors
    Npt = Annulus(pt, 2*radius, 4*radius)
    for n in ... neighbor points ...:
      Npt.subtractDisc(n, 4*radius)

      if n in C:
        C[n].subtractDisc(pt, 2*radius)
        if C[n].isEmpty():
          C.remove(n)
          T.remove(n)
        else:
          T.update(n, n.area)

    if not Npt.isEmpty():
      C[pt] = Npt
      T.insert(pt, Npt.area)

  return P
```

4 Results

In this section, we show results from the boundary sampling algorithm described in Section 3, and compare them to other methods for computing Poisson-disk distributions.

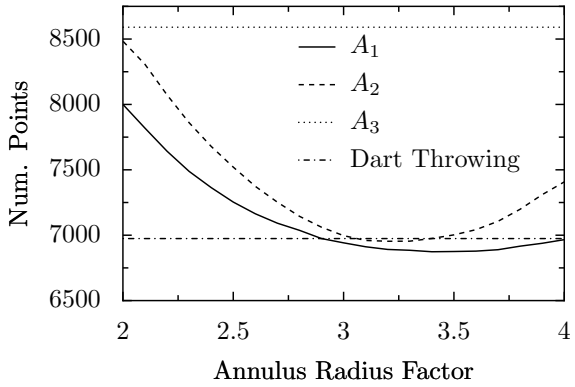


Figure 7: The effects of varying the outer annulus radius for an example point set with $r = 0.01$. The number of points represent the average for 30 trials. For reference, lines for boundary sampling (A_3) and traditional dart throwing are included although they do not use the annulus radius factor.

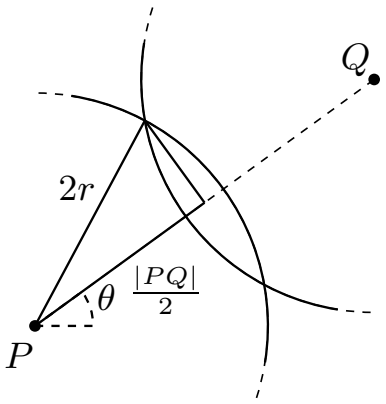


Figure 8: If P and Q are points in a Poisson-disk set with radius r then the angle range about P that Q occludes is easily computed from the geometry of the situation.

The best tiling schemes can generate point sets very efficiently and with spectra comparable to dart-throwing or Lloyd's relaxation, although there will be energy and anisotropy spikes associated with any tiling. Although we do not currently have access to an implementation of a sophisticated tiling method, we expect that the runtime performance of our algorithm is comparable to that of a tiling scheme, but our results will be artifact-free and require no precomputation.

We analyze the properties of two-dimensional noise distributions in the style of McCool and Fiume, who compute the radial power and anisotropy using the periodogram of a point set [McCool and Fiume 1992]. The primary characteristic of a blue noise distribution is a low energy annulus around the central DC spike with energy returning to a relatively constant value outside the annulus. The quality of a distribution depends on the magnitude of the difference between the DC spike, the low energy annulus, and the average energy in the high frequencies. Evaluating the distribution in terms of radial power also requires analyzing the radial anisotropy to ensure that the radial power spectrum is an accurate representation of the pattern along all orientations.

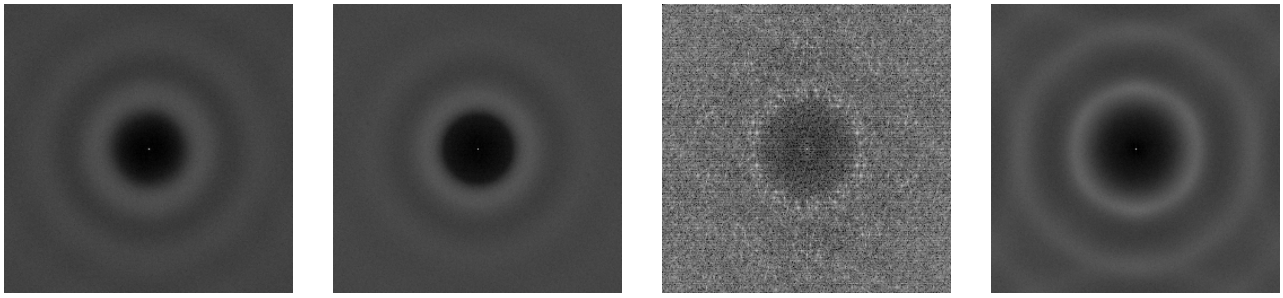
Listing 3: Algorithm 2

```
def sample(radius):
  pt = (random(),random())
  P = [pt]
  C = [pt]

  while C:
    candidate = C.popRandom()

      # compute already occluded angles
    ranges = AngularRange(0, pi*2)
    for n in ... neighbor points ...:
      dx = n[0]-candidate[0]
      dy = n[1]-candidate[1]
      d = sqrt(dx*dx + dy*dy)
      angle = atan2(dy,dx)
      theta = acos(d/(4*radius))
      ranges.subtract(angle-theta, angle+theta)

      # maximize boundary
    while ranges:
      angle = ranges.randomAngle()
      pt = (candidate[0] + cos(angle)*2*radius,
            candidate[1] + sin(angle)*2*radius)
      ranges.subtract((angle-pi/3, angle+pi/3))

      P.append(pt)
      C.append(pt)

  return P
```

Figure 9 shows the radial power spectra of boundary sampling compared to both dart-throwing and linearized dart-throwing. The graph shows that neither linearized sampling nor boundary sampling significantly change the blue noise properties of the resulting distributions.

Figure 10 displays averaged periodograms for 100 point sets having a radius of 0.02, resulting in approximately 2000 points each. The boundary sampling periodogram shows that the higher point density results in a greater magnitude difference between the low energy annulus and the peak transition energy. The periodograms also show that the method of Ostromoukhov et al. is significantly less uniform than that of our method. They address this issue by precomputing relaxation vectors, but these precomputed tables are only sufficient to improve the blue noise properties of small local regions of the generated distribution. As the number of points grows, the lookup table is no longer able to compensate for the inherent structure of the Penrose tiling. Because we are interested in using these point sets for sampling the image plane in high quality image synthesis, hundreds of millions of points will likely be required, and the lookup tables would become quite large.

Timing results for several methods of computing Poisson-disk distributions are shown in Table 1. For methods that require specification of a radius, one is chosen so that the number of generated points is approximately equal to the given value of N, and the time is computed as N times the average number of points per second. The times for Ostromoukhov's method were generated with code provided by the authors of that paper, although they state that the provided code is not fully optimized. The results show that although the linear approximation of dart-throwing (A_2) is more efficient than true dart-throwing (A_1) for large numbers of points, the computational overhead of sector subtrac-



(a) A_1: Dart-throwing (b) Best Candidate (c) Ostromoukhov et al. (d) A_3: Boundary sampling

Figure 10: Averaged periodograms for several sampling methods. Boundary sampling generates the best blue noise spectrum due to its extremely regular and dense sampling of the plane. Ostromoukhov et al.'s results are noisier because their technique does not involve randomness, so averaging multiple runs does not provide smooth periodograms.

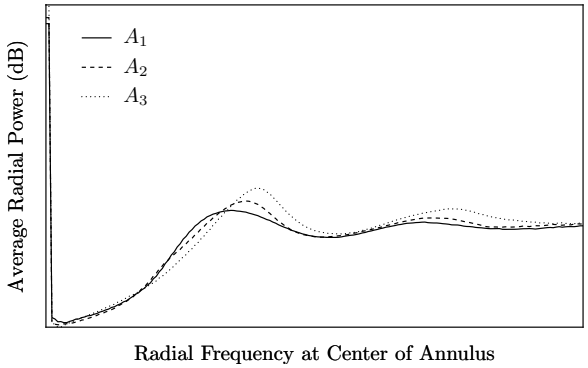


Figure 9: The blue noise properties of the distribution are preserved even if only the boundary of the available neighborhood is sampled. Sampling without regard to the probability density of available neighborhoods also preserves the blue noise properties.

N	1000	10000	100000
Best Candidate	1.454	157.014	6.084h
Dart-throwing (A_1)	0.573	5.905	141.901
O(N) Dart-throwing (A_2)	0.667	6.442	61.186
Ostromoukhov et al.	0.015	0.095	1.546
Boundary Sampling (A_3)	0.001	.058	0.496

Table 1: Timing results for generating point sets of varying sizes. Times are in seconds except where otherwise noted.

tion still adds significant overhead compared to boundary sampling (A_3). For small numbers of points, A_2 may perform more disk subtractions than A_1, making it less efficient than its $O(N \log N)$ counterpart.

5 Conclusion

We have described a new technique for efficiently implementing the dart-throwing algorithm for the generation of Poisson-disk point sets, based on the manipulation of disjoint unions of scalloped sectors. This algorithm runs in $O(N \log N)$ time and motivates a new algorithm for generating 2D Poisson-disk point sets that runs in $O(N)$ time and space and produces excellent blue noise patterns.

References

Cohen, M. F., Shade, J., Hiller, S., and Deussen, O. 2003. Wang tiles for image and texture generation. *ACM Transactions on Graphics 22*, 3, 287–294.

Cook, R. L. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1, 51–72.

Dippé, M. A. Z., and Wold, E. H. 1985. Antialiasing through stochastic sampling. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, ACM Press, New York, NY, USA, 69–78.

Hiller, S., Deussen, O., and Kaufmann, A. 2001. Tiled blue noise samples. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, Aka GmbH, 265–272.

Jones, T. 2006. Efficient generation of poisson-disk sampling patterns. *Journal of Graphics Tools, to appear*.

Kollig, T., and Keller, A. 2002. Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3, 557–563.

Kollig, T., and Keller, A. 2003. Efficient illumination by high dynamic range images. *Rendering Techniques*, 45–51.

Kopf, J., Cohen-Or, D., Deussen, O., and Lischinski, D. 2006. Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics 25*, 3.

Lagae, A., and Dutré, P. 2005. A procedural object distribution function. *ACM Transactions on Graphics 24*, 4, 1442–1461.

McCool, M., and Fiume, E. 1992. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 94–105.

Mitchell, D. P. 1991. Spectrally optimal sampling for distribution ray tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, ACM Press, New York, NY, USA, 157–164.

Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics 23*, 3, 488–495.

Shirley, P. 1991. Discrepancy as a quality measure for sample distributions. In *Proceedings of Eurographics*, 183–194.

Ulichney, R. A. 1988. Dithering with blue noise. In *Proc. of the IEEE 76*, 56–79.

Yellot, J. I. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 221*, 382–385.