DOI: 10.1111/cgf.14606
Eurographics Symposium on Geometry Processing 2022
M. Campen and M. Spagnuolo
(Guest Editors)

Volume 41 (2022), Number 5

Deterministic Linear Time for Maximal Poisson-Disk Sampling using Chocks without Rejection or Approximation

Scott A. Mitchell

Center for Computing Research, Sandia National Laboratories, U.S.A.

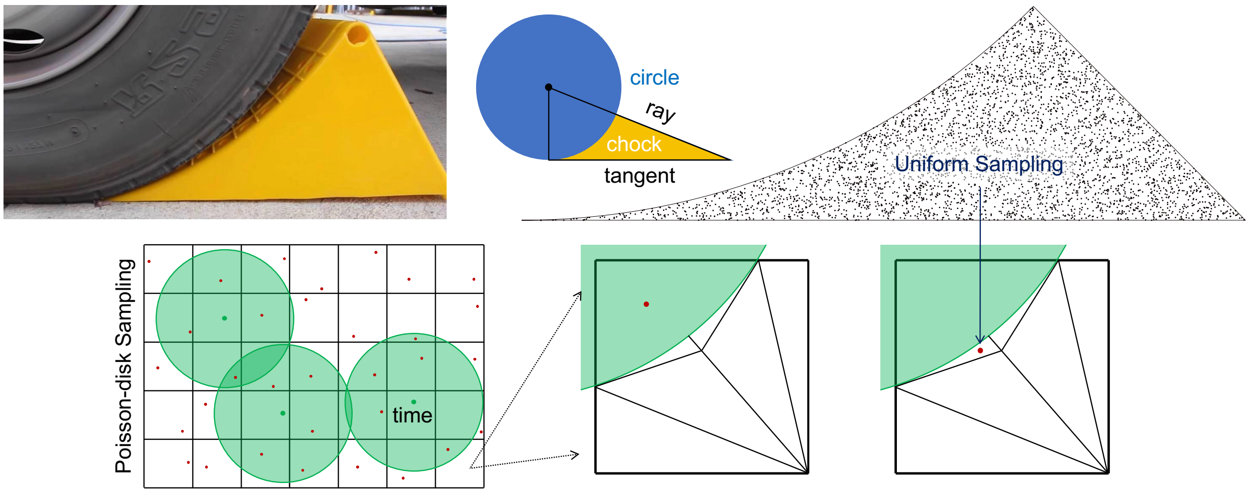


Figure 1: *Upper left, a real-life yellow wheel chock; image courtesy* `https://safetyline.com.au`*. Upper center, our mathematical "chock." Upper right, we show how to perform uniform random sampling in a chock without rejection or approximation, in constant time per sample. Lower, pipeline, this enables 2D maximal Poisson-disk sampling in deterministic linear time, which we name "Deterministic-MPS."*

Abstract

We show how to sample uniformly within the three-sided region bounded by a circle, a radial ray, and a tangent, called a "chock." By dividing a 2D planar rectangle into a background grid, and subtracting Poisson disks from grid squares, we are able to represent the available region for samples exactly using triangles and chocks. Uniform random samples are generated from chock areas precisely without rejection sampling. This provides the first implemented algorithm for precise maximal Poisson-disk sampling in deterministic linear time. We prove $O(n \cdot M(b) \log b)$, where n is the number of samples, b is the bits of numerical precision and M is the cost of multiplication. Prior methods have higher time complexity, take expected time, are non-maximal, and/or are not Poisson-disk distributions in the most precise mathematical sense. We fill this theoretical lacuna.

CCS Concepts

• ***Mathematics of computing*** → *Distribution functions;* • ***Computing methodologies*** → *Rendering;* • ***Theory of computation*** → ***Randomness, geometry and discrete structures;***

1. Introduction

Blue-noise is defined by the Fourier spectrum of points' pairwise distance vectors. Such distributions have random point placement, but no two points are too close together. This avoids bias and aliasing artifacts in images, and is efficient because there are no redundant samples. Poisson-disk sampling is valued in graphics because it generates distributions close to blue-noise. There are other ways to generate blue-noise, but Poisson-disks hold a special place within the pantheon of sampling methods, for historical, practical and theoretical reasons. Herein we restrict our attention to two dimensions, the most important case, but the community has some interest in higher-dimensional Poisson disk sampling and blue noise.

Poisson-disk sampling is defined by the so-called Poisson-disk process: points arrive uniformly at random at a fixed mean rate. If a point falls within the disk around a prior point it is "covered" and rejected; else it is accepted. Maximal Poisson-disk Sampling (MPS) continues this indefinitely until the entire domain is covered. Unfortunately, there is no precise definition of the maximal Poisson-disk output distribution beyond following this process. This is analogous to defining "sort" by the process of bubble sorting, and not by the output being in ascending order. This has made it difficult to efficiently generate distributions that are precise maximal Poisson-disk samplings. To modify the procedure but still generate true Poisson-disk samples, the only known mathematical tools are the observation that the probability of the next point appearing in a particular uncovered subregion is proportional to the area of the subregion, and how to model and update the explicit arrival time of points.

2. Background

Inverse Transform Sampling. Sampling uniformly from a triangle or rectangle is straightforward. For more complicated shapes or distributions, Inverse Transform Sampling [Dev86] is a way to transform a uniform random variable into one from the target distribution. It relies on having a formula for the Cumulative Distribution Function (CDF) of the target distribution. By definition, any area formula is a CDF of the sample density for uniform-by-area sampling. If $u \in [0,1]$ is a uniform random variable, then $v = \mathrm{CDF}^{-1}(u)$ is a random variable from the target. The typical challenge is inverting the CDF. Our MPS algorithm is possible because we found a way to do this for the area of a "chock:" a curved triangle bounded by a circle, a ray, and a tangent; see Figure 1.

Precision Complexity. We consider the problem of ensuring the process of generating MPS sample positions is accurate to b bits of precision. Let $M(b)$ be the cost of b-bit multiplication. Then tan and arctan have time complexity $\mathrm{O}(M(b)\log b)$. The surprising result that trig and exp functions have the same complexity as their inverses results from the quadratic convergence of Newton's method, and using low-precision arithmetic for the first iteration and doubling it every iteration. [Bre76, BZ10] In the same way, we will show the area of a chock and its inverse are $\mathrm{O}(M(b)\log b)$.

2.1. MPS Methods

We give a detailed review of specific features of related MPS methods, because our contribution hinges on replacing their approximations and probabilistic operations with precise deterministic ones. We summarize these features in Table 1.

Many Poisson-disk sampling papers [Jon06, EPM*11] have not made the distinction between expected and deterministic runtime, and have used $\mathrm{O}(\cdot)$ notation regardless. Herein we distinguish deterministic "big O" $\mathrm{O}(\cdot)$ running time vs. probabilistic expected $\mathbb{E}(\cdot)$ running time. For some algorithms the expected time is only observed empirically, and not proven.

Dart-throwing [DW85] was the first proposed Poisson-disk algorithm. It sequentially selects a sample location uniformly at random over the whole domain, and rejects it if it lies inside the disk of a prior sample. The arrival time of each sample is not modeled, only their order, but it is equivalent to following the Poisson-disk process with explicit arrival times. The implementation terminates after a certain number of rejections are observed, without any guarantee of maximality, and the runtime is not described analytically.

The following algorithms avoid generating samples from the entire domain and getting many rejections. Instead, they find the uncovered region by subtracting the accepted samples' disks from the entire domain. They construct some representation of this curved uncovered region, often a collection of triangles or squares. The next sample is chosen uniformly by area from the shapes in that representation. If the representation is an outer approximation (i.e. superset) of the uncovered domain, then the next sample is rejected if it lies outside the true uncovered domain. For an inner approximation (subset) no rejections occur, but the distribution is not precisely Poisson-disk, and some algorithms may terminate without strict maximality.

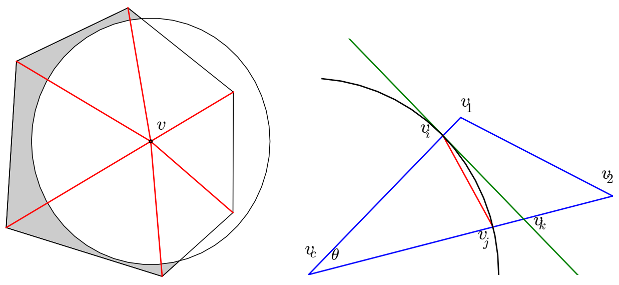


Figure 2: *Left, Voronoi-MPS's representation of the uncovered area (shaded) by Voronoi cells with subtracted disks. Right, it decomposes into triangles $\triangle v_{ijk}$ containing chocks, and quadrilaterals $\square v_{i12k}$. Images courtesy Jones Figures 3 and 5. [Jon06]*

Voronoi-MPS, "Efficient Generation of Poisson-Disk Sampling Patterns" [Jon06] was the first method to generate precise Maximal Poisson-disk distributions with expected $\mathbb{E}(n\log n)$ running time. Voronoi-MPS partitions the remaining uncovered region by a Voronoi diagram of the accepted samples. It subtracts sample disks from a Voronoi cell, and partitions it into quadrilaterals and chocks; see Figure 2. Each chock is outer approximated by the triangle of its vertices. Voronoi-MPS first selects a polygon uniformly by area, then generates a sample within it uniformly by area. If it is a quadrilateral, the sample is always accepted. If it is a triangle and the sample falls outside the chock, then the sample is rejected and one must start over at the top by selecting another polygon. A chock's area is always a large fraction of its enclosing-triangle's area. Hence the chance of a rejection is bounded above by a constant. The entire running time is $\mathbb{E}(n\log n)$. The $\log n$ arises from constructing the Voronoi diagram and from selecting a polygon uniformly by area.

GridOuter-MPS, "Efficient Maximal Poisson-Disk Sampling" [EPM*11] uses a uniform background grid instead of a Voronoi decomposition. The sample regions are squares with disks subtracted, *scooped-squares*. An outer approximation to each scooped-square is used for selecting the next subregion, and for rejection sampling within it; see Figure 3. Thus, as with Voronoi-MPS, the output is a maximal Poisson-disk sampling in the strictest sense, and the runtime is $\mathbb{E}(n\log n)$, not deterministic.

method	approx.	Poisson-disks	maximal	expected time	deterministic time	precision complexity
Dart-throwing [DW85]	outer	Y	-	∞	-	1
Scallop-MDS [DH06a]	in	-	Y	-	$n\log n$	$b+\log b$
Voronoi-MPS [Jon06]	outer	Y	Y	$n\log n$	-	1
GridOuter-MPS [EPM*11]	outer	Y	Y	$n\log n$	-	1
Sequential-MPS [ours]	-	Y	Y	-	$n\log n$	$\log b$
Simple-MPS [EMP*12]	outer	Y	Y	$n^{\dagger}$	-	2^b
GridInner-MPS [JK11]	in	Y‡	Y‡	-	n	2^b
ChockSubdivision-MPS [ours]	-	Y	Y	-	n	$b+\log b$
Deterministic-MPS [ours]	-	Y	Y	-	n	$\log b$

approx. indicates whether samples are generated from an outer approximation (superset) of the uncovered domain, an inner approximation (subset), or from it exactly up to numerical precision.
precision complexity means the cost of achieving an MPS within b bits of numerical accuracy.
The total complexity is a "time" column for n samples $\times$ the "precision complexity" column for b bits $\times$ the cost of b-bit multiplication.
Sequential-MPS is our version of GridOuter-MPS, using chock sampling without outer approximation and rejections.
† **Simple-MPS** linear runtime is empirical. The dependence on bits of precision is small empirically, but exponential in the worst case.
‡ **GridInner-MPS** approximates exclusion disks by enclosing polygons, whose number of sides trade runtime for precision.
Deterministic-MPS is our version of GridInner-MPS, using chock sampling without an inner approximation.

Table 1: *Features of Poisson-disk sampling algorithms. Outer approximations lead to rejections and expected time. Inner approximations lead to distribution deviations. In precision complexity, the 2^b factors come from the magnitude of the geometric approximation. The b factors come from binary search. The $\log b$ factors come from* tan, arctan, *or similar functions.*

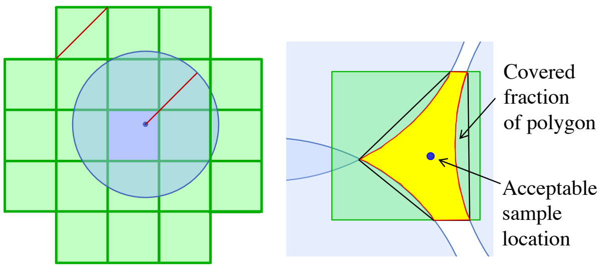


Figure 3: *GridOuter-MPS's scooped-square. Left, many methods use a background grid of squares with diagonal length equal to the Poisson-disk radius. This ensures only one sample can fit in a square, and only 20 neighbor squares could intersect its disk. Right, a polygon forms an outer approximation of a scooped-square. The polygon is triangulated, and rejection sampling is performed. Images courtesy "Efficient Maximal Poisson-Disk Sampling" presentation. [EPM*11]*

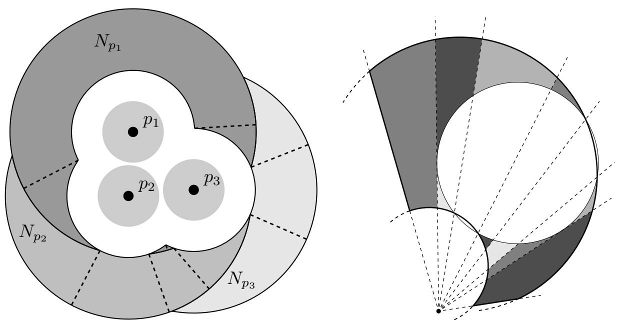


Figure 4: *Scallop-MDS's decomposition. Left, the next sample is taken from an advancing front around prior samples, in nonconformance to the strict Poisson-disk process. Right and left, the front is decomposed into shaded scalloped regions. Inverse transform sampling is uniform by area without rejection. Images courtesy Dunbar and Humphreys [DH06a, DH06b].*

Scallop-MDS, "A Spatial Data Structure for Fast Poisson-Disk Sample Generation" [DH06a] uses an advancing front to generate a maximal disk sampling (but not a Poisson-disk one) in deterministic $O(n\log n)$ time. The next sample is chosen from an annular neighborhood around previously accepted samples. Annuli are split into scalloped sectors bounded by two rays; see Figure 4. The next scalloped region is chosen uniformly by area. A sample is generated within it using an inverse CDF transform, without rejection. A maximal distribution is achieved.

The CDF is the area A of a scalloped sector. A closed form for A is not provided, let alone its inverse. Instead the area is partially described by an integral whose solution involves six trigonometric functions and two inverse trigonometric functions. The paper notes that the full formula also involves two simple variations of this description, but does not give the formula for those variations. The inverse A^{-1} is calculated numerically with a binary search over the forward transform A. Binary search converges slowly and contributes a factor of b to the precision complexity.

While Scallop-MDS is fast and generates useful blue-noise, the community has noted [MEA*18] that the output is not identical to a maximal Poisson-disk distribution. The next sample always arrives in a scalloped region. In the true Poisson-disk sampling process the next sample may arrive outside all scalloped regions but overlap and reduce them. Put another way, Scallop-MDS oversamples near the $4r$ distance around samples.

The effect on the output distribution is visible. In MPS distributions, point pairs likely have smaller distance between them than with Scallop-MDS. One can see this in the light second ring at $4r$ in Dunbar & Humphreys's Figure 5(d) compared to their 5(a). [DH06a] Thus our terminology is Scallop-MDS for "Maximal Disk Sampling" rather than MPS for "Maximal Poisson-disk Sampling." Spoke-Darts [MEA*18] modifies how samples are chosen in annuli to come closer to desirable blue noise, and does not claim to be a Poisson-disk distribution.

GridInner-MPS, "Linear-Time Poisson-Disk Patterns" [JK11] generates a maximal Poisson-disk distribution in deterministic $\mathrm{O}(n)$ time, with a caveat about a numerical/geometric approximation issue associated with the curved boundaries of scooped-squares. (Our contribution is removing this caveat.) GridInner-MPS's key insight is calculating and maintaining the *arrival time* of each candidate sample explicitly, rather than just their order of arrival. The algorithm starts by generating a background grid of squares. Within each square, a candidate sample is generated uniformly by area, with an arrival time expovariate in the area of the square. Any sample that arrives earlier than any nearby sample can be accepted; the order in which these are accepted does not matter. When a sample is accepted, its disk may cover some candidate samples (with later arrival times) in nearby squares. A covered candidate is resampled and replaced by an uncovered one in its scooped-square. Its arrival time is incremented by a time expovariate in the scooped-square area, congruent to the Poisson-disk process. Because the neighbor template (Figure 3 left) is constant size, all of these operations only happen a constant number of times per grid square. The crux of linear runtime is the ability to "sample uniformly from the free space of a grid square in O(1) time." [JK11]

In theory the scooped-square free space is a grid square with some constant number of disks subtracted, just as with GridOuter-MPS. The paper states, "In our reference implementation, we use a constructive planar geometry library and approximate disks with polygons for simplicity. . . ." The implementation [Jon13b] uses regular polygons that contain and approximate Poisson disks. There is an accuracy–runtime tradeoff in this approximation. Let s be the number of polygon sides. The cost of constructing and triangulating a scooped-square is $\mathrm{O}(s)$. The maximum distance d between a polygon point and the true disk is $d = 1 - \cos(\pi/s) = \Theta(s^{-2})$. For b bits of accuracy, we need $d < 2^{\mathrm{O}(-b)}$. Hence $s = 2^{\mathrm{O}(b)}$.

The paper claims that in principle one may use the spatial data structures of Scallop-MDS for calculating areas exactly and sampling from them deterministically. Our contribution is achieving this using chocks instead. How one could use scalloped sectors is not described in either paper and appears challenging because scooped-squares have different geometry than scalloped sectors.

Simple-MPS, "A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions" [EMP*12] has the fastest empirical performance of any known 2D MPS method. It is based on "Poisson Disk Point Sets By Hierarchical Dart Throwing" [WCE07]. These methods dispense with calculating the geometric intersection between circles and squares. Instead they use a quadtree of squares that are an outer approximation to the uncovered region. Rejection sampling is performed in squares. Squares are refined to improve the approximation and reduce rejections; see Figure 5.

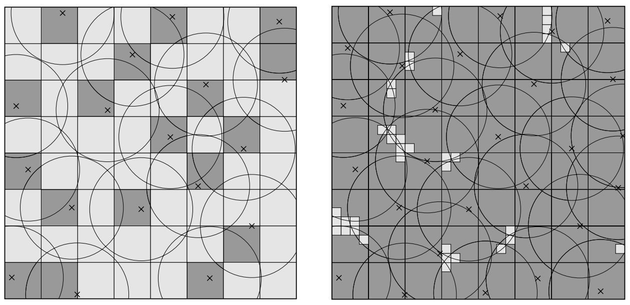


Figure 5: *Simple-MPS's outer approximation is coarse but fast. Right, candidates are generated sequentially in a background grid of squares. A candidate outside all prior disks is accepted as a sample. Dark squares are already covered by a disk and do not generate candidates. Left, squares are refined, and squares outside any single disk (light) are the background grid in the next iteration. Images courtesy Simple-MPS [EMP*12].*

Hierarchical Dart Throwing [WCE07] keeps track of the number of observed rejections in each square, and maintains squares of different sizes, for $\mathbb{E}(n \log n)$ runtime. Simple-MPS performs little bookkeeping and refines all squares at once after $\mathrm{O}(n)$ throws; this improves the empirical runtime to linear in n.

Empirical runtimes have negligible dependence on b because the number of squares per iteration tends to decline as a geometric series. However, the worst-case for Simple-MPS has disastrous b dependence. Suppose after the first sample is accepted the algorithm is unlucky and always selects a candidate inside a disk. This continues until each top-level square is refined into $2^{\mathrm{O}(b)}$ squares of side-length $2^{\mathrm{O}(-b)}$. Each square is thus a single floating-point position, which ensures no more rejections. The software is likely to run out of memory in this case.

The main price of using an outer approximation and rejection sampling in Simple-MPS, Voronoi-MPS and GridOuter-MPS is having expected (and not deterministic) running time.

2.2. Contribution

We implement the first $\mathrm{O}(n)$ deterministic time algorithm for maximal Poisson-disk sampling up to machine precision, for 2D planar rectangular domains. It is a variant of GridInner-MPS that represents the uncovered region exactly. There are two main changes. First, we show how scooped-squares can be partitioned into $\mathrm{O}(1)$ triangles and chocks. Second, and more importantly, we provide formulas for the area of chocks, and inverse calculations for sampling uniformly from within chocks, with zero chance of rejection, in $\mathrm{O}(M(b) \log b)$.

These same changes can replace the outer approximations and rejection sampling from Voronoi-MPS and GridOuter-MPS to turn these expected $\mathbb{E}(n \log n)$ algorithms into deterministic $\mathrm{O}(n \log n)$ ones, in principle. We have actually done so for GridOuter-MPS.

We provide open source code. We show its runtime is indeed linear, and within an order of magnitude of the fastest algorithm in practice, Simple-MPS. There are no third-party library dependences. For example, dynamic precision geometric libraries are not needed because the constructions are designed to be topologically robust to floating point error; see Section 4.2 and
`https://github.com/samitch/DeterministicMPS`

3. Chock Geometry

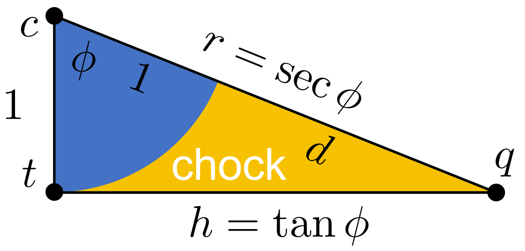


Figure 6: *A* chock *is bounded by a circle, radial ray, and tangent.*

The shape of a chock is uniquely defined by the angle ϕ it subtends at the circle center c; see Figures 1 and 6. The circle center is a Poisson-disk sample. For simplicity we assume the disk radius is 1. The relevant points are the circle center c, the tangent point t, and the ray point q. Define $\phi = \angle tcq$. Triangle $\triangle tcq$ has side lengths $|ct| = 1$, and $|tq| = h = \tan\phi$, and $|cq| = r = 1 + d = \sec\phi$.

4. Algorithm

Figure 1 lower illustrates the Deterministic-MPS pipeline. See Figure 7 for an example run. Algorithm 1 gives our pseudocode. The crux is the resampling:

Line 21. Construct scooped-square geometry.
Line 25. Trim chocks, create chocks next to disks.
Line 26. Triangulate remaining polygons.
Lines 27 and 28. Sample uniformly from triangles and chocks.

Runtime analysis. Let n be the output number of samples, and $\mathcal{G}$ the grid. Each grid square can only fit one Poisson-disk sample, and an area argument shows that at least a constant fraction of grid squares generate an accepted disk, so $n = \Theta(|\mathcal{G}|)$. (We observe $2.87n \approx |\mathcal{G}|$ for MPS.) To show the runtime is linear in the size of the grid, observe that everything in each grid square is constant size and occurs only a constant number of times. Each square is only resampled when a neighbor square's sample is accepted, which can happen at most 9 times (Lemma 9 in GridOuter-MPS [EPM*11]). When a square is resampled, the scooped-square has at most 9 bounding disks, so its #chocks and #triangles is at most a constant. The count of antecedent (earlier) squares is only updated when one of its neighbor squares is resampled. There are 20 neighbor squares, each resampled at most 9 times, giving at most 180 updates. Memory is also linear in n.

4.1. Locally-early Samples

For a candidate sample arriving at a given time, if there is no possibility that a sample can arrive inside its disk at an earlier time, it is *locally-early* and can be accepted. The algorithm can process locally-early samples in any order and still conform to the Poisson-disk process.

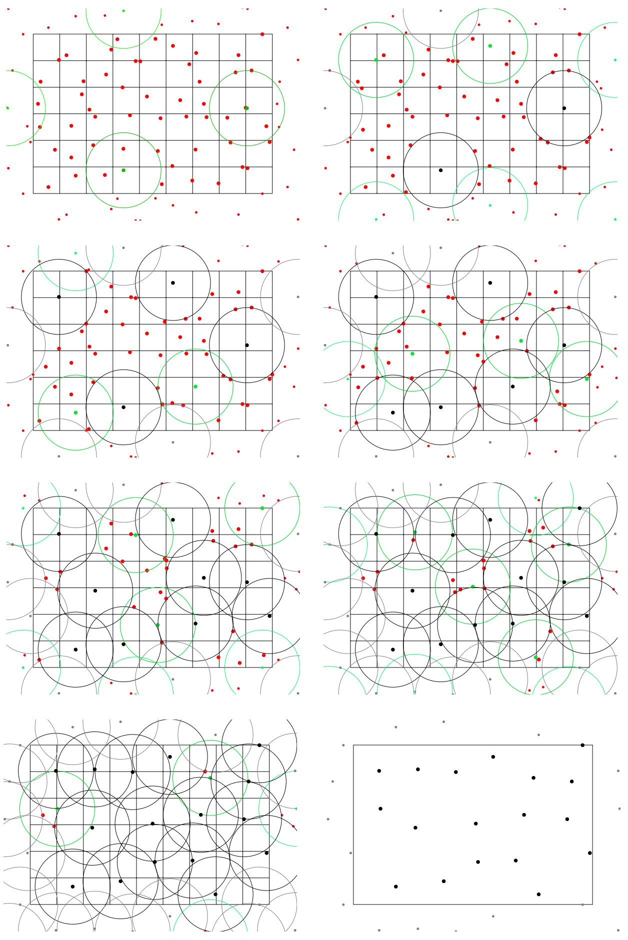


Figure 7: *Deterministic-MPS example run snapshots. Green candidate samples are locally early and in the queue to be accepted. Red candidates are not locally early; they either will become locally early or be resampled. Black disks are already accepted. Squares with no samples are completely covered by disks, with an empty scooped-square. The faint circles and points show wrapping around the periodic boundary, which is optional.*

The *square-neighbor* test says a candidate is locally-early if it is earlier than all of the candidates in its square neighbors. This is sufficient and fast, but we find more locally-early candidates if we only consider squares that intersect the candidate's disk, the *disk-neighbor* test. A heuristic that saves 10% is to do find and accept candidates with no earlier disk-neighbors in seven passes over the entire grid, then switch to square-neighbors and track antecedents.

4.2. Scooped-square Construction

We construct scooped-squares using our own geometric primitives in standard floating point arithmetic. These are topologically robust without the use of geometric libraries or dynamic precision,

Algorithm 1 Deterministic-MPS: maximal Poisson-disk sampling

```
Require: Rectangular grid $\mathcal{G}$ of whole grid squares
Require: Flag if domain is periodic: True or False
Ensure: Maximal Poisson-disk sampling of rectangle
 1: function DETERMINISTIC-MPS($\mathcal{G}$)
 2:     // Initialize Grid $\mathcal{G}$
 3:     for $g \in \mathcal{G}$ do
 4:         $g$.point $= (u,v)$ uniform random in square
 5:         $g$.time $= Ae^{-Aw}$, rand $w$, expovariate in area $A(g)$
 6:         $g$.scooped-square = square polygon $g$
 7:     Global pre-pass heuristic
 8:     // Find locally-early squares
 9:     for $g \in \mathcal{G}$ and $h \in$ neighbors$(g)$ do
10:         increment #antecedents of $g$ or $h$, whichever is later
11:     for $g \in \mathcal{G}$ do
12:         EarlySquares.add($g$ if no antecedents)
13:     // Accept samples and update
14:     repeat
15:         $g$ = EarlySquares.pop()                    ▷ any order
16:         accept $g$.point as Poisson-disk sample
17:         for $h \in$ neighbors$(g)$ do
18:             decrement h.antecedents         ▷ g no longer blocks h
19:             // resample candidates covered by disk($g$.point)
20:             if $h$.point $\in$ disk($g$.point) then
21:                 $h$.scooped-square $-=$ disk($g$.point)
22:                 if $h$.scooped-square is empty then
23:                     $h$.time $= \infty$
24:                 else
25:                     trim chocks from $h$.scooped-square
26:                     triangulate remaining polygon
27:                     pick $U \in \{$chocks, triangles$\}$ by area
28:                     sample $h$.point $\in U$ uniform by area
29:                     $h$.time $+=$ expovar( A($h$.scooped-square) )
30:                 for $s \in$ neighbors$(h)$ do
31:                     if $h$ is later than $s$, but used to be earlier then
32:                         increment h.antecedents
33:                         decrement $s$.antecedents
34:                         EarlySquares.add( $s$ if no antecedents )
35:             EarlySquares.add( $h$ if no antecedents )
36:     until EarlySquares == empty
```

exploiting the convexity and size of the square and disks. We represent a scooped-square by the ccw (counter clockwise) ordered loops of segments bounding the uncovered regions. Segments are circular arcs or lines. Initially, a scooped-square is just a square.

To subtract a disk from a scooped-square, we subtract the disk from each segment, then split the sequence of segments into loops at each instance of the disk. To robustly subtract the disk from a line segment, we calculate the points of the disk intersected by the corresponding grid line, if any. It is easy to ensure the two points are ordered correctly, even in the numerically-degenerate cases of near tangency, because the grid line is axis aligned. Some cases of the ordering of the arc and line segment endpoints are illustrated in Figure 8. Subtracting a disk from a circular arc segment is analogous, where instead of the points' order along a coordinate axis, we use their angular order around the circle's center. Because accepted Poisson-disk centers do not lie in other disks, case Figure 8b cannot happen for arc-arc intersections. We also check for numerically-missed intersections, such as when a segment lies strictly interior to the disk, and the next segment lies strictly exterior.

The resulting segments create a new loop, which we now split into connected components; Figure 9 shows a simple example. A subsequence terminated by two segments from the same disk is a connected component. Multiple such sequences result in multiple connected components. This follows from observing that a disk can only bound a connected component once, in a single segment. For a component, we replace the two disk instances with a single one, using the endpoint of the first instance's cw endpoint and the second instance's ccw endpoint.

4.3. Trim Chocks

We partition a scooped-square by trimming off chocks, leaving a polygon bounded by straight segments. We first split some segments to ensure that chocks are disjoint and interior to the scooped-square; see Figure 11. Each final d-segment generates two chocks; see Figure 10 left.

Since a square diagonal is 1, the largest circular arc in a scooped-square has angle $\pi/3$. Since arcs are split into at least two chocks, the maximum chock angle ϕ is $\pi/6$. In Section 4.5 we will see that for $\phi \leq \pi/4$ the numerics allow us to sample precisely up to machine precision. In other contexts, if a chock is deemed too large it is simple to split it as finely as desired; see Figure 10 right.

4.4. Triangulation

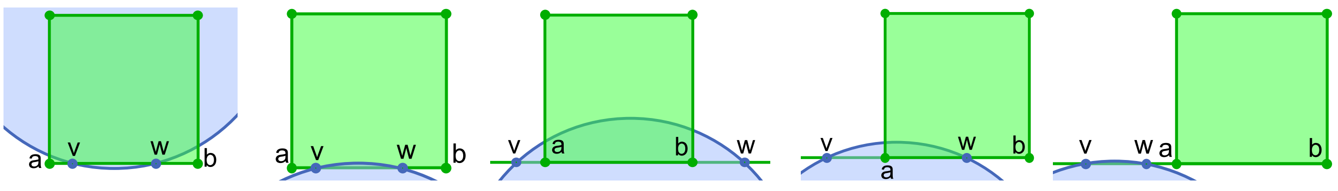
We triangulate the polygons; Figure 12 gives examples. Ear clipping is robust and does not introduce additional points. Let $p_0, p_2, \dots p_m$ be the loop vertices. The principle is to recursively clip a triangle $\triangle p_{i,j,k}$, removing p_j from the loop. Any non-reflex triangle that does not contain another polygon vertex can be clipped, and there are always at least two of them. [Mei75] Checking the angle and emptiness is done with cross products.

Partitioning Alternative. It may be possible to partition into triangles and chocks directly, without first trimming. Some extension of "Linear-size Nonobtuse Triangulation of Polygons" [BMR95] might work well because it is based on circle-packing domains bounded by straight lines and circular arcs. Since Poisson disks can overlap, their triangulation would have to be extended to the case that the radical axis is not tangent to the circles.

4.5. Uniform Sampling from a Chock without Rejection

The available area in a grid cell is represented by some triangles and chocks. To select a point uniformly at random, we first pick one of these triangles or chocks uniformly by area, which requires a formula for the area of a chock, Equation (1). To sample uniformly within a chock, we need to calculate the inverse area, Equations (2) to (4); and the inverse radius by swept area, Equation (5).

Area of a Chock. See Figure 6. The area of a chock is the area of



(a) $\overline{av}, \widehat{vw}, \overline{wb}$ **(b)** $\overline{av}, \widehat{vw}, \overline{wb}$ **(c)** $\emptyset$ **(d)** $\widehat{vw}, \overline{wb}$ **(e)** $\overline{ab}$

Figure 8: *Representative cases of the result of subtracting a disk from an axis-aligned line segment. The result is the possible segments bounding the uncovered region of the square, in ccw order: we may have 3, 2, 1, or 0. We can keep the order of the endpoints topologically consistent just with floating point operations. We re-chain the segments together to build the boundary. In cases like Figure 8a, the arc splits the scooped-square into multiple connected components. In cases like Figure 8d, $\widehat{vw}, \overline{wb}$, the re-chaining replaces the endpoint v with the endpoint computed when subtracting the disk from the vertical grid segment.*

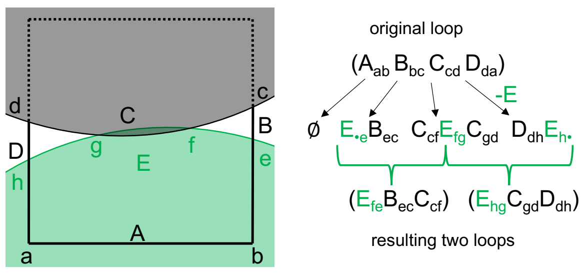


Figure 9: *Example updating a scooped-square. The current loop of segments is $A_{ab}B_{bc}C_{cd}D_{da}$, where A has endpoints a and b, etc. We subtract disk E from each segment, and subsequences between instances of E form components.*

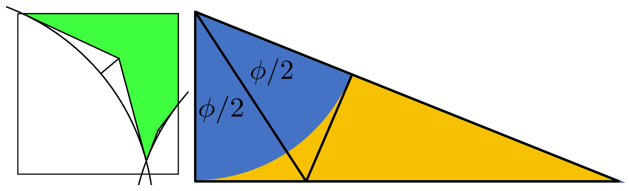


Figure 10: *Left, each circular arc bounding a scooped-square contributes two chocks. Right, a large-angled chock may be split into two half-angle chocks and a right triangle. This may be done recursively to make the chock regions as small as desired.*

the triangle $\triangle cqt$ minus the area of the sector $\lhd\phi$: $A = Area(⊾) - Area(\lhd)$, where $Area(⊾) = h/2 = (\tan\phi)/2$ and $Area(\lhd) = \phi/2$.

$$A(\phi) = (\tan\phi - \phi)/2 \tag{1}$$

Uniform Random Sampling from a Chock. See Figure 13. We use inverse transform sampling [Dev86] to select a point uniformly. We select a uniformly-random area fraction of the whole chock, invert to find the angle of the sub-chock with that area, then sample along that angle's ray proportional to squared radius. That is, let $u \in [0,1]$ be a uniform random variable, and A be the area of the entire chock. We seek ϕ_s such that $A(\phi_s) = uA(\phi)$.

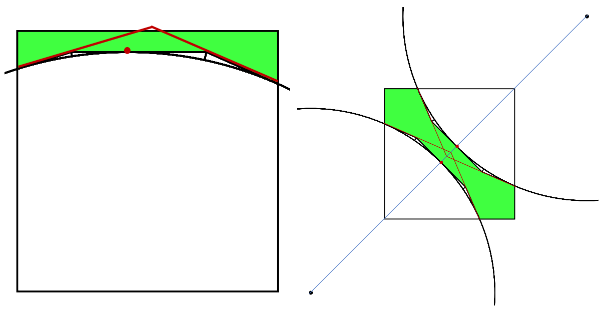


Figure 11: *Right, if a chock has the potential to overlap with some other circle's chock (red), then we split the circular arcs at the closest points. Left, a disk too close to a square edge is analogous.*

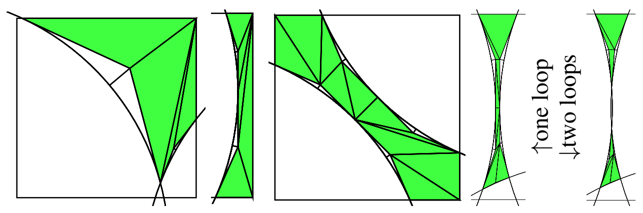


Figure 12: *Some example triangulations.*

Inverse Area of a Chock. We must invert $\tan\phi - \phi$. While we do not have a mathematically closed form for this inverse, we do have a *numerically* closed form. (Note tan itself is numerically closed in the same way, often computed using logarithms.) It is exact up to machine precision for $[0, A(\pi/4)]$, exceeding the needed domain.

The initial guess for the Newton iterate is

$$\phi_0 = \sqrt[3]{6uA} \tag{2}$$

based on the Maclaurin series

$$\tan\phi - \phi \approx \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \ldots$$

The derivative of the area is $(1/\cos^2\phi - 1)/2$, simplifying to

$$A'(\phi) = (\tan^2\phi)/2$$

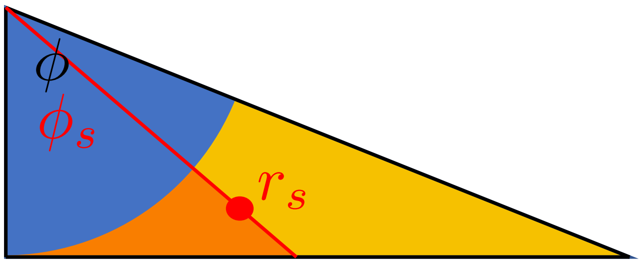


Figure 13: *Uniform-area sampling within a chock. First, select* ϕ_s *so its red sub-chock is a uniform random fraction of the entire yellow chock, i.e. dependent on* $\tan\phi - \phi$. *Second, select* r_s *to be a uniform random fraction of the elemental swept area, i.e. dependent on* r^2. *Our sample is the red point* (ϕ_s, r_s).

Thus the Newton iterate is

$$\phi_{i+1} = \phi_i - \frac{A(\phi_i) - uA}{A'(\phi_i)} = \phi_i - \frac{\tan\phi_i - \phi_i - 2uA}{\tan^2\phi_i} \tag{3}$$

The method is numerically stable, with no need for branches or checks. Using double precision, five iterations suffice to compute $\phi = A^{-1}(uA)$ to 15 digits of absolute accuracy for any angle up to $\pi/4$. Recall radii are 1, so 15 digits is also relative error. Thus our sample is on the radial ray of the chock with angle ϕ_s given by

$$\phi_s = \phi_5. \tag{4}$$

Higher precision requires more iterations, but this cost can be offset by using low precision in early iterations. [Bre76, BZ10] For b bits, the complexity is the sum of a geometric series, and bounded by $M(b)\log b$. (Recall $M(b)$ is the cost of b-bit multiplication.) This is merely a constant multiple of the cost of $\tan\phi$.

Random Radius. We now have an angle ϕ_s for our sample, and it remains to select its radius r_s, its distance along ray ϕ_s. Again, we use the inverse transform, with $\mathrm{CDF}(r_s) = \int_1^{r_s} r\,dr$. The ray segment has extent $[1, \sec\phi]$, thus $\int_1^{\sec\phi} r\,dr = (\sec^2\phi - 1)/2 = (\tan^2\phi)/2$ is the entire segment. We select $u \in [0,1]$ uniformly, and invert to find $r_s : \int_1^{r_s} r\,dr = u(\tan^2\phi)/2$. The solution is

$$r_s = \sqrt{u\tan^2\phi_s + 1}. \tag{5}$$

Note $\tan^2\phi + 1 = \sec^2\phi$ so the upper limit for $u = 1$ is correct.

4.6. Variants

Sequential-MPS (implemented) is Deterministic-MPS without arrival times, which is equivalent to GridOuter-MPS with approximations and rejections replaced by our exact scooped-squares geometry and chock sampling. At each step, Sequential-MPS selects a random scooped-square uniformly by area and generates a uniform sample within it.

ChockSubdivision-MPS. Instead of sampling from chocks, we could sample only from triangles. If the random-area chosen falls inside a chock, recursively split it into a right triangle and two chocks, and continue like a binary search. Perhaps this is simpler than the inverse area using Newton's method, but it has worse precision complexity, O(b). (We assume $b\log b$ can be avoided, here and in Scallop-MDS, by computing the area only to the precision required for determining which branch of a split to take.)

5. Implementation and Reproducibility

The open source C++ code is available from
`https://github.com/samitch/DeterministicMPS`

It includes methods to reproduce the experimental results and figures in this paper. It also includes Sequential-MPS and a 2D version of Simple-MPS.

Limitations. The software only handles rectangular domains composed of whole background squares. For periodic domains (i.e. tori), there must be at least three squares in each dimension to avoid the special case of a disk overlapping with itself. These are limitations of the implementation only. In principle the algorithm extends to 2D polygons with holes.

Trigonometric Functions. In chock-sampling the only trigonometric function needed was tan. We experimented with using the Taylor series expansion of $\tan\phi - \phi$ exclusively, but decided using C++ `std::tan` function from `<cmath>` was simpler and better. Outside the sampling subroutine, the implementation also uses `atan2(y,x)` for converting Cartesian to polar coordinates.

5.1. Experimental Runtime

Our main contribution is Deterministic-MPS's theoretical O$(nM(b)\log b)$ runtime, but our experimental results show that it is also practical. While not as fast as Simple-MPS, recall Simple-MPS has average performance that is not theoretically guaranteed, and poor worst-case performance.

Experiments are performed on a Mac laptop and a Linux workstation: MacOS 11.6.2: 2.2 GHz dual-core Intel Core i7, 8 GB memory, manufactured circa 2015; Red Hat Enterprise Linux 7.8: 2.50GHz Intel Xeon E5-2670 v2, 40 processors, 65 GB memory.

We compare Deterministic-MPS, Simple-MPS, Voronoi-MPS, GridInner-MPS, and Sequential-MPS. We used the open source versions of Voronoi-MPS [Jon13a] in C, and GridInner-MPS [Jon13b] in Python. We implemented Simple-MPS and Sequential-MPS in our framework because the original sources for Simple-MPS and GridOuter-MPS are not publicly available.

On the Mac, Deterministic-MPS generates about 120k samples / second for most test sizes; see Table 2 and Figure 14. Deterministic-MPS is about 2× faster than Sequential-MPS; see Figure 15. This is because generating the random arrival time is fast and Sequential-MPS must recalculate areas every time a sample disk reduces a scooped-square, whereas Deterministic-MPS only needs to do so when that covers a candidate. Simple-MPS is about 2–3× faster than Deterministic-MPS; see Figure 16. Deterministic-MPS is about 30× faster than Voronoi-MPS for 1M points; see Figure 17. Deterministic-MPS is 630× faster than GridInner-MPS; see Figure 18. This is even after we improved the GridInner-MPS runtime by 8× by increasing the square diagonal to the minimum separation between samples. The GridInner-MPS code is adequate

	Determ.	Sequ.	Simple	Voronoi	Inner
Mac	120k	61k	270k	3.8k	190
Linux	91k	40k	300k	11k	68

Table 2: *Runtime efficiency. Samples per second when generating 1M samples. GridInner is slow due to Python.*

to demonstrate correctness and linear time, but is in Python and a reimplementation could be much faster.

Figures 14 to 16 report averages over five runs. Mac runtime varied significantly when rerunning for a second or third time in a row. We speculate this is due to the memory hierarchy and CPU burst mode. For all the methods, the relative performances on Linux are different, due to its different CPU and memory characteristics. For GridInner-MPS, the Python 3 versions and virtual environments differ on the two platforms. For Voronoi-MPS and GridInner-MPS, it's possible that the libraries differ by platform as well.

Memory. We profiled and tuned for fast runtime, not low memory. Deterministic-MPS produces at least 4M samples, Sequential-MPS 2M, Simple-MPS 10M, Voronoi-MPS 2M, and GridInner-MPS 2M, before experiencing paging issues on the 8 GB Mac.

Code Size. Deterministic-MPS is in C++, self-contained, and relatively large: 4500 lines, plus 2900 lines of tests and examples. Our Simple-MPS is 700 lines on top of that. True to its nickname, Simple-MPS took us only a day to reimplement, given our extant grid, disk, and point infrastructure. Our Sequential-MPS is an additional 900 lines. Voronoi-MPS is 1400 lines in C, plus the GNU Triangulated Surface Library, GTS. GridInner-MPS is only 300 lines in Python, plus the python package Polygon.

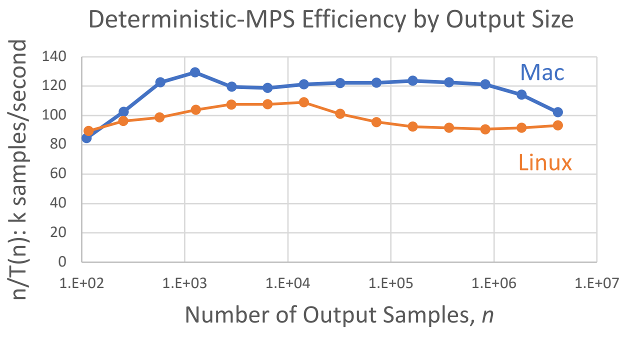


Figure 14: *Deterministic-MPS runtime efficiency. Theoretical runtime is $O(n)$; horizontal would indicate $O(n)$ in practice.*

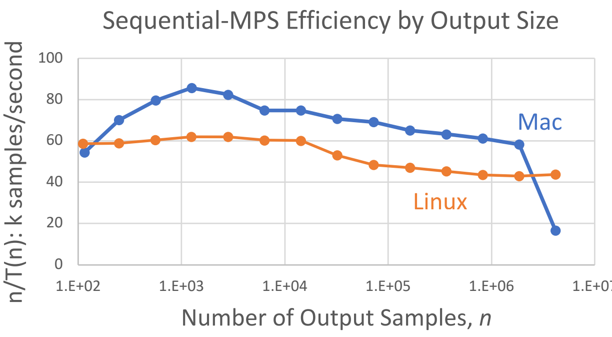


Figure 15: *Runtime efficiency of Sequential-MPS, our $O(n \log n)$ deterministic time version of GridOuter-MPS with chock sampling instead of rejection sampling.*

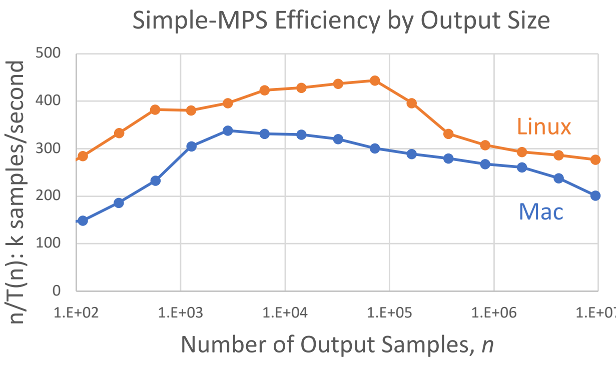


Figure 16: *Runtime efficiency of our reimplementation of Simple-MPS. Linear time would be a horizontal line. The number of operations is linear, but the code slows down waiting for memory access. (This can also be seen in the original authors' data.) The effect of memory slowdown is strongest in this method because it does so little computation.*

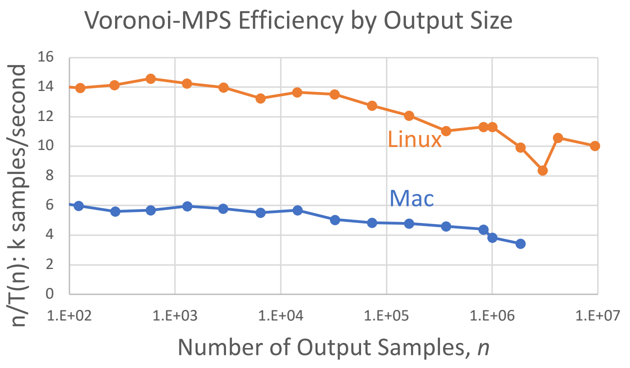


Figure 17: *Voronoi-MPS empirical runtime efficiency. Theoretical runtime is $\mathbb{E}(n \log n)$.*

5.2. Output Quality

The MPS distribution quality is evaluated by spectral analysis using "Point Set Analysis," PSA [Sch13, SD11, HSD13]; see Figure 19. This provides numerical verification that Deterministic-MPS produces maximal Poisson-disk samplings, and not some other distribution. Our code internally verifies saturation and separation in the final distribution. Figure 1 right shows 4000 random points in a $\phi = \pi/4$ chock, and provides visual verification that our numerical sampling procedure is uniform by area within a chock. After a sample is generated we verify that it is in the square and outside any disk: the tolerance is `__DBL_EPSILON__` for squares and twice

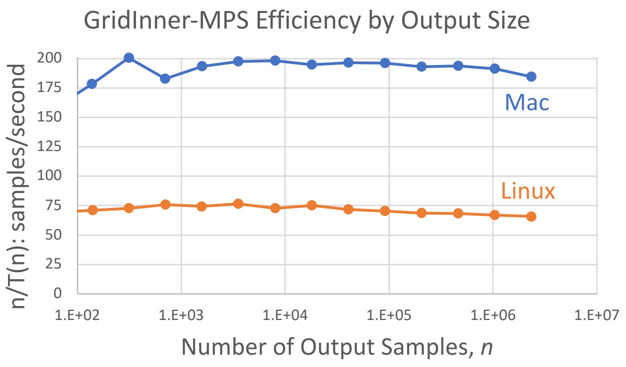


Figure 18: *GridInner-MPS empirical runtime efficiency. Theoretical runtime is $O(n)$. Horizontal would indicate $O(n)$ in practice. Note the y-axis scale is samples / second, and for the other runtime figures it is* thousands *of samples / second.*

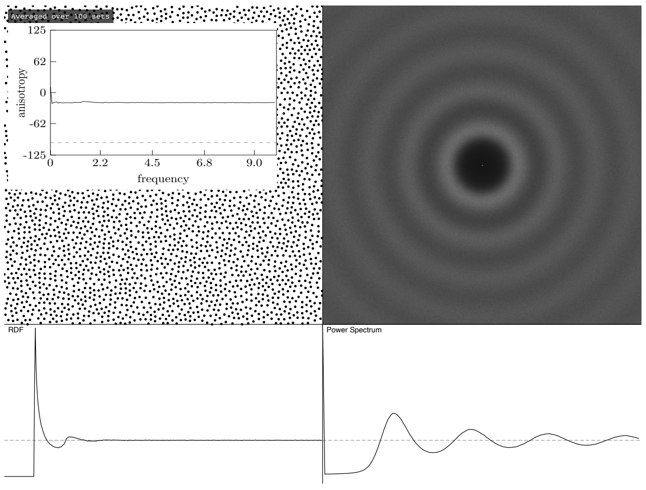


Figure 19: *PSA [Sch13] analysis of the output of 100 Deterministic-MPS runs over a 100×100 periodic grid, about 3500 points each. RDF is the 1D Radial Distance Function, the histogram of distances between all pairs of points. Its Fourier transform is the 1D Power Spectrum. Upper right is the 2D Fourier transform. Upper left is anisotropy, and an example point set.*

that for disks. This has been verified on billions of tests, and some hand-built degenerate cases such as four disks meeting at a point.

6. Open Problems and Conclusion

Domains. We would like to apply the algorithm to more types of domains. Irregular polygonal domains appears straightforward in principle, but the current implementation exploits grid squares being axis-aligned. Planar domains with non-straight boundaries may require inverse transforms for additional shapes. Embedded surfaces would also require new inverse transforms. For higher-dimensions, the challenges include constructing robust geometric primitives for cubes with spheres subtracted, and developing the math for the higher-dimensional analogues of chocks.

Parallelism. We would like a parallel algorithm that takes constant time per sample in the worst case. PixelPie [IYLV13] is GPU-based and empirically has very fast average performance. It does rejection sampling similar to Simple-MPS, but based on pixels instead of boxes. It generates candidate samples centered at some random pixels, with random depth, and uses occlusion culling to determine which can be accepted. The next iteration considers just the remaining uncovered pixels. We would like a spatial statistics proof that if one throws a number of darts proportional to the area of the remaining domain, then a constant fraction of them are accepted on average. This may be sufficient to prove that Simple-MPS in serial has expected linear time, and PixelPie has expected constant time. (The theoretical worst-case performance for both is poor.)

Distributions. While we have focused on exactly reproducing the Poisson-disk process, a definition of the Poisson-disk output spectrum might enable the discovery of better processes for producing such output. Other distributions may be more valuable than Poisson-disk distributions anyway. One variant is step blue noise [HSD13], where the power spectrum is a step function from zero to a constant. (Recall Poisson-disk samplings have a spike in the RDF and associated oscillations in the power spectrum; see Figure 19.) A general variant is importance sampling or sampling of vector fields. There are many optimization procedures [DGBOD12, HSD13, ZH16, CP21] to move 2D points, disks, or other local kernels to match a user-specified spectrum. Other techniques [MREB12] generate such samples a priori, but are not deterministic time.

It may be possible to extend Deterministic-MPS to non-uniform disk radii for these distributions. We may partition the Voronoi regions of Voronoi-MPS into chocks and triangles as before. Power cells and splitting chocks may be helpful. Another option is to use our scoop-chock-triangulate method on the dynamically-sized squares of “Variable Radii Poisson-Disk Sampling” [MREB12].

Summary. We provided the math for uniform sampling from a chock without rejection. This enabled a linear deterministic time algorithm for generating maximal Poisson-disk samplings, with minimal dependence on machine precision. We proved this theoretically, and demonstrated it in practice in open source code.

Acknowledgements

I thank Thouis Ray Jones for suggesting this problem to me in 2012, and wish it had been less than 9 years until I discovered the chock inverse transform sampling solution! I thank him for explaining the details of his prior publications and implementations, and discussing many technical points of the present paper. I apologize for “Efficient Maximal Poisson-Disk Sampling” claiming to have the *first* expected $\mathbb{E}(n \log n)$ algorithm for maximal Poisson-disk sampling; that distinction belongs to “Efficient Generation of Poisson-Disk Sampling Patterns” [Jon06], 5 years earlier.

I thank the anonymous SGP reviewer for pointing out that the recursive subdivision of chocks into a triangle and two smaller chocks in Figure 10 can be used to generate an MPS distribution without the need for an inverse area transform.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Applied Mathematics Program. Sandia National Laboratories is a multi-program laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[BMR95] Bern M., Mitchell S., Ruppert J.: Linear-size nonobtuse triangulation of polygons. *Discrete & Computational Geometry 14*, 1 (1995), 411–428. 6

[Bre76] Brent R. P.: The complexity of multiple-precision arithmetic. *The Complexity of Computational Problem Solving* (1976), 126–165. arXiv:1004.3608. 2, 8

[BZ10] Brent R. P., Zimmermann P.: *Modern Computer Arithmetic*, vol. 18. Cambridge University Press, 2010. 2, 8

[CP21] Cammarasana S., Patanè G.: Kernel-based sampling of arbitrary signals. *Computer-Aided Design 141* (2021), 103103. doi:10.1016/j.cad.2021.103103. 10

[Dev86] Devroye L.: *Non-Uniform Random Variate Generation*. Springer, New York, NY, 1986. doi:10.1007/978-1-4613-8643-8. 2, 7

[DGBOD12] De Goes F., Breeden K., Ostromoukhov V., Desbrun M.: Blue noise through optimal transport. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–11. 10

[DH06a] Dunbar D., Humphreys G.: A spatial data structure for fast Poisson-disk sample generation. In *SIGGRAPH '06* (2006), pp. 503–508. doi:10.1145/1179352.1141915. 3, 4

[DH06b] Dunbar D., Humphreys G.: *Using Scalloped Sectors to Generate Poisson-Disk Sampling Patterns*. Tech. Rep. CS-2006-08, University of Virginia, 2006. 3

[DW85] Dippé M. A. Z., Wold E. H.: Antialiasing through stochastic sampling. In *SIGGRAPH '85* (1985), pp. 69–78. doi:10.1145/325334.325182. 2, 3

[EMP*12] Ebeida M. S., Mitchell S. A., Patney A., Davidson A. A., Owens J. D.: A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Computer Graphics Forum 31*, 2 (May 2012), 785–794. doi:10.1111/j.1467-8659.2012.03059.x. 3, 4

[EPM*11] Ebeida M. S., Patney A., Mitchell S. A., Davidson A., Knupp P. M., Owens J. D.: Efficient maximal Poisson-disk sampling. *ACM Transactions on Graphics (TOG) 30*, 4 (July 2011), 49:1–49:12. doi:10.1145/1964921.1964944. 2, 3, 5

[HSD13] Heck D., Schlömer T., Deussen O.: Blue noise sampling with controlled aliasing. *ACM Trans. Graph. 32*, 3 (jul 2013). doi:10.1145/2487228.2487233. 9, 10

[IYLV13] Ip C. Y., Yalçin M. A., Luebke D., Varshney A.: PixelPie: Maximal Poisson-disk sampling with rasterization. In *Proceedings of the 5th High-Performance Graphics Conference* (2013), pp. 17–26. 10

[JK11] Jones T. R., Karger D. R.: Linear-time Poisson-disk patterns. *Journal of Graphics, GPU, and Game Tools 15*, 3 (2011), 177–182. 3, 4

[Jon06] Jones T. R.: Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools 11*, 2 (2006), 27–36. 2, 3, 10

[Jon13a] Jones T. R.: Code for "Efficient generation of Poisson-disk sampling patterns". https://github.com/thouis/fast-poisson-disk, 5 February 2013. 8

[Jon13b] Jones T. R.: Example implementation for "Linear-time Poisson-disk patterns". https://github.com/thouis/linear-poisson-disk, 5 February 2013. 4, 8

[MEA*18] Mitchell S. A., Ebeida M. S., Awad M. A., Park C., Patney A., Rushdi A. A., Swiler L. P., Manocha D., Wei L.-Y.: Spoke-darts for high-dimensional blue-noise sampling. *ACM Trans. Graph. 37*, 2 (May 2018), 22:1–22:20. doi:10.1145/3194657. 3, 4

[Mei75] Meisters G. H.: Polygons have ears. *The American Mathematical Monthly 82*, 6 (1975), 648–651. 6

[MREB12] Mitchell S. A., Rand A., Ebeida M. S., Bajaj C.: Variable radii Poisson-disk sampling, extended version. In *Canadian Conference on Computational Geometry (CCCG)* (2012), vol. 5, Springer, pp. 1–9. 10

[Sch13] Schlömer T.: PSA point set analysis. Version 1.1, http://code.google.com/p/psa/, 2013. 9, 10

[SD11] Schlömer T., Deussen O.: Accurate spectral analysis of two-dimensional point sets. *Journal of Graphics, GPU, and Game Tools 15*, 3 (2011), 152–160. 9

[WCE07] White K. B., Cline D., Egbert P. K.: Poisson disk point sets by hierarchical dart throwing. In *2007 IEEE Symposium on Interactive Ray Tracing* (2007), IEEE, pp. 129–132. 4

[ZH16] Zhong Z., Hua J.: Kernel-based adaptive sampling for image reconstruction and meshing. *Computer Aided Geometric Design 43* (2016), 68–81. Geometric Modeling and Processing 2016. doi:10.1016/j.cagd.2016.02.013. 10