SAND2011-3606C

Maximal Poisson-Disk Sampling with Finite Precision and Linear Complexity in Fixed Dimensions

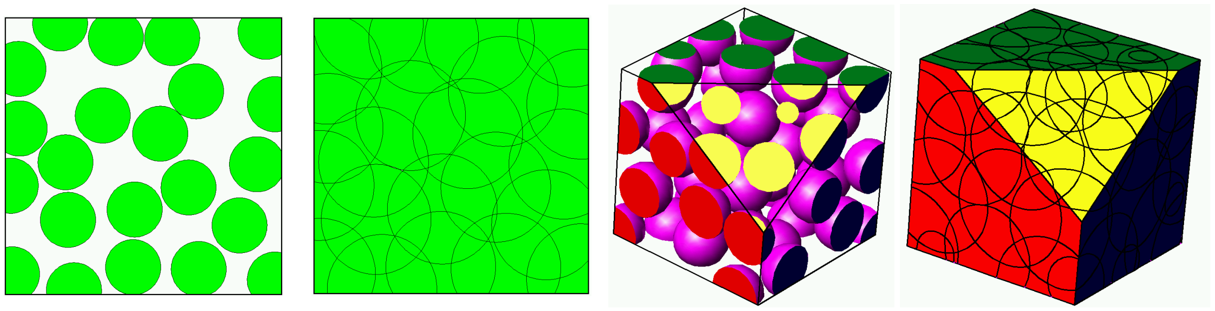


Figure 1: *Maximal Poisson-disk sampling of a unit box. Disks cover the domain and half-radius disks do not overlap. Our software runs in dimensions up to 5. In 2d we generate 1 million points in 12 seconds and 24 million points in 5 minutes.*

Abstract

We solve the d-dimensional maximal Poisson-disk sampling problem over the unit box in the finite precision model; the generated point cloud is bias-free, and maximal up to round-off error. Our algorithm is bias-free with $\Theta(n)$ time and memory for output size n and fixed dimension, d, and bits of precision, b. This improves on prior bias-free results by a $\log n$ factor. More significantly, our implementation shows that the method is practical in dimensions up to 4, and has been demonstrated in 5 dimensions. We use an order of magnitude less memory and time than the alternatives, and our results become more favorable as the dimension increases. No known methods scale well with dimension. Empirically our time is expected $\mathbb{E}(d5^{2d-1}n)$, and provably our memory is deterministic $O(d^2bn)$. Here n is doubly-exponentially dependent on d and the sampling radius r independent of any algorithm. Our runtime does not depend on the precision, and the extra memory required is only what is needed to store higher-precision numbers.

The algorithm proceeds through a finite sequence of uniform grids. The grids guide the dart throwing procedure and track the remaining area not yet covered by a disk. The top-level grid provides an efficient way to test if a candidate dart is disk-free. Like quadtrees, grid cell side-lengths drop in half at each stage. However, our grids are very efficient compared to quadtrees; since our grids are implicit they require little memory, and since they are uniform (only one quadtree-level is active) it is very simple and fast to sample from them uniformly.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling; F.2.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—Nonnumerical Algorithms and Problems

Keywords: Poisson disk, maximal, d-dimensional, linear complexity, sampling, blue noise

1 Introduction

Poisson-disk sampling is a random process for selecting points from a subdomain of a metric space. A selected point must be *disk-free*, at least a minimum distance, r, from any previously selected point. Thus each point has an associated disk of radius r that precludes the selection of nearby points. The selected points are called a *sample* or *distribution*. The sample is *maximal* if no point can be added to it. Euclidean distance is traditional but not essential. Selecting points should be uniform or *bias-free*; i.e. any point not covered by a prior disk has the same chance of being selected. This sampling process is crucial for many scientific fields. In computer graphics, Poisson-disk distribution are desirable because the inter-sample distances follow a certain power law that lacks low frequency noise. This blue-noise-like pattern yields good visual resolution for rendering, imaging and geometry processing [Pharr and Humphreys 2004]. In fracture mechanics, a tessellation based on a random point cloud is crucial to minimize the formation of preferential directions for crack propagations [Bolander and Saito 1998; Jirásek and Bazant 1995]. In Chemistry and statistical physics, Poisson-disk sampling is an example of random sequential adsorption [Dickman et al. 1991]. The performance and the quality of meshing methods is improved if the input random point cloud is maximal [Attali and Boissonnat 2004].

In this paper we present a maximal d-dimensional Poisson-disk sampling algorithm. Samples are guided by a finite sequence of uniform grids with increasing resolutions. The sequence length is bounded by the bits of precision. These grids are generated implicitly. Only a small subset of the possible cells at a given level actually exist, resulting in low storage requirements. The output distribution of n points is bias-free. It is maximal up to the round-off error, meaning that every point of the domain is within distance $r + \epsilon$ of a sample point, where ϵ is the numerical precision. To our knowledge, this is the first algorithm to solve this problem with time linear in n. Compared to other sampling methods satisfying the same conditions, we are about as fast and consume much less memory. This comparison tends to be in our favor as d increases. Our serial implementation samples one million points in about 10, 90, and 1400 seconds in 2D, 3D and 4D respectively. Using 2 GB of memory on a modern laptop, we were able to generate 24, 6, and 1.4 million samples in 2D, 3D and 4D respectively. We sampled 300k points in three hours in 5D. We believe these are the largest d-dimensional bias-free maximal Poisson-disk distributions ever reported in the literature.

Previous work During the last decades many methods were proposed to solve this problem. The classical dart throwing [Dippé and Wold 1985; Cook 1986] algorithm produces unbiased disk-free points, but requires unbounded time to achieve a maximal distribution. With this method, a bias-free dart is thrown and accepted if it does not violates the minimum distance bound with previous successful darts, other wise it is rejected. As more darts make it to the domain, the remaining parts of the domain valid for the sampling process ("voids") get smaller, decreasing the probability of being hit. In order to improve the efficiency, many methods were proposed to solve a relaxed version of the problem. Some methods sacrificed the bias-free condition: Wang tiles [Cohen et al. 2003; Lagae and Dutré 2005] and Penrose tiles [Ostromoukhov et al. 2004; Ostromoukhov 2007], for instance, do not target the whole domain uniformly when a dart is thrown; other methods have the same behavior [Mitchell 1987; Jones 2006; Dunbar and Humphreys 2006; Bridson 2007] for other reasons. For example, Jones picks a sampling sub-region based on the relative area of some Voronoi cells covering the domain. This introduces bias: a relatively large Voronoi cell, with higher selection probability, might contain a relatively small void. Wei's parallel sampling method [2008] used a sequence of multi-resolution uniform grids, but its output distribution is biased and only near-maximal. Bowers et al. [2010] use a similar phase-group-decomposition method to Wei but without a hierarchy.

More accurate methods have been proposed recently to satisfy the sampling conditions (distance at least r between samples, bias-free), while relaxing the maximal condition by discarding voids whose areas are about the size of round-off error. Two methods in this category track the remaining voids via quad-tree subdivision [White et al. 2007; Gamito and Maddock 2009]. Most recently, another method [Ebeida et al. 2011] satisfies the sampling conditions and achieves maximality independent of the round-off error by constructing uncovered areas with geometric primitives. While the output distributions are bias-free, and these methods are reasonably efficient in practice, they have non-linear (e.g. $O(n \log n)$) or unknown time complexity, and large storage requirements in practice.

Our improvements over prior art are based on the use of an implicit uniform grid. Uniform grids provide a fast way to determine if a candidate dart is disk-free. Since the active cells at any given stage are all the same size, it is easy to select one in constant time without bias; in contrast, both quadtrees and geometric voids have active areas of varying size, so need $\log |\mathcal{G}|$ time to select one without bias. Implicit uniform grids need only store spatial indices, and a global depth. We use a fixed-length array. No quadtree hierarchy is needed as in White et al. [2007] and Gamito and Maddock [2009]. No geometric arc-gons and polygons are needed as in Ebeida et al. [2011].

The traditional domain is the unit square, cube, ..., d-dimensional box. Some prior work addresses more complicated domains, such as 2-dimensional polygons with holes [Ebeida et al. 2011]. Because of our dependence on equal-weight uniform grids, we consider only the unit d-box.

2 Dart throwing guided by uniform grids

```
Maximal Poisson sampling:
  Generate background grid $\mathcal{G}_o$
  Mark all cells as valid: $\mathcal{C}_i = \mathcal{G}_o$
  **for** Stage $i = 0, \ldots$ bits_of_precision **do**
    remaining_darts = $A|\mathcal{C}_i|$
    **while** remaining_darts **do**
      Throw candidate dart $c$ into $\mathcal{C}_i$
      **if** $c$ is disk-free **then**
        mark its cell $\mathcal{G}_o^c$ as invalid
        remove $\mathcal{C}_i^c$ from $\mathcal{C}_i$
        remaining_darts−−
      **end if**
    **end while**
    Subdivide valid $\mathcal{C}_i$, retaining valid subcells $\mathcal{C}_{i+1}$
  **end for**
```

Background grid We start by covering the d-dimensional box with a uniform base grid, $\mathcal{G}_o$. Cells are sized so that a cell can accommodate at most one point: diagonals are $< r$ and side lengths $s_o < r/\sqrt{d}$. We choose s_o so that the side of the unit box is a whole integer multiple of the side of a cell: $s_o = 1/\lceil \sqrt{d}/r \rceil$. The grid can be completely described by the geometric position of one corner of the grid, the grid *spacing* s_o, and the number of cells in each axis direction. For breaking ties of where to place a dart, we consider cells to be open on their minimal extremes (half-open squares).

Disk-free check We use the base grid to check the disk-free property of any dart thrown throughout the algorithm: For a candidate dart c, we find its cell $\mathcal{G}_o^c$ in the base grid in $\mathcal{G}_o$, and retrieve all *nearby* cells in $\mathcal{G}_o$. For each of those cells, if it contains a dart we check if its distance to the new dart is at least r. If the candidate dart passes this test, we accept it as a success and store it using a pointer in its cell.

Here a cell is considered *nearby* if it contains a point (not a dart) at distance $< r$ from $\mathcal{G}_o^c$. The nearby cells come from a template of index offsets that we precompute before any of the stages. The template is a subset of the 5^d lattice of cells centered at $\mathcal{G}_o^c$. For each of the cells of the lattice, we compute the distance from its corners to the closest corner of $\mathcal{G}_o^c$. If the closest pair of corners is less than r, the cell is retained in the template. In practice this discards a significant number of cells, especially as the dimension increases. For efficiency, we order the retained cells by increasing distance to their farthest corner. Cells with farthest-corners that are closest to the center cell are checked first. In practice this has a dramatic effect on runtime.

Stages The dart throwing procedure takes place in stages. Each stage, $\mathcal{S}_i, i = 0, 1, 2 \ldots i_{\max}$, utilizes an implicit uniform grid with spacing $2^{-i} \times s_o$. This implicit grid, $\mathcal{G}_i$, is completely specified by the grid level i together with the specification of $\mathcal{G}_o$. Any cell can be addressed by d indices and the grid level i. During stage $S_{i>0}$ any cell of grid $\mathcal{G}_i$ can identify its implicit parent cell in the base grid $\mathcal{G}_o$ via simple integer operations.

The indices of the valid cells $\mathcal{C}_i$ are stored in d arrays, one for each dimension. Invalid cells for $i > 0$ are implicit and require no storage. These arrays are fixed size, and are used for all stages. The first $|\mathcal{C}_i|$ array entries hold $\mathcal{C}$. Removing sampled cell $\mathcal{C}_i^c$ involves replacing the arrays' contents with the contents at positions $|\mathcal{C}_i|$, then decrementing the counter that tracks $|\mathcal{C}_i|$. At the beginning of $\mathcal{S}_o$, all cells are valid. At the begining of $\mathcal{S}_i$, we subdivide the cells $\mathcal{C}_i$ and retain the valid ones as $\mathcal{C}_{i+1}$.

We wish to make the arrays big enough to accommodate future growth and in-place cell-list updates without dynamic memory allocation. We use arrays of length $d|\mathcal{C}_o|$ for $d \leq 4$. During a stage the cell lists only decrease, so it is sufficient to ensure that the number of valid cells must be less than $d|\mathcal{C}_o|$ at the start of a stage. When we transition to a new stage, we split each remaining valid cell into 2^d new cells and store any valid subcells in the existing array. In practice we expect there to be many fewer than 2^d valid subcells on average, because it is likely that the parent cell was sampled from but had a dart miss, so is partially covered. Nonetheless, there is a theory gap between the 2^d possible children and d extra space we have reserved for them. In practice, together with our performance factor A (see "Dart throws per stage", below), the d factor has been

Work	Bias-free	Maximal	Compute bound	Memory bound
Classic dart-throwing [Dippé and Wold 1985; Cook 1986]	yes	no	∞	n
Voronoi [Jones 2006]	no	yes	$n \log n$	$n \log n$
Scalloped sectors [Dunbar and Humphreys 2006]	no	no	$n \log n$	n
Hierarchical dart throwing [White et al. 2007]	yes	yes	$n \log n^{\dagger}$	$n \log n^{\dagger}$
Parallel multi-resolution uniform grid [Wei 2008]	no	no	n	n
Spatial subdivision [Gamito and Maddock 2009]	yes	yes	$n \log n$	$n \log n$
2-phase [Ebeida et al. 2011]	yes	yes	$n \log n$	n
This work	yes	yes	n	n

† White et al. claim $O(n)$ runtime and memory bounds, which are almost certainly in practice the dominant components of runtime and memory usage. However, their quadtree subdivision of the sampling domain, even given their assumption of an exponential dropoff across levels (last paragraph of their section 3.3), is not linear but instead still $O(n \log n)$ for both runtime and memory.

Table 1: *A comparison of Poisson sampling methods.*

adequate in up to four dimensions. For higher dimensions, preliminary results indicate that some combination of increasing A and the array size is needed.

Valid subcells Each top-level grid cell of $\mathcal{G}_o$ stores a Boolean variable indicating whether it is valid (it is available for future sampling) or not. For instance, placing a point (dart) in a cell makes that cell invalid. Since a cell might not contain a dart, but still be completely covered by the disks of several nearby darts, being valid does not guarantee that there exists some available region for sampling within that cell. It simply states that the algorithm cannot invalidate that cell based on the current checks.

When splitting cells for the next stage, must test each of the new subcells for validity. The validity test we use is similar to the square-inside-circle test proposed by White et al. [2007]. However, we improved its efficiency by splitting a cell and we apply the test to each one of its children. This improves the capability of detecting a cell that is invalid because it is covered by more than one point, as illustrated in Figure 2. Note that this test extends to any p-norm distance.

A cell is valid if one of its children is not covered by a single disk. That is, for some child and every disk the child cell's farthest corner is $r + \epsilon$ away from the disk center.

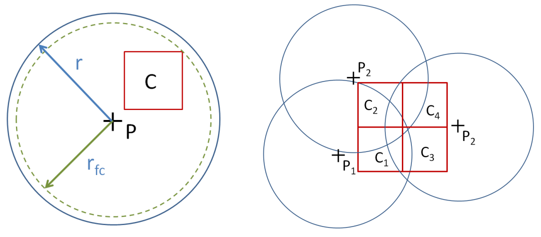


Figure 2: *In (a) cell C is invalid because its farthest corner from disk-center p is at distance $r_{fc} \leq r + \epsilon$. More cells are invalidated when this test is applied to all of its children, as demonstrated in (b).*

Update in place To update in place, stage i cells are popped from position $|\mathcal{C}_i|$, subdivided, and valid subcells placed moving backwards from position "end" $- |\mathcal{C}_{i+1}|$. Then the array is implicitly reversed.

Uniformly sample All the valid cells have the same size during a stage. We dart-throw by uniformly choosing a random valid cell. Then we select the dart's position by sampling uniformly from the cell's box. If the thrown dart is disk-free, i.e. it does not conflict with any previous successful dart, its cell is removed from the active pool, and its parent cell in $\mathcal{G}_o$ is also invalidated.

Dart throws per stage We throw $A|\mathcal{C}_i|$ successful darts at each stage, where A is an empirically-derived constant. Then the stage is complete and we proceed to the next level. The constant A is chosen based on two factors:

1. Speed: smaller A is faster.
2. Storage: larger A uses less memory.

A larger A slows down the algorithm non-linearly. When the remaining disk-free areas are very small and not well-characterized by the current cell resolution, the ratio of unsuccessful throws to successful ones increases. A small A results in a lot of valid cells at the end of a stage. These get refined, which leads to needing larger arrays to store the indices of $\mathcal{C}_i$.

According to our numerical studies, picking $A \in [0.3, 0.6]$ balances these two factors well. Our analysis shows that the performance is not sensitive to the exact value of A within that range.

The number of stages is bounded by the finest resolution that a finite precision world would allow. Our studies require double-precision floating point, resulting in 23 stages. Note that most of the time is consumed during the first few stages, when most valid cells are hit with darts; afterwards, with few remaining valid cells, the rest of the stages run quickly. The stages of sampling of a unit square using $r = 0.2$ are demonstrated in Figure 3.

Stage limit At the end of the last stage, b, the output is "probably" maximal. To get provably maximal, we must repeat the last stage, but with using the same-size cells rather than subdividing them. Each half-open $\mathcal{C}_b$ cell holds a single d-dimensional floating-point number. Thus, when a cell is sampled, there is only one place to put the dart, and we know it was disk-free at the start of the stage. This can be used to show that we expect at least a constant fraction (c_1 dependent on 2^{-d}) of the cells to get invalidated during each repeat of the last stage. The total amount of expected work to invalidate **all** the cells is a second constant c_2 times their cardinality. This second constant is dependent on the infinite geometric series sum of the powers of the first constant, $1/(1 - c_1)$. The bits of precision b does not appear in the runtime analysis because it is dominated by the infinite geometric series bound.

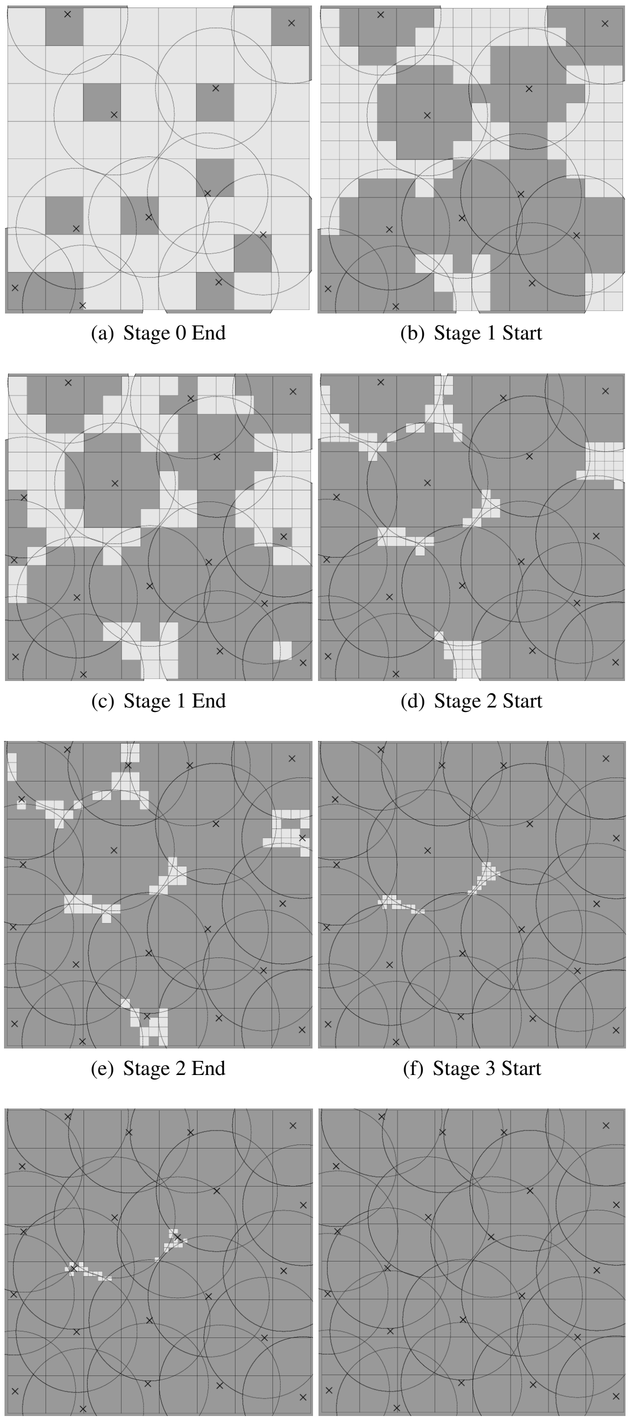

(a) Stage 0 End (b) Stage 1 Start (c) Stage 1 End (d) Stage 2 Start (e) Stage 2 End (f) Stage 3 Start (g) Stage 3 End (h) Stage 4 Start

Figure 3: *Example sample of a unit square with $r = 0.2$. A maximal distribution was achieved in four stages. Valid cells are light, invalid are dark. The end of each stage (left) has many invalid cells that are not detected until the beginning of the next stage (right). A maximal distribution was detected at the beginning of Stage 4 by the valid cells pool being empty.*

In practice the last stage and any repeats do not happen for even the largest samples we have created. The number of stages are usually bounded by the log of the sample size. (The runtime over all the stages is still linear due to the geometric sum bound, analogous to a binary tree with n leaves having $\log n$ depth but only $2n$ total size.) For many cases, such as our test samples of millions of points, the last stage to have any cells is around 16. This is smaller than the bits of precision, 23. We speculate that n would have to be in the billions before stage b would be likely to be needed.

Complexity Given the dimension, d, and bits of precision, b, our memory is $O(d^2bn)$. The amount of memory for a b-bit coordinate is b, and there are d arrays of coordinates of length d. The top level grid is of size n, which hides some dimensional and sampling radius dependence. The following paragraph is a step towards making this more precise.

For fixed dimension, $d^{d/2}r^{-d} = |\mathcal{G}_o| = \Theta(n)$ [Ebeida et al. 2011]. In one direction $|\mathcal{G}_o| \leq n$ since a cell can hold at most one disk-free dart. In the other $|\mathcal{G}_o| \geq n/5^d$, since the center point of each cell must be covered by at least one disk, and there are only 5^d cells close enough to hold such a dart. This gives $d^{d/2}(5r)^{-d} \leq n \leq d^{d/2}r^{-d}$; tight bounds and predictions are not generally accessible. A maximal Poisson disk sampling with radius r is equivalent to a maximal non-overlapping sphere packing with radius $r/2$. The average, worst, and best densities of sphere packings for arbitrary dimensions is a well studied problem in geometry. There are special case arrangements and dimensions (e.g. 24). An adequate discussion of packings is out of scope.

Our time is more difficult to analyze, but appears to be $\mathbb{E}(d5^{2d-1}n)$. We assume that a flop on b-bits takes constant time. Checking the distance between two points takes d time, because there are d coordinates to compare. When checking a dart for the disk-free property, up to 5^d nearby cells (and their darts) may need to be compared against. It remains to bound the ratio of thrown to accepted darts. Empirically, based on limited tests for $A = 0.6$, the ratio is roughly 5^{d-1}. Some intuition follows as to why this depends on dimension. For any valid cell, where it is cut by a disk (if at all) is random, so we expect the average disk-free volume to be about half the cell volume at the start of any stage. The key is that at the beginning of the stage a cell selected to contain a dart has a constant chance of that dart being a disk-free hit. This drops as darts are thrown. When a new disk is accepted, about 5^d top-level grid cells intersect a new disk, reducing their likelihood of containing a subsequent successful dart. That is, as the dimension increases, the likelihood of successfully throwing subsequent darts is reduced.

3 Results

3.1 Memory and Time

Figure 4 shows the effect of the parameter A on the runtime of our algorithm in two dimensions. The effect is rather mild. However, as the dimension increases, the progress made in each stage changes (Figure 7). The low running times and large number of cells for early stages for higher dimensions implies that a larger value of A, dependent on dimension, might give an advantage.

Figures 5 and 6 compares the running time and memory usage of our algorithm to Gamito and Maddock's [2009] in 2, 3 and 4 dimensions, and to Ebeida et al.'s [2011] in 2 dimensions. We are grateful to the authors for freely providing their software. Gamito and Maddock's was the only bias-free, higher-than-two-dimensional Poisson-disk software we were able to obtain for comparison. Gamito and Maddock's results were competitive with the

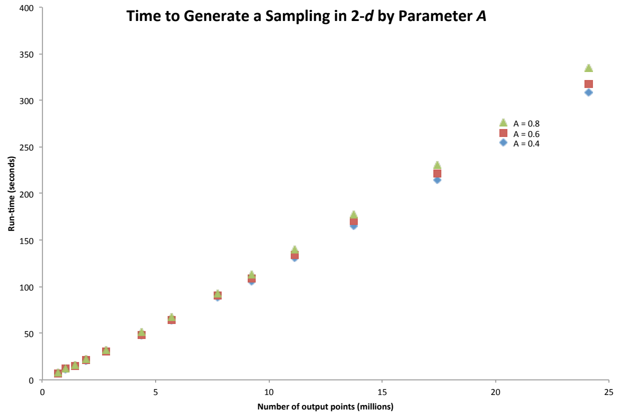
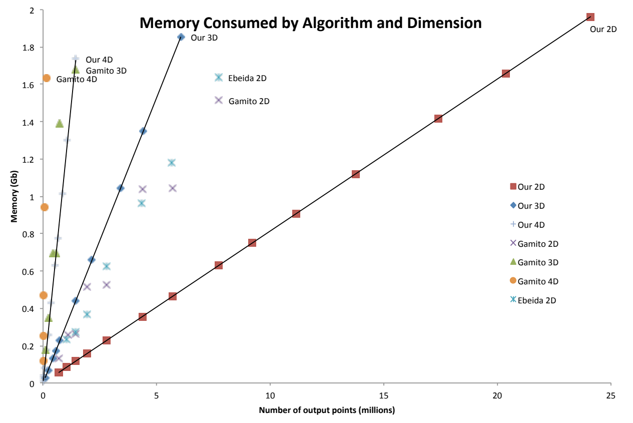
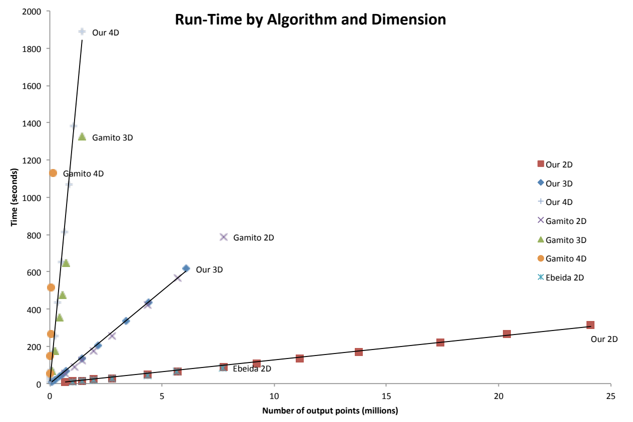


Figure 4: *The parameter A affects the runtime of our 2-d algorithm.*



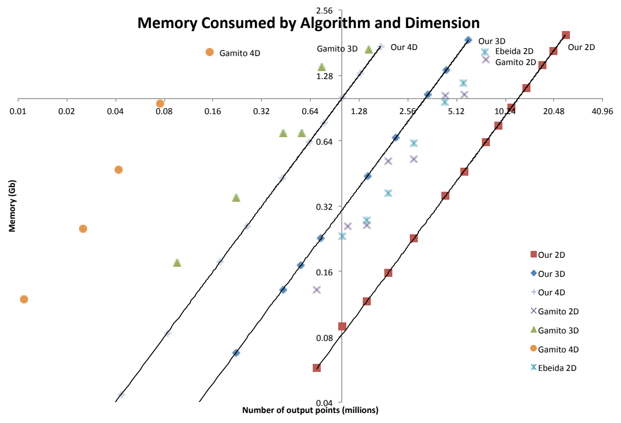
(a) memory



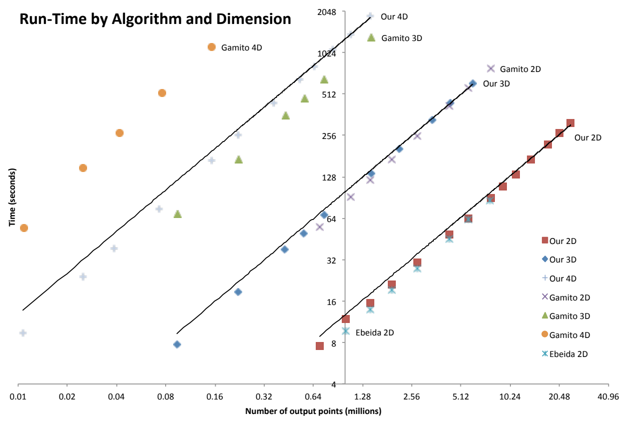
(b) time

Figure 5: *Memory and time used by our algorithm vs. Gamito and Maddock's and Ebeida et al.'s. Linear scale with 0-intercept trendlines.*

alternatives at the time of their publication in 2009, so despite the lack of other available software we believe this is a good basis for comparison.



(a) memory



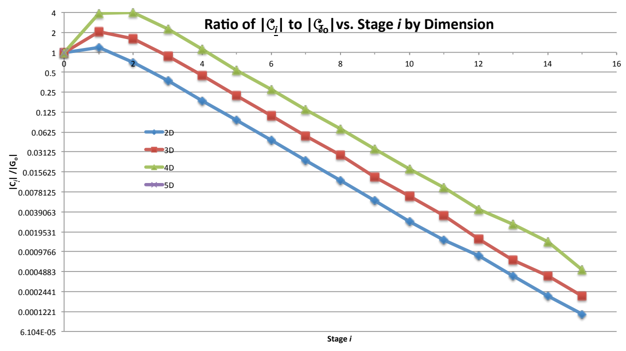
(b) time

Figure 6: *Memory and time used by our algorithm vs. Gamito and Maddock's and Ebeida et al.'s. Log-log scale with 0-intercept trendlines.*

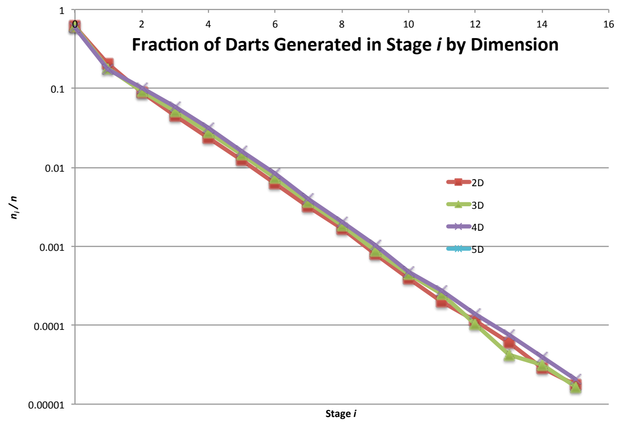
It appears that we use a factor of 8 less time and a factor of 2–3 less memory than Gamito and Maddock in 2 dimensions. In 3 dimensions the factors are 9–10 for time and 4–5 for memory. In 4d, the factors are 6 (for $A = 0.6$) to 8 (for $A = 1.6$) for time and 10 for memory. We are able to go one dimension higher given the same time and space bounds. In 2 dimensions, we use about the same runtime as Ebeida et al., but a factor of 2–3 less memory. In all cases the largest sampling we could create was limited by the available memory, rather than runtime; the most expensive calculations took about half of an hour on a laptop and used all of the two GB of memory.

Gamito and Maddock's Figure 4 [2009] shows both the runtime and memory challenges in efficiently generating a fixed-size sample as the dimension increases; our results in Figure 5 across dimension are consistent with these trends. While the complexity is roughly linear in output size for all of the methods we considered, including ours, the dependence on dimension was severe. As an example of the fundamental difficulty, the ratio of the number of darts we threw to the number that were accepted decreases severely with dimension, even for fixed $A = 0.6$ and number of points. For one million points in 2, 3, and 4 dimensions, the ratios were $\{6, 27, 141\}$.

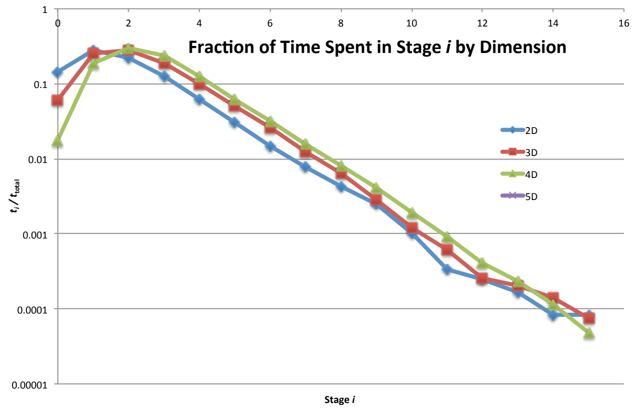
In 5 dimensions, by increasing A to 15.0 but keeping the cell array size at $d|\mathcal{G}_o|$, we generated 341,176 points in three hours. We used



(a) $|\mathcal{C}_i|/|\mathcal{G}_o|$



(b) Fraction of darts by stage.



(c) Fraction of time by stage.

Figure 7: *Time and memory features of our algorithm by stage and dimension, for generating one million points in the unit d-box.*

1.5 GB of memory. We had to throw 2.7 billion (2.7e9) darts. This is a factor of 10 higher what we predicted for $A = 0.6$ in Section 2, but for $A = 0.6$ too many subcells were generated to fit in the cell array.

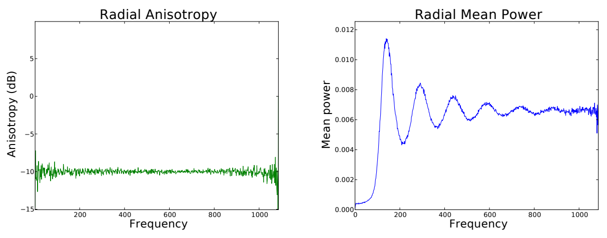


Figure 8: *Radial mean power and anisotropy estimates for our 2D samples. The plots were generated by averaging the spectral behavior of ten samples. The radial mean power follows the expected behavior of a blue noise spectrum, and the anisotropy stays consistently at -10 dB.*

3.2 Bias-free

Our method for maximal Poisson-disk sampling is provably free from bias. To demonstrate this in practice, we present radial mean power and anisotropy variations for an averaged collection of ten sampling patterns. These plots, shown in Figure 8, follow expected blue noise behavior and match past literature [White et al. 2007; Wei 2008]. The anisotropy measure indicates a consistent drop of 10 dB across the spectrum for ten distributions. We are confident that this behavior extends to higher dimensions as well.

4 Conclusions

We report a practical method for generating a maximal Poisson-disk sampling in finite precision. The expected time is linear, and the deterministic memory is linear, in the output size, n, for a fixed dimension. The method extends to arbitrary dimensions, and we provide implementation results in up to five dimensions. In practice our methods do better than the available alternatives, especially for higher dimensions. We are able to generate larger meshes, and we generate them more quickly, often by an order of magnitude. We plan to make our software available to the wider community for experimentation and comparison. GPU implementations are possible.

Our method has several tuning parameters affecting the runtime and memory usage; we plan to investigate these further, especially their optimal values for a given dimension. We encourage the broader community to analyze the family of maximal Poisson-disk methods more carefully in terms of their dependence on dimension, and finite-precision effects.

A challenge is to provide efficient software for sampling in moderate (e.g. 10) dimensions. High dimensions (e.g. 1000) appear to be out of reach for any known approach. Most methods (time and memory) are nearly linear in output size in practice, but finessing the apparent doubly-exponential dependence on dimension appears to be a fundamental challenge.

References

ATTALI, D., AND BOISSONNAT, J.-D. 2004. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete & Computational Geometry 31*, 3 (Feb.), 369–384.

BOLANDER, J. E., AND SAITO, S. 1998. Fracture analyses using spring networks with random geometry. *Engineering Fracture Mechanics 61*, 5-6, 569 – 591.

BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. 2010. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Transactions on Graphics 29* (Dec.), 166:1–166:10.

BRIDSON, R. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, 22.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Transactions on Graphics 22*, 3 (July), 287–294.

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

DICKMAN, R., WANG, J.-S., AND JENSEN, I. 1991. Random sequential adsorption: Series and virial expansions. *Journal of Chemical Physics 94*, 8252–8257.

DIPPÉ, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, 69–78.

DUNBAR, D., AND HUMPHREYS, G. 2006. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics 25*, 3 (July), 503–508.

EBEIDA, M. S., PATNEY, A., MITCHELL, S. A., DAVIDSON, A., KNUPP, P. M., AND OWENS, J. D. 2011. Efficient maximal Poisson-disk sampling. *ACM Transactions on Graphics 30*, 4 (Aug.).

GAMITO, M. N., AND MADDOCK, S. C. 2009. Accurate multidimensional Poisson-disk sampling. *ACM Transactions on Graphics 29*, 1 (Dec.), 8:1–8:19.

JIRÁSEK, M., AND BAZANT, Z. P. 1995. Particle model for quasibrittle fracture and its application to sea ice. *Journal of Engineering Mechanics 121*, 1016–1025.

JONES, T. R. 2006. Efficient generation of Poisson-disk sampling patterns. *journal of graphics tools 11*, 2, 27–36.

LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Transactions on Graphics 24*, 4 (Oct.), 1442–1461.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, 65–72.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics 23*, 3 (Aug.), 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Transactions on Graphics 26*, 3 (July), 78:1–78:6.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

WEI, L.-Y. 2008. Parallel Poisson disk sampling. *ACM Transactions on Graphics 27*, 3 (Aug.), 20:1–20:9.

WHITE, K. B., CLINE, D., AND EGBERT, P. K. 2007. Poisson disk point sets by hierarchical dart throwing. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, 129–132.