Accurate Multidimensional Poisson-Disk Sampling

MANUEL N. GAMITO
Lightwork Design Ltd.
and
STEVE C. MADDOCK
The University of Sheffield

We present an accurate and efficient method to generate samples based on a Poisson-disk distribution. This type of distribution, because of its blue noise spectral properties, is useful for image sampling. It is also useful for multidimensional Monte Carlo integration and as part of a procedural object placement function. Our method extends trivially from 2D to 3D or to any higher dimensional space. We demonstrate results for up to four dimensions, which are likely to be the most useful for computer graphics applications. The method is accurate because it generates distributions with the same statistical properties of those generated with the brute-force dart-throwing algorithm, the archetype against which all other Poisson-disk sampling methods are compared. The method is efficient because it employs a spatial subdivision data structure that signals the regions of space where the insertion of new samples is allowed. The method has $O(N \log N)$ time and space complexity relative to the total number of samples. The method generates maximal distributions in which no further samples can be inserted at the completion of the algorithm. The method is only limited in the number of samples it can generate and the number of dimensions over which it can work by the available physical memory.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Antialiasing*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*; I.4.1 [**Image Processing and Computer Vision**]: Digitization and Image Capture—*Sampling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Accurate Poisson-disk sampling, dart throwing, multidimensional sampling, maximal sampling, spatial subdivision

ACM Reference Format:

Gamito, M. N. and Maddock, S. C. 2009. Accurate multidimensional poisson-disk sampling. ACM Trans. Graph. 29, 1, Article 8 (December 2009), 19 pages.
DOI = 10.1145/1640443.1640451 http://doi.acm.org/10.1145/1640443.1640451

1. INTRODUCTION

Poisson-disk sampling is a process that distributes uniform random samples on a domain of n-dimensional space based on a minimum distance criterium between samples. We propose an efficient Poisson-disk sampling method that generates samples on the domain $D = [0, 1]^n$, consisting of a unit hypercube in n-dimensional space. The outcome of the method is a set $X = \{\mathbf{x}_i \in D;\ i = 1, 2, \cdots, N\}$ of N samples for which the sampling conditions can be expressed as

$$\forall \mathbf{x}_i \in X,\ \forall S \subseteq D :\ P(\mathbf{x}_i \in S) = \int_S \mathbf{dx}, \tag{1a}$$

$$\forall \mathbf{x}_i, \mathbf{x}_j \in X :\ \|\mathbf{x}_i - \mathbf{x}_j\| \geqslant 2r, \tag{1b}$$

where the parameter r is called the *distribution radius*.

Condition (1a) states that a uniformly distributed random sample $\mathbf{x}_i$ of X has a probability of falling inside a subset S of D that is equal to the hypervolume of S. For example, if $S \subset D$ is one half the size of D then the probability of a new sample being placed inside S is exactly one half. Condition (1a) already takes into account the fact that $\int_D \mathbf{dx} = 1$ for the unit hypercube, irrespective of the dimension n of the space. Condition (1b) enforces the minimum distance constraint between any pair of samples.

A Poisson sampling process is one that enforces condition (1a) alone. The reason for the name is because the number of samples that falls inside any subset $S \subseteq D$ obeys a discrete Poisson distribution [Snyder 1991]. Poisson sampling, although simple to implement, is not favored in computer graphics because it leads to sample distributions where the samples are noticeably grouped into clusters of different sizes and densities. This "clumpiness" of Poisson distributions is a consequence of the samples being generated independently so that any two samples can be arbitrarily close to each other. It is better to have a sampling process that distributes random samples in an even manner across D so that no clustering is perceived. Condition (1b) helps to combat clustering by preventing samples from being closer than some chosen value $2r$.

M. N. Gamito was supported by grant SFRH/BD/16249/2004 from the Fundação para a Ciência e a Tecnologia, Portugal.
Authors' addresses: M. N. Gamito, Lightwork Design Ltd., Rutledge House, 78 Clarkehouse Road, Sheffield S10 2LJ, UK; email: manuel.gamito@lightworkdesign.com; S. C. Maddock, Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, UK; email: s.maddock@dcs.shef.ac.uk.

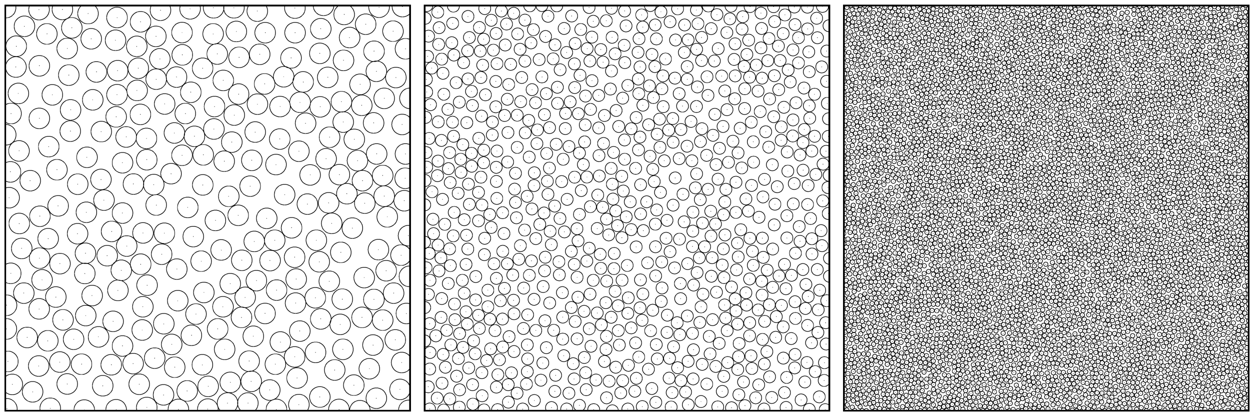


Fig. 1. Two-dimensional Poisson-disk sampling on the unit square. From left to right, the distribution radii are 0.025, 0.015, and 0.0075. The number of samples is 294, 798, and 3 148, respectively. The time taken to generate the samples with our algorithm was 0.003s, 0.009s, and 0.041s, respectively.

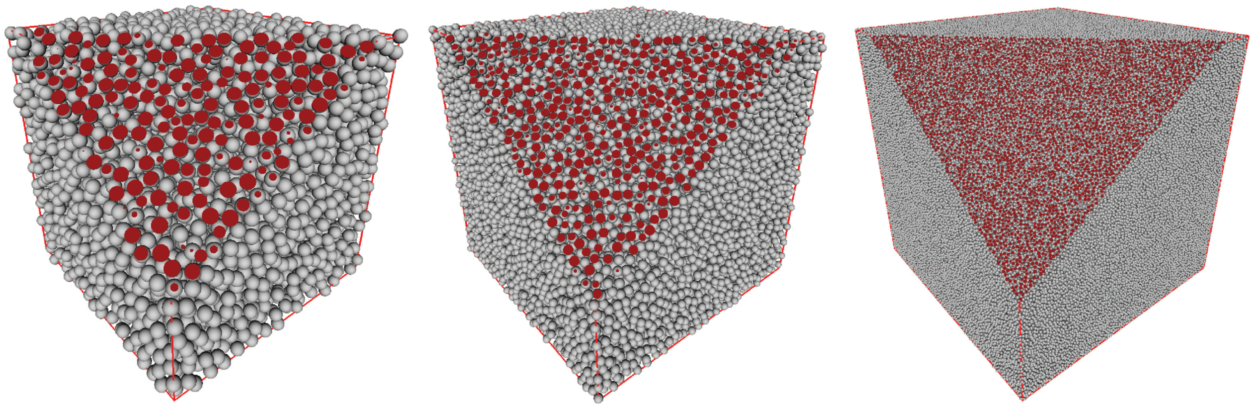


Fig. 2. Three-dimensional Poisson-disk sampling on the unit cube. From left to right, the distribution radii are 0.025, 0.015, and 0.0075. The number of samples is 6 457, 28 807, and 223 518, respectively. The time taken to generate the samples with our algorithm was 1.870s, 9.966s, and 86.391s, respectively.

The term Poisson-disk stems from the observation that samples in two dimensions are located at the center of disks of radius r so that no overlapping of disks occurs within the distribution. This is exemplified in Figure 1, which shows three dense Poisson-disk distributions for decreasing values of the r parameter. In three dimensions, the same sampling process can be referred to as Poisson-sphere because samples now occupy the center of nonoverlapping spheres of radius r, as Figure 2 exemplifies. For historical reasons, we keep the name Poisson-disk irrespective of the dimensionality of the space.

There is some confusion in the computer graphics literature regarding the parameter r of a Poisson-disk distribution. Some authors use r to label the minimum allowable distance between samples. Other authors use it to label the distribution radius, that is, the radius of the Poisson disks or spheres. We employ the latter definition with the understanding that it is equal to half the value of the former.

Poisson-disk sampling processes were first studied by the Swedish statistician Bertil Matérn to describe the distribution of trees in a forest [Matérn 1960; Ripley 1977]. He devised two processes through which a distribution of Poisson-disk samples could be generated. In particular, the Poisson-disk process that we are interested in studying, known in computer graphics as *dart-throwing*, is equivalent to the *Matérn second process*. In this process, samples have a uniform random position and a uniform random birth time. As time progresses, samples are accepted or rejected based on their distance to the samples that have already been born. In the fields of chemistry, statistical physics, and the physics of granular materials, the Poisson-disk process is also known as the hard-core process and is one instance of random sequential adsorption and of random close packing [Baddeley and Møller 1989; Dickman et al. 1991; Jaeger and Nagel 1992].

Although conditions (1a) and (1b) are enough to specify a valid Poisson-disk distribution of samples, we are interested in an additional condition that characterizes a distribution as being *maximal*. A maximal Poisson-disk distribution is one where it is not possible to insert any further samples without violating the minimum distance constraint. Specifically, a Poisson-disk distribution is maximal when it verifies the condition

$$\forall \mathbf{x} \in D,\ \exists\, \mathbf{x}_i \in X :\ \|\mathbf{x} - \mathbf{x}_i\| < 2r. \tag{2}$$

Condition (2) implies that there are no more available points in the domain over which to place new samples. This is because every domain point $\mathbf{x}$ is already at a distance smaller than $2r$ from at least one sample in the distribution. Placing a new sample at $\mathbf{x}$ is not possible as it would violate condition (1b). For this reason, a maximal distribution can also be said to be *jammed*[1]. Maximal Poisson-disk distributions are desirable because they maximize the number of samples generated while, at the same time, eliminating any space that remains unoccupied. Valid Poisson-disk distributions that are not maximal may potentially have large gaps inside of them devoid of samples. This makes sparse Poisson-disk distributions comparable to the simpler Poisson distributions in that there may be a noticeable clustering of samples.

We present an efficient method to generate large numbers of samples in a n-dimensional space as the outcome of a maximal Poisson-disk sampling process. Our method uses a subdivision tree in n dimensions to help in the placement of samples and generates accurate Poisson-disk distributions. We begin in Section 2 by discussing the applications of Poisson-disk sampling in computer graphics. Section 3 then presents current methods for Poisson-disk sample generation. Our method is explained in Section 4. A comparison of our method with previous methods is done in Section 5, where results are also presented and discussed. Section 6 concludes the article and proposes future developments. Appendix A gives a proof of the correctness of our proposed algorithm for the generation of Poisson-disk samples with the desired statistics. Appendix B details the routine for testing the intersection between a sample and a node in the tree, represented by a hypercube.

2. POISSON-DISK SAMPLING FOR COMPUTER GRAPHICS

Poisson-disk sampling was introduced to computer graphics by Dippé and Wold [1985], who proposed it as a sampling technique for image antialiasing. The reason why Poisson-disk sampling is ideal for antialiasing is because of its blue noise spectral properties. A blue noise spectrum is characterized by the absence of significant power content in the low frequencies with the exception of a strong DC spike at the origin. Based on results by Leneman [1966], it is possible to derive an analytic expression for the mean power spectrum of a one-dimensional Poisson-disk process. Figure 3 shows this power spectrum. The frequency response of a typical lowpass antialiasing filter is also shown in Figure 3. The gap in the Poisson-disk spectrum for low frequencies (with a width that varies with the inverse of the radius r) allows the antialiasing filter to recover the original signal whose spectrum is centered around the DC spike. Copies of the spectrum from the original signal are also spread along the high frequencies by the Poisson-disk process in an uncorrelated manner. The high-frequency content is attenuated by the frequency response curve of the filter and shows up as a low-power high-frequency noise. This type of noise is much less objectionable to the human visual system than the coherent aliasing artifacts that would have resulted if a deterministic sampling process had been used instead. Research done by Yellot [1983] on the distribution of photoreceptor cells in the retina of rhesus monkeys shows that it closely resembles a Poisson-disk distribution for the areas outside the fovea. Such a result helps to explain why visual perception is so forgiving of low-power noise artifacts and, therefore, why Poisson-disk sampling is useful for image sampling and antialiasing. The simpler Poisson sampling, in contrast, presents a flat spectrum across all frequencies and is not recommended for image sampling since it cannot separate

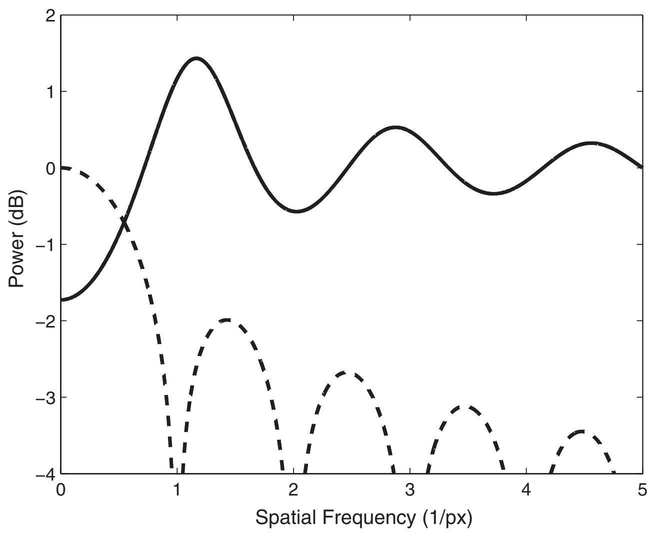


Fig. 3. The average power spectrum of a one-dimensional Poisson-disk process corresponds to a blue noise spectrum. The dotted line shows the frequency response of a typical low-pass filter.

the low frequencies of the signal from the high frequencies of the aliases.

Cook [1986] suggested that Poisson-disk sampling could be used as part of a distributed raytracing algorithm, not only to perform antialiasing but also to generate such effects as motion blur, depth of field, smooth shadows, glossy reflection, and transparency. Unfortunately, methods for generating Poisson-disk distributions were inefficient at the time and Cook relied instead on *jittered sampling* as a less expensive alternative. Jittered sampling places samples at the nodes of a uniform grid and perturbs their position by adding a small random displacement. The spectral properties of a jittered sampling distribution are not as good as those for Poisson-disk sampling, with a low-frequency gap that is smaller and less well resolved. Recently, Hachisuka et al. [2008] extended the results of Cook [1986] by sampling directly in the multidimensional space of the *rendering equation*, instead of sampling in image space. The authors were able to show that adaptively sampling in the parameter space of the rendering equation leads to a reduction in noise and also a reduction in the number of required samples.

The applications of Poisson-disk sampling to raytracing are instances of multidimensional Monte Carlo integration. Monte Carlo integration is also ubiquitous in global illumination where it is often used to accumulate the contribution of incoming radiance over the hemisphere above a surface point [Dutré et al. 2006]. The Monte Carlo integration technique computes a numerical approximation of an integral by accumulating the contributions of random samples taken from the integrand function [Glassner 1995]. For greater accuracy, the samples are distributed according to a probability density function that is proportional to the integrand function, in a process known as *importance sampling*, so that areas where the integrand takes on larger values are sampled at a higher density. Importance sampling can be achieved by first generating a Poisson-disk distribution in a canonical hypercube $[0, 1]^n$, for an n-dimensional integral, and using the cumulative probability density function to warp this hypercube into the domain of integration [Shirley 1992]. Ostromoukhov et al. [2004] and Kopf et al. [2006], however, have shown that direct generation of a nonuniform distribution leads to visually better results when compared with the domain warping

[1]We classify a distribution as jammed based on the ability to insert more samples. In other fields, a different definition is used: a distribution of N points is jammed when the disks or spheres around those points become interlocked and cannot move.

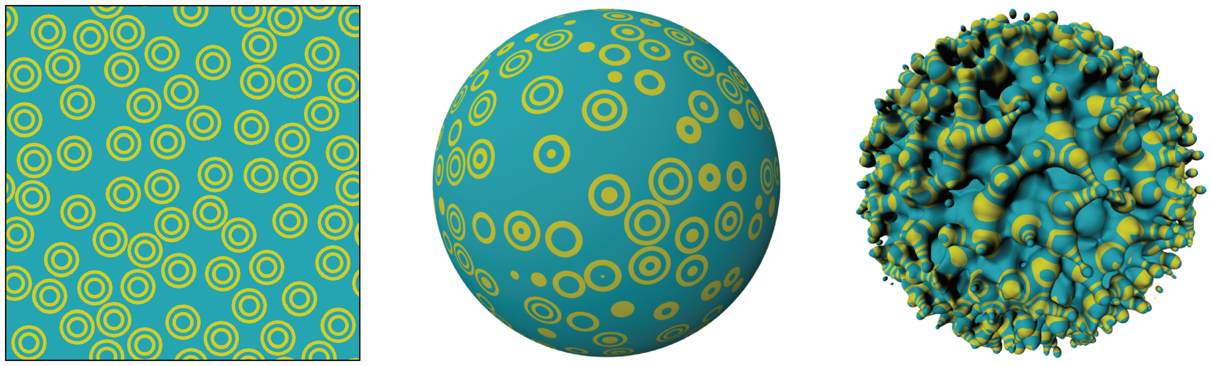


Fig. 4. A procedural object distribution function used to generate a two-dimensional texture (left), a solid texture (middle), and both a solid texture and a hypertexture (right).

method. A nonuniform Poisson-disk sampling can be formally defined to obey conditions (1a) and (1b) with a variable radius so that a sample placed at $\mathbf{x} \in D$ should have no samples closer than $2r(\mathbf{x})$ [Bartlett 1974]. Ostromoukhov et al. [2004] demonstrated the application of nonuniform distributions to perform importance sampling of environment maps while Kopf et al. [2006] used them to perform image dithering in real time.

Graphical objects can be distributed in space based on a Poisson-disk sampling. This has been used to distribute ink strokes for nonphotorealistic rendering [Deussen et al. 2000; Secord et al. 2002] and it has also been used to create plant ecosystems by instancing multiple copies of a single plant object [Deussen et al. 1998; Cohen et al. 2003]. It is often important that copies of the original object do not intersect. If the object can be bounded by a disk or a sphere of radius r then a Poisson-disk distribution with the same radius will not create interferences between the copies. Lagae and Dutré [2005, 2006a] have proposed a procedural object placement function by creating an infinite nonperiodic distribution of Poisson-disk samples on the plane. The same authors later extended their procedural object placement function to three dimensions [Lagae and Dutré 2006b]. Figure 4 shows examples of procedural placement of a radially symmetric primitive. More complex examples can include pseudorandom rotations and scalings that are applied when instancing the primitive. Sample placement with Poisson-disk properties over two-dimensional manifolds is another possibility [Fu and Zhou 2008; Lehtinen et al. 2008; Li et al. 2008; Cline et al. 2009]. A generalization of Poisson-disk sampling that uses elliptical samples instead of disks has been introduced to computer graphics by Feng et al. [2008]. Poisson-disk sampling can also be used instead of Poisson sampling when generating procedural noise functions such as sparse convolution noise or cellular texture noise, leading to a better distribution of the features in these noise functions [Lewis 1989; Worley 1996].

3. METHODS FOR POISSON-DISK SAMPLING

We present here an introduction to the methods that have been developed to generate Poisson-disk distributions. The reader is referred to Lagae and Dutré [2008] for a more in-depth survey of these methods. We make a distinction between *accurate methods*, *approximate methods*, and *tile-based methods* for Poisson-disk sampling. Accurate methods obey conditions (1a) and (1b), generating correct Poisson-disk distributions. Condition (1b), however, is generally difficult to enforce and this has led to the development of approximate methods. These methods generate distributions that, although not having true Poisson-disk properties, attempt to reach a blue noise power spectrum with variable degrees of success. Tile-based methods can generate distributions in real time by drawing from a finite set of tiles containing carefully selected samples. In most cases, tile-based methods use either accurate or approximate sampling methods as part of a precomputation stage that populates their initial tile set with samples. Special rules are used that prevent the tilings from becoming periodic. Nevertheless, these methods generate a weak form of quasiperiodic tilings because each tile is copied repeatedly across space. If the sampling space is displaced, several copies of the tiles overlap. This quasiperiodicity becomes apparent when periodograms of the distributions are computed. The periodograms show many spikes due to the copies of a same tile overlapping exactly in frequency space.

Tile-based Poisson-disk sampling methods are the fastest because all samples have already been generated at runtime. The methods simply select, for any given point in space, which tile should samples be drawn from. Approximate methods have intermediate complexity. They gain speed, relative to accurate methods, by relaxing one or both of the Poisson-disk sampling conditions. Accurate methods are the slowest since they are required to enforce exactly both conditions for Poisson-disk sampling. Despite the existence of many fast approximate and tile-based sampling algorithms, the need for algorithms that can generate Poisson-disk distributions accurately is still justified. The need to enforce the uniform probability distribution constraint (condition (1a)) is important for image filtering operations and, more generally, for Monte Carlo integration. The failure to verify this constraint has the potential of introducing bias into the Monte Carlo integral. The need to enforce the minimum distance constraint (condition (1b)) is important for procedural object placement methods. Failure to verify this constraint could cause some of the object instances to intersect.

3.1 Accurate Methods

Dart-throwing was the first method developed in computer graphics for Poisson-disk sampling [Dippé and Wold 1985]. Random samples are continually tested and only those that satisfy the minimum distance constraint relative to samples already in the distribution are accepted. The main source of inefficiency of the method is a rejection sampling mechanism: a large number of samples is attempted but only a small percentage of them are inserted into the distribution. The algorithm cannot guarantee that a maximal distribution will be generated; as the allowable area for new insertions gradually shrinks, the probability that attempted samples will fall inside this area becomes progressively smaller. This also means that the algorithm does not have a guaranteed termination. If too many samples are requested, the algorithm can effectively become locked as it tries to generate samples that fall into arbitrarily small areas of space.

For many years, dart-throwing was the only available method for accurate Poisson-disk sampling. Its inefficiency led to the development of approximate sampling algorithms. The situation changed recently with the development of efficient dart-throwing methods. These new methods take advantage of a spatial data structure to guide in the placement of samples. The data structure encodes the regions of space where the insertion of samples is allowed. This avoids to a great extent the expensive procedure of having to blindly test new samples by trial and error. Every time a sample is inserted in the distribution, the spatial data structure is updated to remove the portion of space occupied by the new sample. The spatially guided methods, used in computer graphics, were developed specifically for two-dimensional sample distributions and do not extend well to higher dimensions.

The first spatially guided method was proposed by Jones [2006]. The method uses a Voronoi tessellation as the spatial data structure with the samples at the centroids of the Voronoi cells. The Voronoi cell of a sample is randomly selected and a new sample is inserted in the available area of the cell that falls outside a circle of radius $2r$ with the original sample at the center. A weighted binary tree helps in the selection of samples, with the Voronoi cells as the leaves and with the available areas of the Voronoi cells as the weights. This ensures that sample placement is done with a uniform probability distribution; the tree is randomly traversed top to bottom, with the area weights giving the probability of selecting the left or right child of each tree node. The placement of a new sample requires the computation of the intersection between the Voronoi cell (a polygon) and the circle of radius $2r$, which can be reduced to four fundamental cases. A rejection sampling method cannot be avoided but the probability of a new sample being accepted is much larger than the probability of it being rejected. Although Voronoi tesselations can be extended to three dimensions, placing a new sample in the available area of a three-dimensional Voronoi cell requires the computation of the intersection between the cell (a polytope) and a sphere of radius $2r$. This is a more complex procedure than its two-dimensional equivalent and cannot be reduced to a small number of simple cases.

Another spatially guided method was proposed by Dunbar and Humphreys [2006]. It uses a spatial data structure that the authors have termed "scalloped sector," which is bounded by two arcs of circles of different radii and centered at distinct points. The available area around each sample can be represented as the disjoint union of several scalloped sectors. Similar to the method by Jones [2006], a weighted binary tree is used to select a scalloped sector for the placement of a new sample, resulting in a spatial uniform probability distribution. A rejection sampling strategy is avoided as sampling inside a scalloped sector is always guaranteed to generate a valid Poisson-disk sample. It is not known how the scalloped sector data structure can be extended to three dimensions.

Yet another spatially guided method in two dimensions was proposed by White et al. [2007]. Similar to our proposed method, a quadtree is used to signal the allowable sample insertion space. An auxiliary uniform grid stores neighboring information about samples and is used to check for minimum distance conflicts for every new sample. The cells in the grid have lateral size $2r$ and all the possibly conflicting samples of a newly inserted sample are found by looking in the grid cell where the new sample falls plus the eight surrounding cells. This method can easily be generalized to higher dimensions but it does not scale well due to the need for a uniform grid. The memory size of the grid is $O(r^{-n})$ for n dimensions and this can become intractable for small r. Our proposed method does not require an auxiliary grid and scales better with increasing n or decreasing r since a single subdivision tree data structure is used.

Accurate Poisson-disk distributions can also be generated with a molecular dynamics simulation method [Lubachevsky and Stillinger 1990]. A distribution of molecules in a gas is initialized with random positions and velocities and the algorithm tracks the elastic collisions that occur between molecules. At the same time, the radius of the molecules is gradually increased, which also increases the frequency of collisions. The algorithm must stop before the rate of collisions begins to diverge, which occurs when a small increase in molecule radius leads to a very large increase in the number of collisions. This method can generate distributions with a tight packing of samples and, given enough time, can even generate crystalline structures in two dimensions. The minimum distance constraint is always verified relative to the molecule radius at the termination of the algorithm. Because the algorithm models a gas in a state of equilibrium, all points in space have an equal probability density for molecule placement. Molecular dynamics can generate Poisson-disk distributions, provided the algorithm is stopped before the molecules progress significantly towards a deterministic hexagonal packing that corresponds to the tightest possible distribution (refer to the concept of *relative radius*, explained in Section 4.6). The generation of maximal distributions is not guaranteed because the number of molecules is kept constant and, as they move, empty spaces may appear inside the domain. The molecular dynamics method has been extended to three and higher dimensions and also to ellipsoidal shapes [Lubachevsky et al. 1991; Donev et al. 2005; Skoge et al. 2006]. The formation of crystalline structures has only been verified in the two-dimensional case, however.

3.2 Approximate Methods

In the category of approximate sampling methods, there is the already mentioned jittered sampling [Cook 1986]. A similar jittering technique that perturbs samples away from the nodes of a hexagonal grid is also possible [Glassner 1995]. Error diffusion algorithms that were originally developed for image dithering can be applied to generate approximate Poisson-disk distributions [Mitchell 1987; Ulichney 1988].

Two popular algorithms were devised to overcome the deficiencies of the original dart-throwing algorithm. Glassner [1995] calls them the "best candidate algorithm," by Mitchell [1991], and the "decreasing radius algorithm," by McCool and Fiume [1992]. These algorithms have the interesting property of generating hierarchical streams of samples. If the sample sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}$ is approximately Poisson-disk with radius r_i, the larger sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_{i+1}\}$ is also approximately Poisson-disk with radius $r_{i+1} < r_i$, up to a maximum sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. These hierarchical streams are attractive because they allow several Poisson-disk distributions with different radii to be generated from a single run of the algorithm.

The best candidate algorithm works by trying mi samples when placing the ith new sample, where m is a supplied parameter. From all mi samples attempted, the one that is farther away from all previous $i - 1$ samples is chosen. The algorithm does its best to place samples well away from each other but it does not enforce any particular distribution radius r. There is the probability, however small, that a sequence of unfavorable sampling outcomes will make the best candidate sample be arbitrarily close to some other previous sample.

The decreasing radius algorithm, as the name implies, slowly decreases the radius r_i of the distribution at each iteration i until the final desired radius r is reached or a desired number of samples is generated. For each intermediate radius, it makes a finite number of attempts to place new samples, proportional to i, before proceeding to the next smaller radius. What makes the decreasing radius algorithm an approximate Poisson-disk sampling method is that it uses radii that are larger than r for most of the iterations. This violates condition (1a) because, for iteration i, the probability of placing a new sample at a distance of between $2r$ and $2r_i$ relative to a previous sample is zero. In fact, this probability should be proportional to the area of the annulus around the previous sample with inner and outer radii of $2r$ and $2r_i$, respectively, not considering the presence of other nearby samples that may reduce this allowable area.

Dunbar and Humphreys [2006] give a fast $O(N)$ method that results from collapsing their scalloped sector data structure into a single arc of a circle with radius $2r$. With this transformation, every new sample is always placed at a distance of exactly $2r$ from some other previous sample. This signifies that condition (1a) cannot be enforced since samples are not free to be placed anywhere in space with equal probability.

Bridson [2007] proposed a multidimensional sampling method that subdivides the domain into a uniform grid for easier neighbor sample checking, similar to White et al. [2007]. An active list of samples is kept. At each iteration, a sample from the active list is randomly chosen and several dart-throwing attempts try to insert a new sample inside a hypersphere of radius $4r$ centered on the chosen sample. The new sample is added to the grid and to the active list while the previously chosen sample is removed from the list if dart-throwing did not succeed after some number k of attempts. The method does not distribute samples uniformly because every new sample is always placed inside a hyperspherical neighborhood of some previous sample.

Wei [2008] proposed a parallel sampling method that can run on a GPU. The method uses a multiresolution strategy where uniform subdivisions of the domain with increasing resolution are considered one at a time. The cells in each resolution level are then arranged into distinct cell groups in such a way that the insertion of new samples inside each group cell can proceed independently from the insertion of samples in the other cells of the same group. This allows sample insertion to be parallelized for each of the groups of any resolution level. Sample insertion is done by making k dart-throwing attempts inside every group cell. Although the sampling inside each group is random, the sequence of groups visited for every resolution level follows a predetermined order. This violates the uniform sampling condition because samples inside a group cannot be placed until all previous groups at the same resolution level have been sampled. A more detailed analysis about this parallel sampling method can be found in Appendix A.

3.3 Tile-Based Methods

The first tile-based Poisson-disk sampling methods used *Wang tiles* and were proposed by Hiller et al. [2001] and Cohen et al. [2003]. Wang tiles have colors assigned to their edges in specific ways. A Wang tile can only be placed next to another if they share the same color along the common edge. This allows nonperiodic tilings of the plane to be created. The generation of Poisson-disk samples inside each tile must respect the minimum distance constraint across the edges of the tile relative to all other tiles that share the same edge color. The authors achieve this by using several steps of Voronoi relaxation [Lloyd 1982].

In the initial Wang tile methods, the tiling had to be computed in advance inside some finite region of space. Lagae and Dutré [2005] introduced procedural tiling rules that allow a Wang tile to be assigned on-the-fly in a consistent way to any arbitrary point in space. This leads to the creation of infinite nonperiodic tilings of Poisson-disk samples. Lagae and Dutré [2006a] later introduced procedural tiling rules for *corner tiles*. Corner tiles have colors associated to their corners instead of their edges. They can enforce the minimum distance constraint across tiles that share a common corner. The same authors also extended corner tiles to three dimensions, creating *corner cubes* [Lagae and Dutré 2006b].

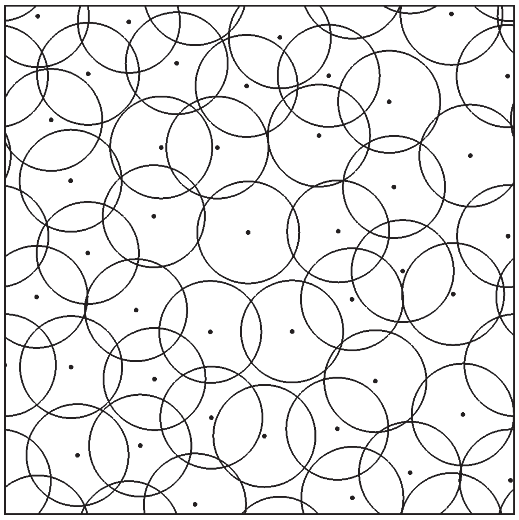
Methods for nonuniform Poisson-disk sampling have been proposed based on tile distributions. Kopf et al. [2006] apply subdivision rules to Wang tiles in order to create sample distributions with varying density across space. Similar subdivision rules can be applied to *Penrose tiles* or *polyominoes* [Ostromoukhov et al. 2004; Ostromoukhov 2007b]. Each Penrose tile or polyomino has a single sample inside, which is subject to a Voronoi relaxation together with the samples from other tiles or polyominoes to reduce sampling artifacts. The parallel sampling method of Wei [2008] can also accommodate nonuniform distributions with the help of a subdivision tree that refines regions of the domain more than others. The generation of nonuniform low-discrepancy sequences of samples can be done with dodecagonal nonperiodic tilings of the plane [Ostromoukhov 2007a]. Low-discrepancy sequences are different from Poisson-disk distributions in that they are deterministic. They can, however, fill the plane without creating noticeable artifacts.

4. POISSON-DISK SAMPLING BY SUBDIVISION REFINEMENT

Our method for Poisson-disk sampling performs a subdivision refinement of the allowable space for the insertion of new samples. It can be used in all applications of Poisson-disk sampling that were discussed in Section 2, with the exception of the direct generation of nonuniform distributions. In the case of the tile-based methods of Section 3.3, it can be used to pregenerate the distributions for the tiles. The method only requires the specification of the distribution radius $0 < r \leqslant \sqrt{n}/2$ as a starting parameter. Given that samples are generated inside the unit hypercube $[0, 1]^n$, values $r > \sqrt{n}/2$ can be supplied to the algorithm but have no practical interest since they are certain to lead to distributions with only one sample. Unlike some of the previous methods, we do not enforce a maximum number of samples to be generated. Samples keep being inserted until there is no more allowable space for new ones and the distribution becomes jammed. There is, however, the option of specifying a desired number N of samples. The algorithm will attempt to reach a number of samples as close as possible to N while still generating a maximal distribution. As with all other Poisson-disk sampling methods, it is possible to perform a Voronoi relaxation at the completion of the algorithm to further smooth the distribution of samples.

In what follows, we will often explain the algorithm in its two-dimensional version for increased clarity of presentation. The extension to three or higher dimensions is straightforward. The samples, in particular, will be represented as disks of radius $2r$ instead of the r half-disks that were shown in Figure 1. The larger disks, shown in Figure 5(a), are now intersecting but a property still holds that no disk contains any sample other than the sample at its center. The same disks are gray shaded in Figure 5(b). The white areas in this image represent the empty space where new samples can be inserted. Once the unit square becomes uniformly colored in gray, the distribution is jammed.

A spatial subdivision data structure is used to mark the allowable insertion space. In two dimensions, this data structure is a quadtree and in three dimensions it is an octree [Samet 1990]. The same type of data structure can be easily extended to higher dimensions. The


(a) disk perimeters

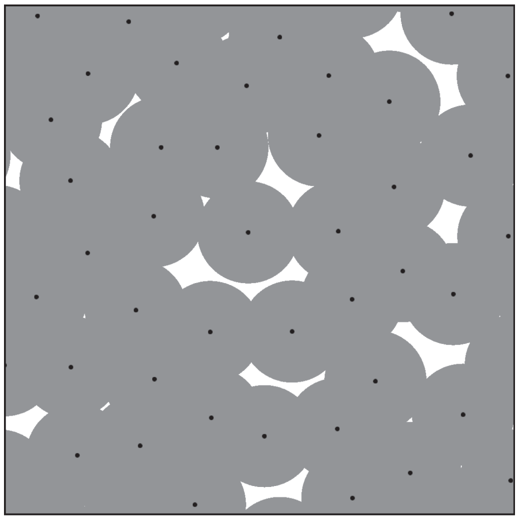
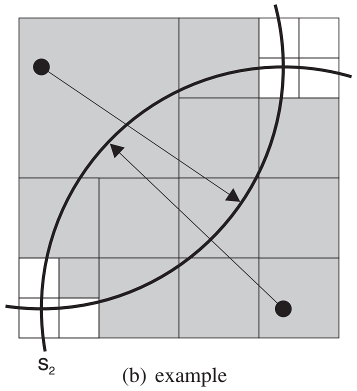

(b) disk areas

Fig. 5. The top image shows an incomplete Poisson-disk distribution visualized with disks of radius $2r$. The bottom image shows the same distribution with gray shaded disks. New samples can be inserted in the white areas.

```
initialise tree with hypercube $[0, 1]^n$;
while tree not empty
  Generate random sample inside tree;
  if sample is valid
    Update tree with sample;
    output sample;
```

(a) pseudocode



(b) example

Fig. 6. The main algorithm on top for Poisson-disk sample generation. The diagram at the bottom shows an example. The gray shaded nodes have already been pruned from the tree. The top right node is a candidate node. All the other nodes that intersect with disks are potential candidates.

main algorithm is shown as pseudocode in Figure 6(a). The routine Generate and Update, invoked by the main algorithm, will be further explained in the following sections. The algorithm is initialized by placing a root node with dimensions [0, 1] × [0, 1] in the tree at subdivision level 0. For purposes of sample generation, leaf nodes in the tree can be classified as *candidates* or as *potential candidates*. A leaf node is a candidate if it is not intersected by the disks of any previously inserted samples, otherwise it is a potential candidate. Figure 6(b) shows an example of a distribution where the node on the top right is a candidate node. Nodes that have already been pruned from the quadtree are shaded in gray; these are nodes that are completely inside the disk of one of the samples and which can be discarded. All the remaining leaf nodes in the tree are potential candidates.

At each iteration, a leaf node is selected by randomly traversing the tree in top-to-bottom fashion. Similar to the methods by Jones [2006] and Dunbar and Humphreys [2006], the area beneath each node in the tree is used to derive the probabilities of choosing one of the child nodes of any given node. This strategy enforces the constraint that samples must be chosen with a uniform spatial probability density. Once a leaf node has been selected, a random sample is generated inside of it. The validity of the sample is then checked by finding its distance to neighboring samples. Every leaf node in the quadtree keeps a list of the samples whose disks intersect with it. If the sample list for a leaf node is empty then it is a candidate node, otherwise it is a potential candidate node. To find if a newly generated sample is valid, therefore, it is only necessary to find its distance to the samples that are kept in the list of the selected leaf node.

If the newly generated sample is found to be valid then it is accepted into the distribution and the tree is updated, to account for the presence of the new sample, by invoking the tree Update routine. If, on the other hand, the new sample is found to be invalid then it must be rejected. The leaf node inside of which the attempted sample was generated is subdivided, as part of the same Generate routine that also generates the sample, and the areas of all nodes in the tree are updated to account for this subdivision. Unlike the original dart-throwing algorithm where a rejected sample represents a wasted computational effort, our algorithm continually improves the accuracy of the quadtree through node subdivision even when samples are rejected. After each iteration, the area represented by the quadtree either remains constant or decreases. The decrease in area is caused by the pruning of nodes from the tree, which happens when either new samples are inserted or leaf nodes are subdivided. The algorithm terminates when the quadtree becomes empty and its area becomes zero, signifying that a maximal distribution has been achieved.

A maximum-subdivision-level condition is important to prevent the algorithm from becoming locked in an infinite loop in situations

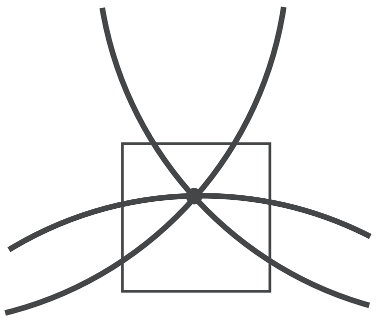


Fig. 7. Three or more disks intersecting at a common point cause a situation where a node is subdivided indefinitely. A maximum subdivision level must be applied to force the algorithm to terminate.

where three or more disks are intersecting at the same point. Figure 7 illustrates this scenario. Although the node shown in this figure is covered by the disks, none of the three disks by themselves provides a complete coverage. The node is deemed to be subdivided and the same problem is going to occur for all the descendants that contain the point of intersection of the disks. The consequences of enforcing a maximum level of subdivision can be formalized by stating that the algorithm generates distributions obeying the following condition.

$$\exists \varepsilon > 0,\ \forall \mathbf{x} \in D,\ \exists \mathbf{x}_i \in X :\ \|\mathbf{x} - \mathbf{x}_i\| < 2r + \varepsilon \quad \text{with} \quad \varepsilon = O(2^{-l_{MAX}}) \qquad (3)$$

The constant ε is arbitrarily small and is related to the size of the leaf nodes at the maximum level of subdivision l_{MAX}. By increasing this maximum level, ε converges to zero, in which case condition (3) converges to the condition (2) of a maximal distribution. Given that the probability of three disks intersecting at the same point is very low, the specification of a sufficiently large maximum subdivision level causes the proposed algorithm to generate maximal distributions almost always except in rare cases where empty areas of small size may remain in the distribution.

4.1 Generating Samples in the Tree

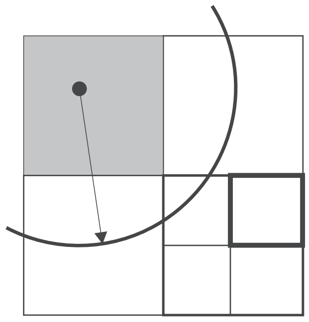
The generation of new samples inside the tree is shown as pseudocode in Figure 8(a). The routine `Generate` is recursive and accepts as arguments a current node in the tree, a reference variable *selsample*, where the generated sample is returned, and the current subdivision level l. The routine also returns, as part of each recursive invocation, the total area δ of the nodes that have been pruned beneath the current node as a result of subdivisions that may have occurred. The area δ is used to control the pruning of further nodes higher in the hierarchy as the recursive call stack is unwound. A node is discarded when the total area of all pruned nodes beneath it is equal to its own area. The routine is initially invoked on the root node of the tree. The reference variable *selsample* is initialized with a null sample.

If the current node is a leaf, the `Generate` routine attempts to place a uniform random sample inside of it. The routine finds if the sample is valid by checking it against all the neighboring samples that are stored in the list of samples for the leaf node. If the sample is valid, the routine places it in the *selsample* reference variable and returns an area of zero since no subdivision occurred and hence no pruning took place. If the sample is invalid then the routine subdivides the current leaf node and returns the area that may have been pruned from the tree as a result of the subdivision. If, however, the maximum subdivision level l_{MAX} has been reached, the routine does not perform any further subdivision and simply returns the total area of the leaf node. This return value will cause the node to be discarded when control returns to the next higher invocation of the `Generate` routine on the call stack. It is at this point that the possibility exists that small unoccupied areas in the domain may be incorrectly discarded when they could still receive a new sample. As expressed by Eq. (3), the probability of incurring such a sampling error can be made arbitrarily small by making the maximum subdivision level l_{MAX} arbitrarily high.

A child of the current node is randomly chosen using the areas of all the children to derive the probabilities of the discrete random event. If a node i has m children with areas $a_j,\ j = 1, \ldots, m$, the probability of child j being chosen is a_j/a_i, where $a_i = \sum_{j=1}^{m} a_j$ is the area of the parent node. A child is chosen when the current node is not a leaf. The `Generate` routine is then recursively invoked

```
Generate($node.\ selsample,\ l$)
if $node$ is a leaf
  place random $sample$ in $node$;
  if $sample$ is not valid
    if $l < l_{MAX}$
      return Subdivide($node$);
    else
      return area of $node$;
  else
    let $selsample = sample$;
    return 0;
let $child$ of $node$ be randomly chosen;
let $\delta$ = Generate($child,\ selsample,\ l+1$);
if $\delta <$ area of $child$
  decrement area of $child$ by $\delta$;
else
  discard $child$ from the tree;
return $\delta$;
```

(a) pseudocode



(b) example

Fig. 8. The `Generate` algorithm. The diagram at the bottom shows an example. Initially, there are three partial candidates and one pruned node. The node on the lower right, shown with medium thickness, is randomly selected with a 3 : 1 probability. If a uniform random sample is placed on the top left of the chosen node, inside the disk of the previous sample, the node is subdivided. In a subsequent iteration, the child shown with maximum thickness is randomly selected with a 4 : 1 probability. Since it is a candidate node, any random sample generated inside of it is guaranteed to be valid.

on the chosen child node. The rest of the routine deals with some book-keeping procedures to manage the areas of the nodes. If the value δ returned from the invocation of `Generate` on a child node is less than the area of the child, then this area is simply decremented to reflect the removal of some of the descendant nodes. Through the recursive invocation of `Generate`, the same value δ is also decremented from all the ascendants of the chosen child node. If, on the other hand, δ is equal to the area of the child node then the child is pruned from the tree. As δ is returned through the unwinding call stack, other ascendants of the child node may also be pruned from the tree.

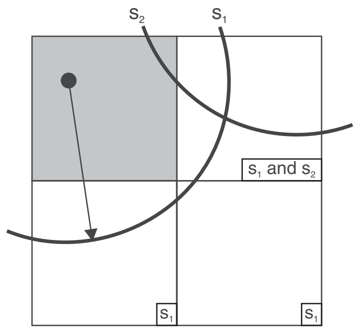
4.2 Subdividing Nodes in the Tree

The pseudocode for the `Subdivide` routine is shown in Figure 9(a). This routine is invoked from within `Generate` and receives a node in the tree as its argument. It returns the area δ of the children that may have been discarded after subdivision. Every child of a subdivided node must be compared against the samples that have already been inserted in the distribution. The children of a node only need to be compared against the samples contained in the node's list. These are the only samples in the whole distribution that can possibly interact with the children for that node. If a sample has a disk that intersects with a child node, that sample is added to the list of samples of the child node. If, on the other hand, a sample's disk completely contains the child node, that child node can be discarded and the value of δ incremented correspondingly.

The tests for intersection or containment between a node and a disk is based upon the sphere-box intersection test first proposed by Arvo [1990] and recently improved by Larsson et al. [2007]. More details about our intersection test are given in Appendix B. This intersection test works for any number of dimensions. Each child is finally appended to the tree below the current node unless it was previously found to be contained inside the disk of one of the node's samples. At the end of the `Subdivide` routine, the parent's list of samples is no longer necessary and is discarded to free up memory. The total value for δ is also returned.

```
Subdivide($node$)
let $\delta = 0$;
generate children of $node$;
for each $child$
  for each $sample$ in $node$'s list of samples
    if $child$ is intersected by $sample$'s disk
      add $sample$ to $child$'s list of samples;
    else if $child$ is contained in $sample$'s disk
      discard $child$;
      increment $\delta$ by area of $child$;
      break from inner loop;
  if $child$ has not been discarded
    add $child$ to tree below $node$;
discard $node$'s list of samples;
return $\delta$;
```

(a) pseudocode



(b) example

Fig. 9. The `Subdivide` algorithm. The diagram at the bottom shows an example for a parent node that intersects samples s_1 and s_2. The gray shaded node is discarded. The small boxes for each leaf node indicate which samples intersect that node.

4.3 Updating the Tree with New Samples

The state of the tree needs to be updated whenever a new sample is inserted in the distribution. This is done by traversing the tree depth-first in recursive fashion. The recursive routine `Update` is shown in Figure 10(a). It accepts a current node in the tree as an argument, together with the sample that has just been inserted. The routine also returns, as part of each recursive invocation, the total area δ of the nodes that have been pruned beneath the current node. The routine is first invoked for the node at the top of the tree. The routine checks the node against the disk of the sample. If the node is completely outside the disk, an early return is made and no pruning occurs. If the node is completely inside the disk then it is pruned together with all its descendants. The area of the node is returned since it corresponds to the area that has been removed from the tree. If neither of these conditions is verified, the node must be intersecting the disk of the new sample. In this case, if the node is a leaf, the new sample is appended to the node's list of samples, otherwise all the node's children are checked in turn against the new sample.

`Update` implements the same book-keeping procedures for the areas of the nodes that the `Generate` routine already implemented. If the pruned area δ returned from a recursive `Update` call to a child is the same as the area of the child, then the latter is removed from the tree, otherwise the area of the child is decremented by δ. The areas pruned from all children of the current node are accumulated in Δ and returned to higher nodes.

4.4 Time and Space Complexity

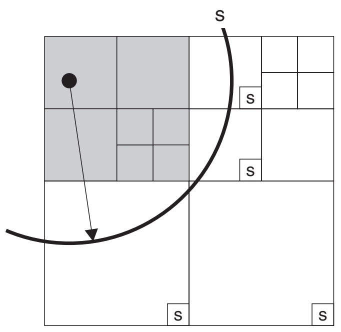
The procedures `Generate`, `Subdivide`, and `Update` have different time complexities. The subdivision of a node in the tree is done in constant time, owing to the list of samples that is kept in the node. This list obviates the need for an exhaustive comparison between the child nodes and every sample in the distribution. Sampling the tree takes logarithmic time since the tree needs to be traversed all the way down to a leaf node. Updating the tree with an accepted sample also takes logarithmic time since a recursive tree traversal is required. These two logarithmic times are dominant in the algorithm since node subdivision is invoked from within the sample generation procedure. As a consequence, for a distribution with N samples, the total time complexity is $O(N \log N)$. The space complexity is also $O(N \log N)$ since a subdivision tree is used.

4.5 Optional Generation of Periodic Distributions

It is sometimes desirable to have a n-dimensional distribution that is n-periodic. This allows the hypercube to wrap around along all of its dimensions. This also allows a simple tiling scheme where

```
Update(node, sample)
if $node$ is outside $sample$'s disk
    return 0;
if $node$ is inside $sample$'s disk
    prune $node$ and all its descendants
    return area of $node$;
if $node$ is a leaf
    append $sample$ to $node$'s list of samples
    return 0;
let $\Delta = 0$;
forall children of $node$
    let $\delta$ = Update($child$, $sample$);
    if $\delta <$ area of $child$
        decrement area of $child$ by $\delta$;
    else
        discard $child$ from the tree;
    increment $\Delta$ by $\delta$;
return $\Delta$;
```

(a) pseudocode



(b) example

Fig. 10. The Update algorithm. The diagram at the bottom shows an example for the insertion of a new sample s. The gray shaded nodes are pruned from the tree as a result of the insertion. Leaf nodes that intersect with the new sample are marked with a small box.

copies of the hypercube are placed side by side to create an infinite periodic tiling that is still a valid Poisson-disk distribution. Periodic tilings are not very interesting because they introduce noticeable repeating patterns but n-periodic distributions are still useful in that they avoid the boundary artifacts that occur with nonperiodic distributions. In the original nonperiodic case, the boundaries of the unit hypercube act as constraints, implying that the probability of a sample being placed outside the hypercube is always zero. This subtle deviation from a uniform probability sampling scheme causes samples to cluster more densely close to the boundaries, trying to use all the available space that is left there.

The enforcement of periodic boundary conditions can be obtained by introducing a small modification in the main algorithm shown in Figure 6. If we define the discrete set $\Sigma = \{-1, 0, +1\}$, it is possible to write the following.

```
for every displacement vector $\sigma \in \Sigma^n$
    Update tree with sample $s + \sigma$;
```

This pseudocode fragment replaces the single invocation of the Update procedure in the algorithm of Figure 6. The generation of periodic distributions is more expensive since 3^n invocations of the Update procedure are required per iteration. Many of these invocations terminate early, however, in the cases where the disk of the offset sample $s + \sigma$ does not intersect with the unit hypercube at the root of the tree.

4.6 Specifying the Desired Number of Samples

There are three different ways of supplying initial parameters to a Poisson-disk sampling algorithm. The basic parameters are the distribution radius r and the total number of samples N that is desired. One can start a Poisson-disk sampling algorithm by specifying r or N or both in conjunction. The specification of r is used when exact control over the distance between samples is required. The specification of N usually leads to sample distributions that are not maximal, as it is difficult to control the sampling process so that it becomes jammed exactly after the Nth sample has been inserted. The radius r is the starting parameter for the proposed sampling algorithm in order to generate maximal distributions where the total number of samples is not constrained. It is possible, however, to specify a value of r so that the resulting distribution is maximal and the total number of samples generated is *approximately* equal to some desired number N.

Lagae and Dutré [2005] unified the parameters r and N by introducing the concept of *relative radius*. The packing density $\gamma_n \in (0, 1)$ of a N sample distribution is the percentage of n-space that is occupied by the hyperspheres of radius r centered around the samples. For a unit hypercube, the packing density is

$$\gamma_n = N V_n(r), \tag{4}$$

where $V_n(r) = \pi^{n/2} r^n / \Gamma(n/2+1)$ is the volume of a n-dimensional hypersphere of radius r and Γ is the gamma function. A distribution has a maximum packing density when the samples are placed deterministically according to a crystalline grid. In two dimensions, for example, this grid is hexagonal. The maximum packing densities $\gamma_{n_{MAX}}$ have been determined for dimensions up to $n = 8$ [Weisstein]. If a distribution with a fixed number N of samples is desired, the maximum possible radius will occur when the hyperspheres occupy the densest possible configuration. For samples generated inside the unit hypercube, the maximum distribution radius is then

$$r_{MAX} = \sqrt[n]{\frac{\gamma_{n_{MAX}}}{N} \frac{\Gamma\left(\frac{n}{2}+1\right)}{\pi^{n/2}}}. \tag{5}$$

The relative radius of the distribution is a parameter $\rho_n \in [0, 1]$ such that the absolute radius becomes $r = \rho_n r_{MAX}$. When $\rho_n = 0$ is specified, no distance constraints are enforced and a Poisson distribution with N samples is generated. When $\rho_n = 1$ is specified, a deterministic placement of N samples is achieved, having the highest possible packing density. It must be remarked, however, that no random sample placement algorithm can attain this result. Poisson-disk sampling algorithms can only generate N samples exactly for relative radii that don't significantly exceed 0.8. For larger values of ρ_n, the probability increases of the distribution becoming jammed before N samples can be inserted. The case $\rho_n = 1$, in particular, is always guaranteed to generate maximal distributions with a number of samples that is smaller than N.

Table I shows the average packing densities γ_n, obtained with the proposed algorithm in two, three, and four dimensions, and compares them with the maximum packing densities $\gamma_{n_{MAX}}$, evaluated with known formulae [Weisstein]. Each average packing density was

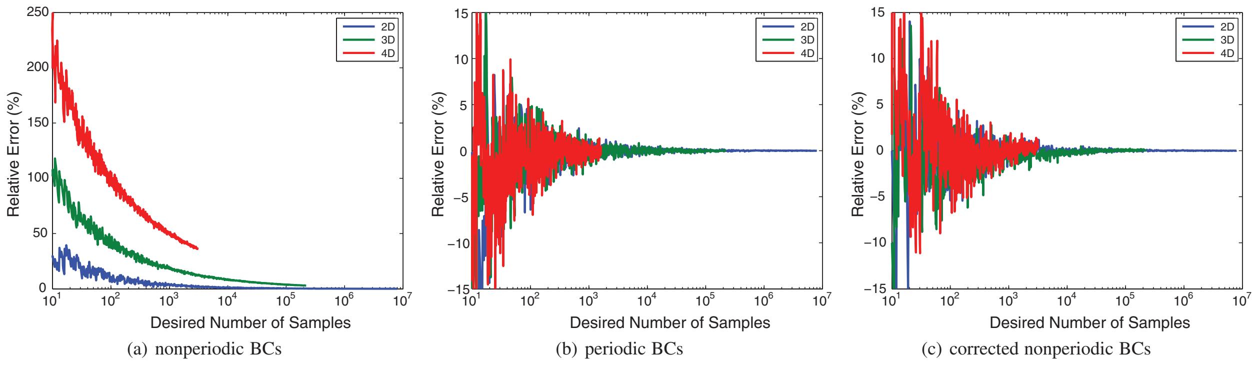


Fig. 11. The relative error between the total number of samples generated N' and the desired number of samples N.

Table I. Average Packing Density, the Maximum Packing Density and the Relative Radius in Several Dimensions

n	γ_n	$\gamma_{n_{MAX}}$	ρ_n
2	0.5470	0.9069	0.7766
3	0.3841	0.7405	0.8035
4	0.2599	0.6169	0.8057

Table II. Parameters for the Error in the Number of Samples Due to the Presence of Boundary Effects

n	α	β
2	1.0997	−0.4999
3	2.2119	−0.3538
4	4.1114	−0.3056

obtained from Eq. (4) by averaging N over 100 runs of the algorithm for a given value of r. The radius r was chosen so as to generate the maximum possible number of samples per run, given the hardware's memory constraints, providing, therefore, a good level of accuracy. Periodic boundary conditions were enforced to prevent boundary effects from contaminating the results. The average packing densities are close to half of their maximum possible values, signifying that there is still a significant degree of randomness in the generated distributions. Given these average packing densities, a distribution with N samples can be generated with the following distribution radius.

$$r = \sqrt[n]{\frac{\gamma_n}{N}\frac{\Gamma\left(\frac{n}{2}+1\right)}{\pi^{n/2}}} \tag{6}$$

The radius r from Eq. (6) amounts to a relative radius that results from dividing Eq. (6) by Eq. (5), leading to $\rho_n = (\gamma_n/\gamma_{n_{MAX}})^{1/n}$. The relative radii are also shown in Table I. They are within the region for which the sampling algorithm is able to generate maximal distributions without becoming jammed before a number N of samples is reached. The radius obtained with Eq. (6) is then supplied as the starting parameter to the proposed algorithm, given some desired number N.

Figure 11 shows the relative error $e = (N' - N)/N$, in two, three, and four dimensions, between the number of samples generated N' and the desired number of samples N after the radius (6) has been used. Graphs are shown for the algorithm without (Figure 11(a)) and with (Figure 11(b)) periodic boundary conditions. The boundary effects that are present when periodicity is not enforced cause an increase in sample density around the boundaries of the hypercube and this, in turn, causes the packing densities to deviate from the values shown in Table I. The final consequence is that Eq. (6) becomes less accurate at controlling the number of generated samples, as the graph of Figure 11(a) shows. The error of Eq. (6), in this case, decreases with increasing N, and, correspondingly, with decreasing r, because the boundary effect is manifested over a progressively smaller boundary region. In the case where periodicity is enforced, the graph of Figure 11(b) shows good agreement between the desired number of samples and the number of samples actually generated.

The error due to the presence of boundary effects, shown in Figure 11(a), can be represented on average in the form $e = \alpha N^{\beta}$. A logarithmic regression was used to estimate the parameters α and β, which are shown in Table II for two, three, and four dimensions. Starting from the definition of the relative error e and rearranging terms, one obtains

$$N' = N + \alpha N^{\beta+1}. \tag{7}$$

This equation can be used to correct for the presence of boundary effects in the nonperiodic case. The desired number of samples is supplied as the value for N' and the equation is solved for $N < N'$. The value of N is then used in Eq. (6) to obtain the distribution radius. Because Eq. (7) is nonlinear for N, it must be solved with numerical methods. A Newton-Raphson root finder is used, starting with $N = 1.0$, which only requires a few iterations to achieve good accuracy [Press et al. 1992]. Figure 11(c) shows the relative error after the value N has been internally corrected to account for the boundary effects. The error is now similar to the error generated with periodic boundary conditions, quickly decreasing as the number of samples increases.

5. RESULTS AND DISCUSSION

We compare our method against previous methods for the generation of accurate Poisson-disk sample distributions in two dimensions. The methods used for comparison are:

—the molecular dynamics algorithm by Skoge et al. [2006];
—the Voronoi decomposition algorithm by Jones [2006];
—the accurate scalloped sector algorithm by Dunbar and Humphreys [2006];

Table III. Timing Results in Seconds for Several Poisson-Disk Sampling Algorithms (in two dimensions with wall boundary conditions)

	N		
Algorithm	1 000	10 000	100 000
Skoge (accurate)	0.760s	10.521s	481.916s
Jones (accurate)	0.245s	2.563s	27.140s
Gamito (accurate)	0.013s	0.157s	1.938s

—the approximate scalloped sector algorithm by Dunbar and Humphreys [2006][2].

When performing the comparison, we generate maximal distributions with all the methods except for the molecular dynamics algorithm that cannot generally produce such distributions. Although the dart-throwing algorithm of Dippé and Wold [1985] is the basis for all spatially guided accurate Poisson-disk sampling algorithms, it cannot be used for comparison purposes because it does not have a guaranteed termination time when maximal distributions are generated. The sourcecode for the algorithms used in the comparison has been made publicly available by their respective authors. We would have liked to include the method by White et al. [2007] in the comparison, as it also generates accurate and maximal distributions, but it was not possible to obtain the sourcecode from the authors. From the results presented in their paper, the method of White et al. [2007] is likely to be currently the fastest accurate method in two dimensions. The approximate sampling algorithm by Dunbar and Humphreys [2006] is included in the comparison to provide an idea of the speedups that can be achieved when one or both of the conditions for Poisson-disk sample generation are relaxed. For simplicity, we henceforth refer to the algorithms by the name of their respective first authors.

Table III compares timing results between the proposed algorithm and the algorithms by Jones and by Skoge for the computation of two-dimensional distributions with N samples. For the algorithm by Jones [2006], the timings for the desired number N of samples were achieved by trying several values of the distribution radius r. The timing results were then extrapolated from the actual number of samples in the distribution (when sufficiently close to N) to the desired value for N. In the case of the algorithm by Skoge et al. [2006], the number N of molecules was specified as an input parameter while all the other parameters, such as the growth rate of the molecular radius, were left at their default values as given in the sourcecode. The simulations were stopped once the radii given by Eq. (6) were reached, preventing the molecules from progressing towards deterministic distributions. The resulting distributions were visualized and it was verified that they were not maximal. In the case of our proposed algorithm, the technique described in Section 4.6 was used to generate distributions close to the desired number of samples, followed by extrapolation. The number of samples generated was 1 001, 10 025, and 100 047, respectively. The timings were obtained on a dual AMD Athlon MP2600 2.1 GHz machine with 4Gb of main memory. The molecular dynamics algorithm has the ability to minimize the space between the disks, creating almost hexagonal packings. The other algorithms can only approach the same result with the use of Voronoi relaxation [Lloyd 1982]. The molecular dynamics simulations, however, do not scale well with increasing N and this becomes worse for higher dimensions.

Table IV. Timing Results in Seconds for Several Poisson-Disk Sampling Algorithms (in two dimensions with n-periodic boundary conditions)

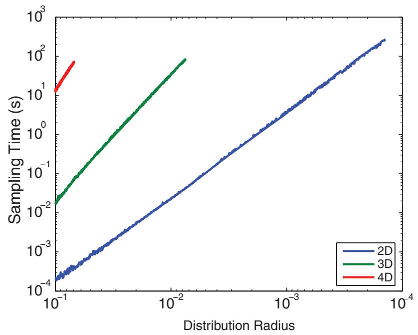
	N		
Algorithm	1 000	10 000	100 000
Skoge (accurate)	0.763s	10.854s	427.585s
Dunbar (accurate)	0.491s	4.884s	47.874s
Gamito (accurate)	0.014s	0.162s	1.984s
Dunbar (approximate)	0.005s	0.052s	0.521s

Table IV compares timing results similar to those of Table III but using n-periodic boundary conditions instead. The algorithms by Dunbar and Humphreys [2006] are only included in this comparison because they employ a toroidal mapping in two dimensions that corresponds to enforcing a periodicity condition. For the opposite reason, the algorithm by Jones [2006] is not included because the Voronoi decomposition part of the algorithm does not enforce periodicity. The approximate sampling algorithm by Dunbar and Humphreys [2006] is orders of magnitude faster than all the others while our own algorithm, based on subdivision refinement, is the fastest of all the accurate sampling algorithms that were compared. Comparing Tables III and IV, for the two algorithms that can generate both periodic and nonperiodic distributions, there does not appear to be a significant difference in the sampling times for the two boundary conditions.

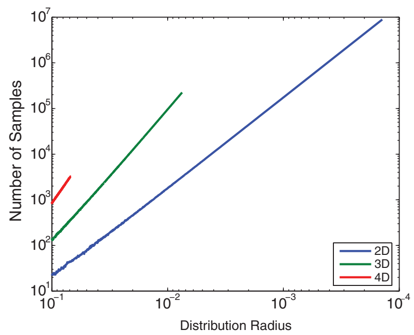
Figure 12 shows statistics of our Poisson-disk sampling algorithm in two, three, and four dimensions that were obtained by gradually decreasing the distribution radius, starting from the value $r = 0.1$. The running time of the sampling algorithm is shown in Figure 12(a), the number of generated samples for each value of r is shown in Figure 12(b), and the average sampling rate (sample insertions per second) is shown in Figure 12(c). For each dimension, the statistics are obtained down to a minimum value of r for which the subdivision tree still fits in the 4Gb of memory of the machine used in the tests. Samplings for radii smaller than this minimum value are possible but the timings will increase significantly due to memory page faults. The minimum radii were found to be approximately $r = 0.00014$ for $n = 2$ dimensions, $r = 0.0075$ for $n = 3$ dimensions, and $r = 0.0685$ for $n = 4$ dimensions. The statistics for the Poisson-disk sampling algorithm running with periodic boundary conditions were found to be essentially similar to the graphs shown in Figure 12. The most notable difference is that a smaller number of samples is generated for every value of r. This difference is only visible in the graphs for small radii when the boundary effect is more pronounced.

Figure 13(a) shows an estimate of the average power spectrum for two-dimensional Poisson-disk sampling, obtained by averaging 100 independent runs of our algorithm for the case $r = 0.005$. Periodic boundary conditions were used so that the discrete Fourier transform naturally becomes a discrete Fourier series without concerns for boundary effects. Figures 13(b) and 13(c) show the radial power spectrum and the anisotropy, respectively. The curves for radial power and anisotropy fit well with the reference curves for two-dimensional Poisson-disk sampling that were obtained by Lagae and Dutré [2008]. The radial power and anisotropy spectra were numerically computed using standard techniques that integrate the average power spectrum inside successive concentric annuli in the frequency domain [Ulichney 1988; McCool and Fiume 1992]. In Appendix A, it is demonstrated that our sampling algorithm generates correct Poisson-disk distributions. The spectra of Figure 13, therefore, are true representations of a blue noise process apart from the noise caused by averaging a relatively small number of sampling

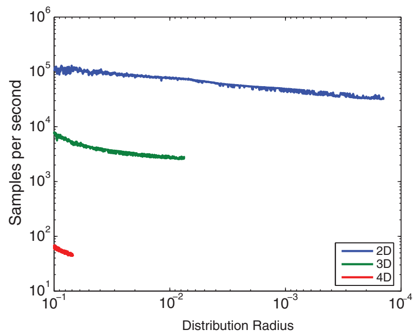
[2]Dunbar and Humphreys actually proposed three algorithms in their paper: one accurate algorithm with $O(N \log N)$ complexity and two approximate algorithms with $O(N)$ complexity. The algorithm that results from collapsing scalloped sectors into arcs of a circle is the fastest of the two approximate algorithms and is the one we use for the purpose of comparison.



(a) total computation time

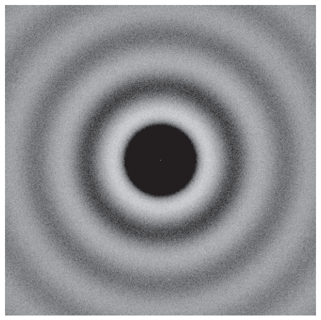


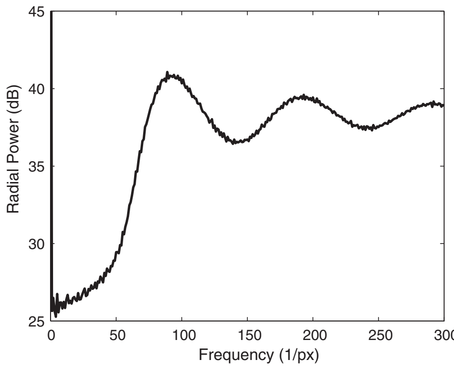
(b) number of samples



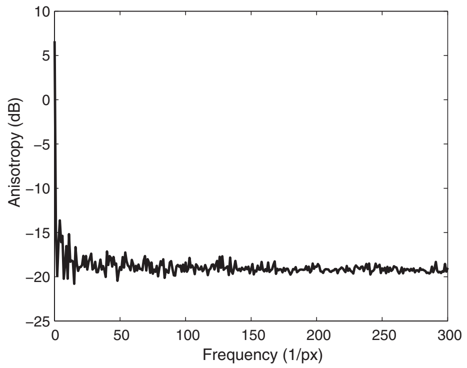
(c) samples per second

Fig. 12. Statistics for the Poisson-disk sampling algorithm in two, three, and four dimensions. The distribution radius in all graphs decreases from left to right along the horizontal axis to illustrate that complexity is larger for smaller radii.


(a) two-dimensional spectrum



(b) radial power



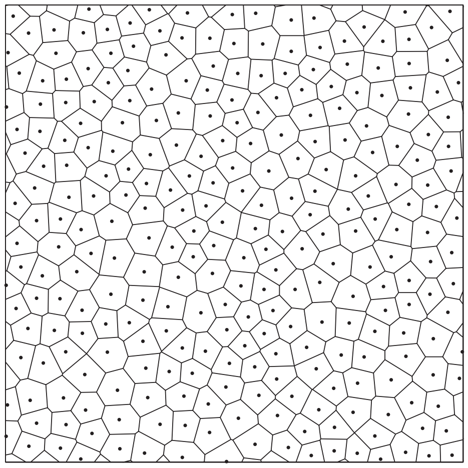
(c) anisotropy

Fig. 13. The average power spectrum for two-dimensional Poisson-disk sampling with $r = 0.005$, the radial power spectrum, and the anisotropy spectrum. The radial power and anisotropy spectra were computed from the 2D spectrum.

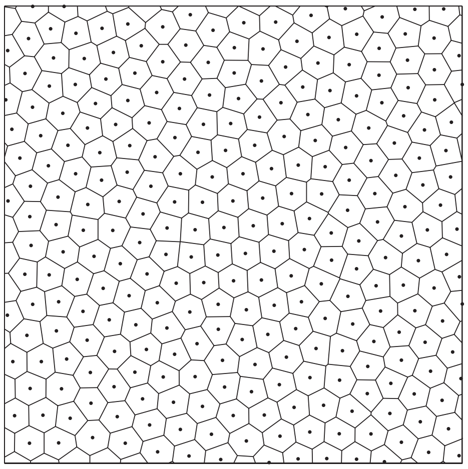
runs. Spectra for three and four dimensions were not estimated since they are a priori guaranteed to be similar to the ones shown in Figure 13.

For the `Generate` algorithm of Figure 8, we employ a maximum subdivision level $l_{MAX} = 24$, which corresponds to the 24 bits of precision (including the implicit lead bit) of a single precision floating point number. This means that we are able to represent sample positions accurately within even the smallest of tree nodes with single precision floating point coordinates. The hypervolume of any node in the subdivision tree is stored as a $32n$-bit fixed point number, for n dimensions, thereby eliminating any round-off that might occur when the hypervolumes are decremented due to pruning. The presence of floating point round-off error could lead to an incorrect decision not to remove a node from the tree due to the less-than test between δ and the node's hypervolume in the `Generate` and `Update` algorithms of Figures 8 and 10, respectively. The size of any empty space left after the maximum subdivision level has been reached is bounded by $0 < \varepsilon < 2^{-l_{MAX}}\sqrt{n}$ (refer to the discussion concerning Eq. (3)). In two dimensions, for example, we have $\varepsilon < 4.2 \times 10^{-8}$. It was verified experimentally that no empty space was found in the generated distributions for $n = 2$. In three and four dimensions, only a very small number of leaf nodes at the maximum subdivision level were left unresolved at the completion of the algorithm with the possibility that some empty space may have been contained inside of them. As Figure 7 illustrates, a leaf node left unresolved at the maximum subdivision level when the algorithm completes does not necessarily contain empty space where new samples could be placed. A maximum subdivision leaf node is left unresolved when it cannot be categorized as fully inside one of the distribution samples or fully outside of every distribution sample.

Figure 14 shows the result of performing 50 steps of Voronoi relaxation on a two-dimensional distribution, generated with our algorithm, with parameter $r = 0.0025$. Only part of the sampling domain is shown for increased clarity. Voronoi relaxation creates a more regular distribution of samples but may cause the conditions for correct Poisson-disk sampling to be violated. To study the regularity of a Poisson-disk distribution, we measured all the distances r_{ij} between the ith and the jth neighbor samples. The condition $r_{ij} \in [2r, 4r)$ must hold for a collection of samples that is a valid Poisson-disk distribution with radius r and that is also maximal. The lower bound is a restatement of the minimum distance constraint for Poisson-disk samples. The upper bound must not be larger than or equal to $4r$, otherwise it would be possible to insert a new sample in-between samples i and j and the distribution would not be maximal. We define the parameters $r_{\text{MIN}} = \min\{r_{ij}\}$, $r_{\text{MAX}} = \max\{r_{ij}\}$ and the relative spread $\sigma = (r_{\text{MAX}} - r_{\text{MIN}})/(2r)$. Table V shows these parameters for the distribution of Figure 14 before and after Voronoi relaxation. Before relaxation, the distribution has a nearly maximal spread and is characterized by an irregular Voronoi decomposition with Voronoi cells of several different sizes. Voronoi relaxation minimized the spread in the distribution but also caused at least a pair of samples to have a distance $r_{ij} < 2r$, as can be seen in Table V. There is a global minimum $\sigma_r = 0\%$ that corresponds to a deterministic hexagonal packing but which is very rarely achieved.



(a) before



(b) after

Fig. 14. A Voronoi decomposition of a distribution generated by our algorithm before and after Voronoi relaxation.

Table V. Minimum Distance r_{MIN}, Maximum Distance r_{MAX} and Relative Spread σ_r for a Two-Dimensional Poisson-Disk Distribution with $r = 0.0025$

	r_{MIN}	r_{MAX}	σ_r
Before relaxation	0.0050000	0.0099887	99.77%
After relaxation	0.0049132	0.0089410	80.56%

The relaxation scheme, in this case, settled for a local minimum at $\sigma_r \approx 80\%$, leading to a more even distribution of the samples.

If we consider the distances d_{ij} between the ith Poisson-disk sample and the jth vertex of its corresponding Voronoi cell, it is possible to show that the distribution of such distances must be in the range $d_{ij} \in [2r/\sqrt{3}, 2r]$. Table VI shows the minimum and maximum values of d_{ij} for the distribution of Figure 14, together with the relative spread $\sigma_d = (d_{MAX} - d_{MIN})/(2r(1 - 1/\sqrt{3}))$. The results are within the correct range of approximately [0.0028867, 0.005] both before and after Voronoi relaxation. The reduction in the spread σ_d is quite significant when compared with the reduction of σ_r and corresponds to the very even distribution of Voronoi cells in Figure 14(b).

Table VI. Minimum Distance d_{MIN}, Maximum Distance d_{MAX} and Relative Spread σ_d for a Two-Dimensional Poisson-Disk Distribution with $r = 0.0025$

	d_{MIN}	d_{MAX}	σ_d
Before relaxation	0.0028869	0.0049999	99.98%
After relaxation	0.0033356	0.0045219	56.14%

Figure 15 illustrates the use of a four-dimensional Poisson-disk distribution. It shows a sphere hypertextured with a blobby model where each blob is centered on a Poisson-disk sample. The hypertexture is animated by sweeping a unit cube through the unit four-dimensional hypercube. The use of Poisson-disk sampling ensures that all blobs are conveniently spaced apart in both space and time while still allowing the blobs to blend along the boundaries of their respective Poisson-hyperspheres. Periodic boundary conditions were enforced so that the resulting animation can be cycled.

6. CONCLUSIONS AND FUTURE DEVELOPMENTS

We have presented an algorithm for generating Poisson-disk sample distributions in n-dimensional space. The samples are generated inside the canonical domain $D = [0, 1]^n$, which can be subsequently modified by the application of any Euclidean transform without changing the Poisson-disk nature of the distribution. The algorithm generates correct Poisson-disk distributions with the samples being uniformly distributed in D and with the distance between every pair of samples being equal to or greater than a specified distance $2r$, where r is the distribution radius. The algorithm also generates maximal distributions, in the sense that no new samples can be further inserted in D without violating the minimum distance constraint relative to other samples. Exceptions occur when there is a point $\mathbf{x} \in D$ that is at an almost equal distance of $2r$ to three or more samples. Depending on the maximum subdivision level of the tree, the algorithm may fail to place an additional valid sample at $\mathbf{x}$. The maximum subdivision level used in our implementation is large enough for these exceptions to occur very rarely. In the worst case, if $\mathbf{x}$ is at a distance of exactly $2r$ to three or more samples, the algorithm will fail to place a sample at $\mathbf{x}$ irrespective of the maximum subdivision level.

We have shown that the algorithm is computationally efficient owing to the use of a subdivision tree that tracks the diminishing subset of D where new samples can be inserted. In principle, the algorithm can be applied to any number n of dimensions due to the ease with which the tree can be expressed in any n-dimensional space. All the procedures that were given in pseudocode and which form part of the algorithm are independent of the number of dimensions, leading to a code implementation that does not require the handling of special cases for particular values of n. The subdivision tree, however, can be memory intensive and this imposes restrictions on how large n and how small r can be. We have also introduced a simple technique to generate Poisson-disk distributions that are maximal and have a number of samples approximately equal to a desired number N.

We have successfully demonstrated our algorithm in two, three, and four dimensions since these are the dimensions more commonly used when generating Poisson-disk distributions for computer graphics applications. Our algorithm can also be applied to many other fields in cases where high-dimensional integrals need to be computed and where Poisson-disk sampling can be applied as part of a variance reduction technique for Monte Carlo integration.

There are several possible ways by which the proposed sampling algorithm can be improved and we mention a few here. Sampling can

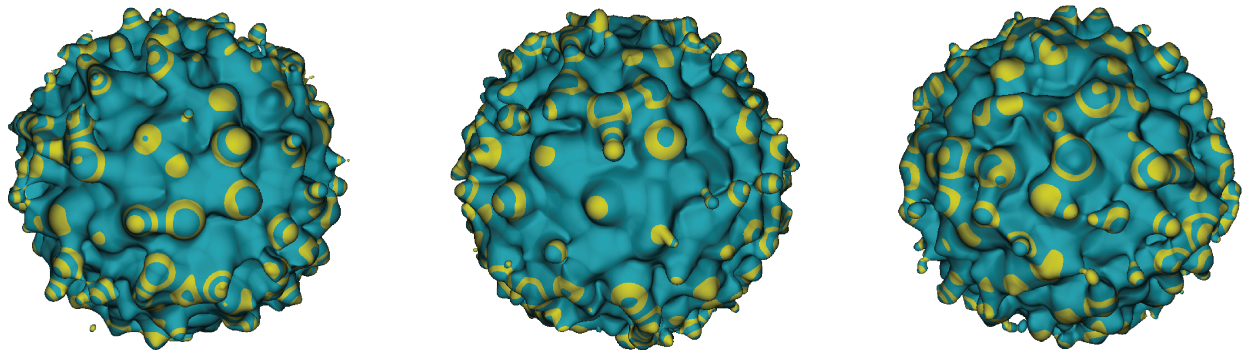


Fig. 15. Three snapshots from an animation showing a sphere hypertextured with a time-varying blobby model. The centeres of the blobs are Poisson-disk samples distributed inside a four-dimensional hypercube.

be done inside domains with irregular boundaries and not just inside a hypercube. Subdivision must be performed for tree nodes that straddle the boundary of the domain, together with the subdivision that already occurs for nodes that intersect sample hyperspheres. This strategy is likely to be more efficient than simply generating samples inside a hypercube that contains the domain and rejecting those samples that fall outside the boundary of the domain.

The Poisson-disk sampling algorithm can be extended to spherical surfaces. The subdivision takes place in a 2-dimensional parameter space of spherical coordinates. Distances between samples must be computed along geodesic lines, which, in the case of a sphere, is done trivially. The intersection routine will have to be modified to compute the intersection between a spherical disk and the spherical sector that corresponds to a given square node in parameter space. Random traversal of the subdivision tree must also take into account the areas of the spherical sectors generated by the tree nodes so that a uniform sampling on the sphere is achieved. Random sample insertion inside the parameter space of a tree node must obey a specific probability density function that transforms into a uniform probability density when the node is mapped onto the sphere. Periodic boundary conditions are imposed on the longitudinal boundary of the parameter space. Extensions to other types of parametric manifolds may also be possible although the computation of geodesic distances becomes more involved.

The proposed algorithm can be parallelized on multicore processors or on multiprocessor machines. The parallelization is achieved by having several independent threads performing tree traversal and sampling insertion simultaneously. Some careful synchronization among the threads is necessary to ensure that the tree data structure remains consistent. For example, if a thread prunes a section of the tree after inserting a new sample, other threads that are working inside the same tree section need to be notified so that they can abort and proceed to a new tree traversal. If a subdivision tree is sufficiently deep, the interference between different threads should be minimal. Parallelization on a GPU, as in Wei [2008], does not seem possible, unfortunately, since GPUs lack synchronization mechanisms between the hardware shaders.

APPENDIXES

A. STATISTICAL PROPERTIES OF DART-THROWING ALGORITHMS

Dart-throwing algorithms can be formalized as a sequential sampling process that places a set of N samples $\{\mathbf{x}_1, \mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in a unit hypercube domain $D = [0, 1]^n$ in n dimensions. After $i < N$ samples have been placed, a possibly disjoint allowable set $S_i \subseteq D$ can be defined where the next sample $\mathbf{x}_{i+1}$ has to be placed. The allowable set is given by

$$S_i \;=\; \{\mathbf{x} \in [0, 1]^n : \;\|\mathbf{x} - \mathbf{x}_j\| \geqslant 2r,\;\; j = 1, 2, \ldots, i\}. \qquad \text{(A.1)}$$

The set S_i represents the portion of the unit hypercube that is not covered by the union of hyperspheres of radius $2r$ centered on the samples $\mathbf{x}_1$ to $\mathbf{x}_i$ that have already been placed. It is represented in green in Figure 16. Once S_i has been defined with (A.1), Poisson-disk sampling dictates that the next sample $\mathbf{x}_{i+1}$ must be placed within S_i with a uniform probability density. This means that for any subset $s \subset S_i$, the conditional probability that $\mathbf{x}_{i+1}$ is placed in s, knowing that $\mathbf{x}_{i+1} \in S_i$, is given by

$$P(\mathbf{x}_{i+1} \in s \mid \mathbf{x}_{i+1} \in S_i) \;=\; a/A_i, \qquad \text{(A.2)}$$

where a and A_i are the hypervolumes of the sets s and S_i, respectively. We have that $A_i < 1$, except for $A_0 = 1$, which is the hypervolume of the unit hypercube: the allowable set $S_0 = [0, 1]^n$ at the time when the first sample $\mathbf{x}_1$ is placed. This iterative formulation of the Poisson-disk process was first presented by Diggle et al. [1976], who called it *simple sequential inhibition*.

An example of a subset s is shown in orange in Figure 16. After sample $\mathbf{x}_{i+1}$ has been inserted, the next allowable set $S_{i+1} \subset S_i$ (with $A_{i+1} < A_i$) can be defined similarly to (A.1) and the procedure can be iterated until a maximum number N of samples has been reached for which $S_N = \emptyset$ (and $A_N = 0$). With S_N being the empty set, no more samples can be inserted and the distribution is said to be *maximal*. The end result is a uniform distribution of N samples in the unit hypercube where the distance between any pair of samples is never smaller than $2r$, with r being called the *distribution radius*.

A.1 Naïve Dart-Throwing

The naïve dart-throwing algorithm of Dippé and Wold [1985] was the first algorithm to accurately generate Poisson-disk distributions and is the benchmark against which all other Poisson-disk sampling algorithms are compared in terms of sampling quality. Dart throwing applies a principle of *rejection sampling* in order to generate a sample $\mathbf{x}_{i+1}$ within an arbitrarily complex set S_i with uniform probability density. Samples are repeatedly generated within the unit hypercube and rejected if they are found to be outside of S_i. The first sample that is found to be inside of S_i becomes the next sample $\mathbf{x}_{i+1}$ in the distribution. The conditional probability of $\mathbf{x}_{i+1}$ being inside any subset s, knowing that it has been accepted in the

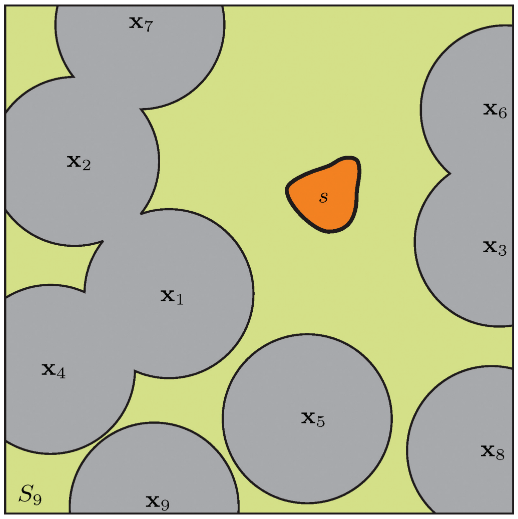


Fig. 16. The allowable set S_9, shown in green, after samples $\mathbf{x}_1, \ldots, \mathbf{x}_9$ have been inserted. The next sample $\mathbf{x}_{10}$ must be inserted in this set. The set s, shown in orange, is an arbitrary subset of S_9.

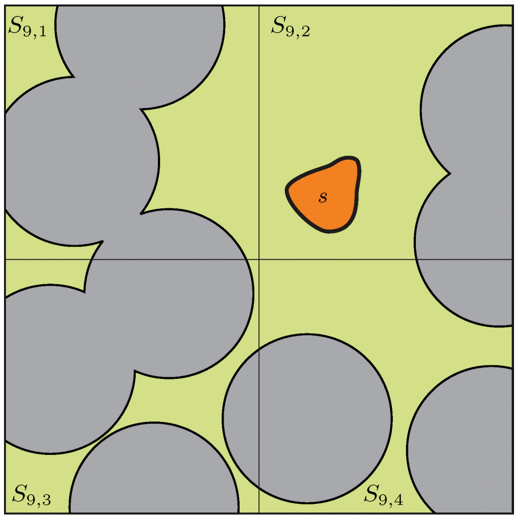


Fig. 17. The allowable set S_9 is split into four smaller subsets $S_{9,1}$ to $S_{9,4}$, shown in green, as a result of domain subdivision. The set s, shown in orange, is an arbitrary subset of $S_{9,2}$.

distribution, is

$$P(\mathbf{x}_{i+1} \in s \,|\, \mathbf{x}_{i+1} \in S_i) = \frac{P(\mathbf{x}_{i+1} \in s \cap \mathbf{x}_{i+1} \in S_i)}{P(\mathbf{x}_{i+1} \in S_i)}. \quad \text{(A.3)}$$

Because $s \subset S_i$, we have that $P(\mathbf{x}_{i+1} \in s \cap \mathbf{x}_{i+1} \in S_i) = P(\mathbf{x}_{i+1} \in s)$ and we obtain, as desired,

$$P(\mathbf{x}_{i+1} \in s \,|\, \mathbf{x}_{i+1} \in S_i) = \frac{P(\mathbf{x}_{i+1} \in s)}{P(\mathbf{x}_{i+1} \in S_i)} = \frac{a}{A_i}. \quad \text{(A.4)}$$

The main problem with naïve dart-throwing is the number of attempts that need to be taken before a sample can be accepted in the distribution. The probability of a sample being accepted is A_i and, correspondingly, the probability of it being rejected is $1 - A_i$. The probability of sample $\mathbf{x}_{i+1}$ being accepted after $k > 0$ attempts obeys a discrete *geometric distribution*, with the expression

$$P_k = A_i(1 - A_i)^{k-1}. \quad \text{(A.5)}$$

As required, $\sum P_k = 1$ when $0 < A_i \leqslant 1$, meaning that sample $\mathbf{x}_{i+1}$ will inevitably be accepted at some point as long as there is a nonzero hypervolume A_i where it can land. The expected number of attempts before the sample can be accepted is given by

$$E[k] = \sum_{k=1}^{\infty} kP_k = A_i \sum_{k=1}^{\infty} k(1 - A_i)^{k-1} = 1/A_i. \quad \text{(A.6)}$$

So, as the sampling progresses and the allowable sets S_i shrink in size, the hypervolumes A_i converge to zero and, correspondingly, the expected number of attempts before accepting a sample goes to infinity. Naïve dart-throwing becomes progressively less efficient the more the hypercube becomes filled with samples. It may take an inordinately large amount of time for dart-throwing to terminate if a maximal Poisson-disk distribution is desired.

A.2 Parallel Dart-Throwing

We discuss here the parallel dart-throwing algorithm of Wei [2008] as it is currently the most advanced Poisson-disk sampling algorithm, capable of running in parallel on a GPU. The unit hypercube domain is progressively subdivided into increasing resolution levels. For each resolution level, the cells are separated in a clever way into distinct groups so that cells in the same group can be sampled in parallel without the risk of conflict between samples. Sampling inside a cell is done by performing k dart-throwing attempts. A sample will not be inserted in the cell if all k attempts fail. Once all cells in a group have been sampled, the algorithm moves on to the next group at the same resolution level.

Figure 17 shows an example where the domain has been decomposed into a 2×2 resolution level. At this shallow level there are four groups of cells where each group has only one cell in it. For each resolution level, a fixed scanning order is used to visit all the groups. In this example, we consider the case that $S_{i,1}$ is sampled first, followed by $S_{i,2}$. The conditional probability that the sample $\mathbf{x}_{i+1}$ is placed in s, knowing that it is a valid sample, is given by the probability that all k dart-throwing attempts in $S_{i,1}$ have failed multiplied by the probability that a sample uniformly generated inside $S_{i,2}$ will lie in the set s.

$$P(\mathbf{x}_{i+1} \in s \,|\, \mathbf{x}_{i+1} \in S_i) = (1 - 4A_{i,1})^k \frac{a}{A_{i,2}} \quad \text{(A.7)}$$

The factors $A_{i,j}$ are the hypervolumes of their respective sets $S_{i,j}$. The probability (A.7) is not the same as (A.2). The asymmetry expressed in (A.7) exists for any resolution level. The fixed scanning order that is used to loop over all cell groups within a level means that the sampling within a group will depend on the sampling done in the previous groups. Nevertheless, Wei [2008] has shown that sample distributions obtained with the parallel sampling algorithm have good blue noise properties. This is likely because the sampling asymmetry is smeared out over successive resolution levels. Different scanning orders are used for different levels. At the resolution level that follows the one shown in Figure 17, for example, the scanning might proceed from right to left instead of the left-to-right order that was used before.

The smearing of the asymmetry expressed in (A.7) works better if a large number of resolution levels is used. Such a number is determined by the distribution radius so that the lateral size of cells at the highest resolution level must be smaller than $2r/\sqrt{n}$. The

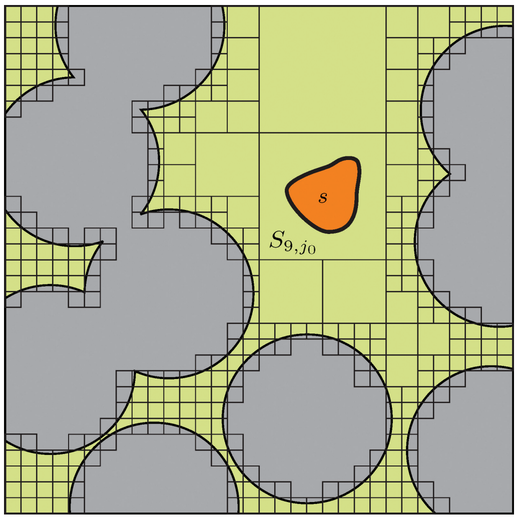


Fig. 18. The allowable set S_9, shown in green, after five steps of recursive subdivision. Nodes that fall outside of S_9 are discarded. The set s, shown in orange, is an arbitrary subset of S_9. S_{9,j_0} is the candidate node where the set s is contained.

number of resolution levels is then equal to $\lceil \log_2 \sqrt{n}/r \rceil$. If r is large, the number of resolution levels is small and artifacts due to sampling asymmetry may arise more easily. As r decreases, more resolution levels are introduced and the quality of the distribution improves.

A.3 Dart-Throwing by Subdivision Refinement

Our algorithm relies on a recursive subdivision of the unit hypercube. Nodes that fall outside of S_i are discarded and the subdivision tree keeps those nodes that are either inside of S_i or that straddle the boundary of that set. One of the leaf nodes $S_{i,j}$, with $j = 1, \ldots, L$, is randomly chosen for sample placement. In order to ensure that sample placement has a uniform density probability in S_i, the probability of choosing a particular leaf node $S_{i,j}$ is proportional to its hypervolume $A_{i,j}$. Figure 18 shows a situation where the arbitrary set s is inside a candidate node S_{i,j_0} in the subdivision tree. The conditional probability of the sample $\mathbf{x}_{i+1}$ being inside s, knowing that it is a valid sample, is equal to the probability of choosing S_{i,j_0} multiplied by the probability that a sample uniformly generated inside S_{i,j_0} will lie in the set s.

$$P(\mathbf{x}_{i+1} \in s \mid \mathbf{x}_{i+1} \in S_i) = \frac{A_{i,j_0}}{\sum_{j=1}^{L} A_{i,j}} \frac{a}{A_{i,j_0}} = \\ = \frac{a}{\sum_{j=1}^{L} A_{i,j}} \tag{A.8}$$

Our algorithm employs a lazy subdivision strategy. Rather than subdividing all of the tree before performing sample placement, the nodes are subdivided as part of the top-down random traversal of the tree if they are found to straddle the boundary. The end result is the same as if the tree had been subdivided in advance. In the limit of an infinite number of subdivisions, we have

$$P(\mathbf{x}_{i+1} \in s \mid \mathbf{x}_{i+1} \in S_i) = \frac{a}{\sum_{j=1}^{\infty} A_{i,j}} = \frac{a}{A_i}, \tag{A.9}$$

which is the same as (A.2). If, however, S_{i,j_0} is a potential candidate node instead of a candidate node (meaning that some part of it is outside S_i), the probability of placing a valid sample in s then becomes

$$P(\mathbf{x}_{i+1} \in s \mid \mathbf{x}_{i+1} \in S_i) = \frac{A_{i,j_0}}{\sum_{j=1}^{L} A_{i,j}} \frac{a}{A_{i,j_0} - a_{i,j_0}} = \\ = \frac{a}{\sum_{j=1}^{L} A_{i,j}} \frac{1}{1 - a_{i,j_0}/A_{i,j_0}}, \tag{A.10}$$

where $a_{i,j_0} < A_{i,j_0}$ is the hypervolume of the part of S_{i,j_0} that is outside of the valid set S_i. As subdivision progresses, the shape of the node S_{i,j_0} tracks the boundary of S_i with increasing accuracy. In particular, the descendant nodes of S_{i,j_0} that are completely outside of S_i are discarded so that the outside hypervolume a_{i,j_0} gradually converges to zero. In the limit we then have, therefore, the same result of Eq. (A.9).

From a theoretical standpoint, our algorithm performs correct Poisson-disk sampling for an infinitely subdivided tree. In reality, we impose a maximum subdivision level of 24, which is deep enough to guarantee correct Poisson-disk distributions for all practical purposes. The probability (A.9) was obtained in the simple case where the arbitrary set s falls completely inside one of the leaf nodes in the tree. If that is not the case, the same result as (A.9) can be obtained except that the derivation is a bit more complex. One will have to consider the disjoint union of all fragments of s that fall in different leaf nodes.

B. INTERSECTION TESTING BETWEEN A HYPERSPHERE AND A HYPERCUBE

Arvo [1990] presented three Boolean methods for testing the intersection between a hypersphere and a hypercube in n dimensions, depending on whether the hypersphere and the hypercube should be treated as surfaces or as solidss. Recently, Larsson et al. [2007] improved on the solid-solid intersection method of Arvo [1990] by providing early exits from the routine as soon as a decision about the intersection state can be made. Rather than a Boolean test, we need an intersection test for our Poisson-disk sampling method that returns one of three possible outcomes for the state of the hypercube relative to the hypersphere:

`In`: The hypercube is entirely contained inside the hypersphere.

`Out`: The hypercube is located entirely on the outside of the hypersphere.

`Over`: The hypercube either intersects with or contains the hypersphere.

A hypersphere in $\mathbb{R}^n$ is defined by a center $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ and a radius $2r$. A hypercube is defined by the Cartesian product of the intervals $T = [t_{1_{\text{MIN}}}, t_{1_{\text{MAX}}}] \times [t_{2_{\text{MIN}}}, t_{2_{\text{MAX}}}] \times \cdots \times [t_{n_{\text{MIN}}}, t_{n_{\text{MAX}}}]$. The intersection status between the two objects can be determined by computing the minimum and maximum distances from $\mathbf{c}$ to all points $\mathbf{x} \in T$. We have

$$d_{\text{MIN}}(\mathbf{c}) \triangleq \min_{\mathbf{x} \in T} (d(\mathbf{x}, \mathbf{c})) \tag{B.1}$$

$$d_{\text{MAX}}(\mathbf{c}) \triangleq \max_{\mathbf{x} \in T} (d(\mathbf{x}, \mathbf{c})), \tag{B.2}$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ is the standard Euclidean distance in $\mathbb{R}^n$ between points $\mathbf{x}_1$ and $\mathbf{x}_2$. The minimum and maximum distances are

```
let $d^2_{\text{MIN}} = 0$;
let $d^2_{\text{MAX}} = 0$;
for $i = 1, 2, \ldots, n$
  let $d_{i_{\text{MIN}}} = t_{i_{\text{MIN}}} - c_i$;
  if $d_{i_{\text{MIN}}} > 0$
    if $d_{i_{\text{MIN}}} > 2r$ return Out;
    let $d^2_{\text{MIN}}$ += $d^2_{i_{\text{MIN}}}$;
    if $d^2_{\text{MIN}} > 4r^2$ return Out;
    let $d^2_{\text{MAX}}$ += $(t_{i_{\text{MAX}}} - c_i)^2$;
    continue
  let $d_{i_{\text{MIN}}} = c_i - t_{i_{\text{MAX}}}$;
  if $d_{i_{\text{MIN}}} > 0$
    if $d_{i_{\text{MIN}}} > 2r$ return Out;
    let $d^2_{\text{MIN}}$ += $d^2_{i_{\text{MIN}}}$;
    if $d^2_{\text{MIN}} > 4r^2$ return Out;
    let $d^2_{\text{MAX}}$ += $(c_i - t_{i_{\text{MIN}}})^2$;
    continue
  let $d^2_{\text{MAX}}$ += $\max^2(c_i - t_{i_{\text{MIN}}}, t_{i_{\text{MAX}}} - c_i)$;
if $d^2_{\text{MAX}} > 4r^2$ return Over else return In;
```

Fig. 19. Pseudocode for the intersection test between a hypercube and a hypersphere in *n* dimensions.

computed with

$$d_{\text{MIN}}(\mathbf{c}) = \sqrt{\sum_{i=1}^{n} d^2_{i_{\text{MIN}}}(c_i)} \qquad \text{(B.3)}$$

$$d_{\text{MAX}}(\mathbf{c}) = \sqrt{\sum_{i=1}^{n} d^2_{i_{\text{MAX}}}(c_i)}, \qquad \text{(B.4)}$$

where $d_{i_{\text{MIN}}}(c_i) = \min_{x \in [t_{i_{\text{MIN}}}, t_{i_{\text{MAX}}}]}(x - c_i)$ and similarly for $d_{i_{\text{MAX}}}(c_i)$. The outcome of the intersection test is determined from the distances $d_{\text{MIN}}(\mathbf{c})$ and $d_{\text{MAX}}(\mathbf{c})$ according to the following inequalities.

`In`:	$d_{\text{MAX}}(\mathbf{c}) < 2r$
`Out`:	$d_{\text{MIN}}(\mathbf{c}) > 2r$
`Over`:	$d_{\text{MIN}}(\mathbf{c}) \leqslant 2r \leqslant d_{\text{MAX}}(\mathbf{c})$

Following common practice, we compare the squares of the distances with the square $4r^2$ of the radius to avoid having to compute the square roots in Eq. (B.3). Figure 19 shows the pseudocode for our intersection test. We iterate over all the dimensions $i = 1, 2, \ldots n$ while accumulating the values of the minimum and maximum distances. Similar to Larsson et al. [2007], we also provide early exits whenever possible. Specifically, if it is found that $\sum_{j=1}^{i} d^2_{j_{\text{MIN}}}(c_j) > 4r^2$ for some $i < n$, then it is known that $d^2_{\text{MIN}}(\mathbf{c}) > 4r^2$ and a return code of `Out` can be issued without having to wait for the remaining iterations to complete. The same reasoning applies if $d_{j_{\text{MIN}}}(c_j) > 2r$ for any j. The variables $2r$ and $4r^2$ are static and can be initialized at the start of the algorithm.

The intersection test is where our Poisson-disk sampling algorithm spends most of its time and the efficiency of this test is critical to determine the efficiency of the whole algorithm. The pseudocode of Figure 19 has plenty of branching conditions to prevent floating point operations from being carried out unless they are strictly necessary. For processors that have SIMD instruction sets, it may be preferable instead to compute the $d_{i_{\text{MIN}}}(c_i)$ and $d_{i_{\text{MAX}}}(c_i)$ factors for several dimensions in parallel with the help of vectorized registers and avoiding the branching conditions. The reader is referred to Larsson et al. [2007] for a SIMD computation of $d_{\text{MIN}}(\mathbf{c})$ as part of their intersection test. With a little extra work, it is possible to do the same for the distance $d_{\text{MAX}}(\mathbf{c})$ and have a SIMD implementation of our intersection test. Current SIMD hardware allows for the computation of intersections in single precision up to dimension $n = 4$, which corresponds to the maximum dimension used in this article.

ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers whose suggestions helped to improve this article.

REFERENCES

ARVO, J. 1990. A simple method for box-sphere intersection testing. In *Graphics Gems*, A. S. Glassner, Ed. Academic Press Professional, San Diego, CA, 335–339.

BADDELEY, A. AND MØLLER, J. 1989. Nearest-Neighbour Markov point processes and random sets. *Int. Statist. Rev. 57*, 2, 89–121.

BARTLETT, M. S. 1974. The statistical analysis of spatial pattern. *Adv. App. Probab. 6*, 2, 336–358.

BRIDSON, R. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH'07 Sketches and Applications*. ACM Press, 22.

CLINE, D., JESCHKE, S., WHITE, K., RAZDAN, A., AND WONKA, P. 2009. Dart throwing on surfaces. *Comput. Graph. Forum 28*, 4, 1217–1226.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3, 287–294.

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1, 51–72.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MECH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modelling and rendering of plant ecosystems. In *Proceedings of the SIGGRAPH'98 Conference*, M. Cohen, Ed. Vol. 22. ACM Press, 275–286.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating points: A method for computing stipple drawings. *Comput. Graph. Forum 19*, 3, 40–51.

DICKMAN, R., WANG, J.-S., AND JENSEN, I. 1991. Random sequential adsorption: Series and virial expansions. *J. Chem. Phy. 94*, 12, 8252–8257.

DIGGLE, P. J., BESAG, J., AND GLEAVES, J. T. 1976. Statistical analysis of spatial point patterns by means of distance methods. *Biometrics 32*, 3, 659–667.

DIPPÉ, M. A. Z. AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. In *Proceedings of the SIGGRAPH'85 Conference*, B. A. Barsky, Ed. Vol. 19, 69–78.

DONEV, A., TORQUATO, S., AND STILLINGER, F. H. 2005. Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. I. Algorithmic details. *J. Comput. Phys. 202*, 2, 737–764.

DUNBAR, D. AND HUMPHREYS, G. 2006. A spatial data structure for fast Poisson-disk sample generation. *ACM Trans. Graph. 25*, 3, 503–508.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination*, 2nd Ed. AK Peters Ltd, Wellesley, MA.

FENG, L., HOTZ, I., HAMMAN, B., AND JOY, K. I. 2008. Anisotropic noise samples. *IEEE Trans. Visualiz. Comput. Graph. 14*, 2, 342–354.

FU, Y. AND ZHOU, B. 2008. Direct sampling on surfaces for high quality remeshing. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*, E. Haines and M. McGuire, Eds. ACM Press, 115–124.

GLASSNER, A. S. 1995. *Principles of Digital Image Synthesis*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann Publishers, San Francisco, CA.

HACHISUKA, T., JAROSZ, W., WEISTROFFER, R. P., DALE, K., HUMPHREYS, G., ZWICKER, M., AND JENSEN, H. W. 2008. Multi-Dimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3, 33:1–33:10.

HILLER, S., DEUSSEN, O., AND KELLER, A. 2001. Tiled blue noise samples. In *Vision, Modeling and Visualization 2001*, B. Girod, G. Greiner, H. Niemann, and H.-P. Seidel, Eds. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 256–272.

JAEGER, H. M. AND NAGEL, S. R. 1992. Physics of the granular state. *Sci. 255*, 5051, 1523–1531.

JONES, T. R. 2006. Efficient generation of Poisson-disk sampling patterns. *J. Graph. Tools 11*, 2, 27–36.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LICHINSKY, D. 2006. Recursive Wang tiles for real-time blue noise. *ACM Trans. Graph. 25*, 3, 509–518.

LAGAE, A. AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph. 24*, 4, 1442–1461.

LAGAE, A. AND DUTRÉ, P. 2006a. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Trans. Graph. 25*, 4, 1442–1459.

LAGAE, A. AND DUTRÉ, P. 2006b. Poisson sphere distributions. In *Vision, Modeling and Visualization 2006*, L. Kobbelt, T. Kuhlen, T. Aach, and R. Westermann, Eds. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 373–379.

LAGAE, A. AND DUTRÉ, P. 2008. A comparison of methods for generating Poisson disk distributions. *Comput. Graph. Forum 27*, 1, 114–129.

LARSSON, T., AKENINE-MÖLLER, T., AND LENGYEL, E. 2007. On faster sphere-box overlap testing. *J. Graph. Tools 12*, 1, 3–8.

LEHTINEN, J., ZWICKER, M., TURQUIN, E., KONTKANEN, J., DURAND, F., SILLION, F. X., AND AILA, T. 2008. A meshless hierarchical representation for light transport. *ACM Trans. Graph. 27*, 3, 37.

LENEMAN, O. A. Z. 1966. Random sampling of random processes: Impulse processes. *Inf. Control 9*, 4, 347–363.

LEWIS, J.-P. 1989. Algorithms for solid noise synthesis. In *Proceedings of the SIGGRAPH'89 Conference*, J. Lane, Ed. Vol. 23. ACM Press, 263–270.

LI, H., LO, K.-Y., LEUNG, M.-K., AND FU, C.-W. 2008. Dual Poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE Trans. Visualiz. Comput. Graph. 14*, 5, 982–998.

LLOYD, S. P. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory 28*, 2, 129–137.

LUBACHEVSKY, B. D. AND STILLINGER, F. H. 1990. Geometric properties of random disk packings. *J. Statist. Phys. 60*, 5-6, 561–583.

LUBACHEVSKY, B. D., STILLINGER, F. H., AND PINSON, E. N. 1991. Disks vs. spheres: Contrasting properties of random packings. *J. Statist. Phys. 64*, 3-4, 501–524.

MATÉRN, B. 1960. Spatial variation. *Meddelanden från Statens Skogsforskningsinstitut 49*, 1–144.

MCCOOL, M. AND FIUME, E. 1992. Hierarchical Poisson disk sampling distributions. In *Proceedings of Graphics Interface'92*. Canadian Information Processing Society, 94–105.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *Proceedings of the SIGGRAPH'87 Conference*, M. C. Stone, Ed. Annual Conference Series, vol. 21. ACM Press, 65–72.

MITCHELL, D. P. 1991. Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the SIGGRAPH'91 Conference*, T. W. Sederberg, Ed. Vol. 25. ACM Press, 157–164.

OSTROMOUKHOV, V. 2007a. Building 2D low-discrepancy sequences for hierarchical importance sampling using dodecagonal aperiodic tiling. In *Proceedings of GraphiCon'07*. 139–142.

OSTROMOUKHOV, V. 2007b. Sampling with polyominoes. *ACM Trans. Graph. 26*, 3, 78.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23*, 3, 488–495.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing* 2nd Ed. Cambridge University Press.

RIPLEY, B. D. 1977. Modelling spatial patterns. *J. Royal Statist. Soc. Series B 39*, 172–212.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA.

SECORD, A., HEIDRICH, W., AND STREIT, L. 2002. Fast primitive distribution for illustration. In *Proceedings of the 13th Eurographics Workshop on Rendering Techniques*, S. Gibson and P. Debevec, Eds. Eurographics Association, 215–226.

SHIRLEY, P. 1992. Nonuniform random point sets via warping. In *Graphics Gems III*, D. Kirk, Ed. Academic Press, San Diego, CA, 80–83.

SKOGE, M., DONEV, A., STILLINGER, F. H., AND TORQUATO, S. 2006. Packing hyperspheres in high-dimensional Euclidean spaces. *Phys. Rev. E 74*, 4, 041127.

SNYDER, D. L. 1991. *Random Point Processes in Time and Space*, 2nd Ed. Springer-Verlag, Berlin.

ULICHNEY, R. A. 1988. Dithering with blue noise. *Proc. IEEE 76*, 1, 56–79.

WEI, L.-Y. 2008. Parallel poisson disk sampling. *ACM Trans. Graphi. 27*, 3, 20.

WEISSTEIN, E. W. Hypersphere packing. From MathWorld—A Wolfram Web Resource.

WHITE, K. B., CLINE, D., AND EGBERT, P. K. 2007. Poisson disk point sets by hierarchical dart throwing. In *Proceedings of the IEEE/Eurographics Symposium on Interactive Ray Tracing*, A. Keller and P. Christensen, Eds. IEEE Press, 129–132.

WORLEY, S. P. 1996. A cellular texture basis function. In *Proceedings of the SIGGRAPH'96 Conference*, H. Rushmeier, Ed. Vol. 30. ACM Press, 291–294.

YELLOT, JR, J. I. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Sci. 221*, 382–395.

Received August 2007; revised August 2009; accepted September 2009