Fast Poisson-Disc Sample Generation in n-Dimensional Space by Subdivision Refinement

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

We present a fast method to generate samples based on a Poisson-disc distribution. This type of distribution, because of its blue noise spectral properties, is useful for image sampling in Computer Graphics. It is also useful for multidimensional Monte Carlo integration and as part of a procedural object placement function in both 2D and 3D settings. Our method extends trivially from 2D to 3D or any higher dimensional space over which one wishes to generate samples. The efficiency of the method derives from the use of a spatial subdivision data structure that signals the regions of space where the insertion of new samples is allowed. This data structure is a quadtree in 2D, an octree in 3D, and their equivalents in higher dimensions. The method has $O(N \log N)$ time and space complexity relative to the total number of samples. The method also generates dense distributions in the sense that no further samples can be inserted at the completion of the algorithm. We show that our method is faster than the most recent methods for accurate Poisson-disc sampling in two dimensions (being only surpassed by approximate sampling methods) and that, unlike those recent methods, it can also be easily extended to three-dimensional graphics problems.

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: Poisson-disc sampling, spatial subdivision

1. INTRODUCTION

Poisson-disc sampling is a process that distributes random samples on a domain of n-dimensional space based on two pre-defined criteria. Specifically, for a sampling process on a domain $D \subset \mathbb{R}^n$:

(1) Each sample must be placed randomly in D according to a uniform density distribution.
(2) The distance between every pair of samples must not be smaller than some chosen distance r.

In this paper, we propose a fast Poisson-disc sampling method that generates samples on the domain $D = [0,1]^n$, consisting of the 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 two previous criteria can be expressed mathematically as follows:

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

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

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

The first author is supported by grant SFRH/BD/16249/2004 from the Fundação para a Ciência e a Tecnologia, Portugal.
Author's address: Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK; email: {m.gamito,s.maddock}@dcs.shef.ac.uk.

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. The parameter r is the *distribution radius*.

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 favoured 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 the distribution radius.

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

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

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 r 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 dense distribution is also said to be *jammed*. Dense Poisson-disc distributions are desirable because they maximise the number of samples generated while, at the same time, eliminating any space that remains unoccupied. Valid Poisson-disc distributions that are not dense may have large gaps inside of them devoid of samples. This makes sparse Poisson-disc distributions comparable to the simpler Poisson distributions in that there may be a noticeable clustering of samples.

The term Poisson-disc stems from the observation that samples in two dimensions are located at the centre of discs of radius $r/2$ so that no overlapping of discs occurs within the distribution. This is exemplified in Figure 1, which shows three dense Poisson-disc 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 centre of non-overlapping spheres of radius $r/2$, as Figure 2 exemplifies. For historical reasons, we keep the name Poisson-disc irrespective of the dimensionality of the space in which samples are being generated. The Poisson-disc sampling process provides a solution to the random sphere packing problem, which attempts to place non-overlapping spheres in three dimensions whilst minimising the volume of unoccupied space. In the fields of Chemistry and Statistical Physics, the Poisson-disc process is also known as the Hard-Core process and is one example of Random Sequential Adsorption [Baddeley and Møller 1989; Dickman et al. 1991].

In this paper, we present a fast method to generate large numbers of samples in a n-dimensional space as the outcome of a dense Poisson-disc sampling process. Our method uses a subdivision refinement tree in n dimensions to help in the placement of samples. We begin in Section 2 by discussing the applications of Poisson-disc sampling in computer graphics. Section 3 then presents current methods for Poisson-disc sample generation. Our method is explained in Section 4. A comparison of our method with the methods from Section 3 is done in Section 5. An Appendix details the routine for testing the intersection between a sample, which is surrounded by a hypersphere of radius r, and a node in the tree, represented by a hypercube.

2. POISSON-DISC SAMPLING FOR COMPUTER GRAPHICS

Poisson-disc sampling was introduced to computer graphics by Dippé and Wold [1985], who proposed it as a sampling technique for image anti-aliasing. The reason why Poisson-disc sampling is ideal for anti-aliasing

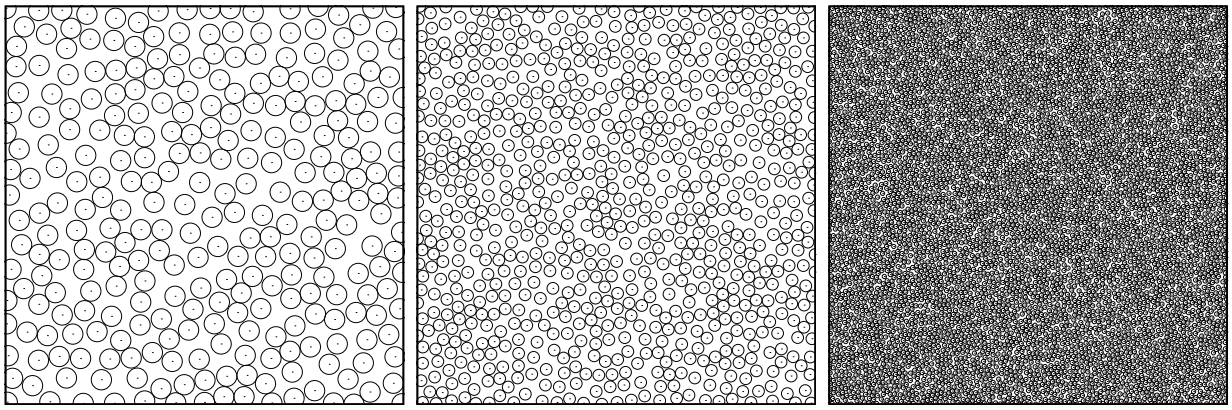

Fig. 1. Two-dimensional Poisson-disc sampling on the unit square. From left to right, the distribution radii are 0.05, 0.03 and 0.01. The number of samples is 288, 784 and 6 945, respectively. The time taken to generate the samples with our algorithm was 0.036s, 0.046s and 0.280s, respectively.

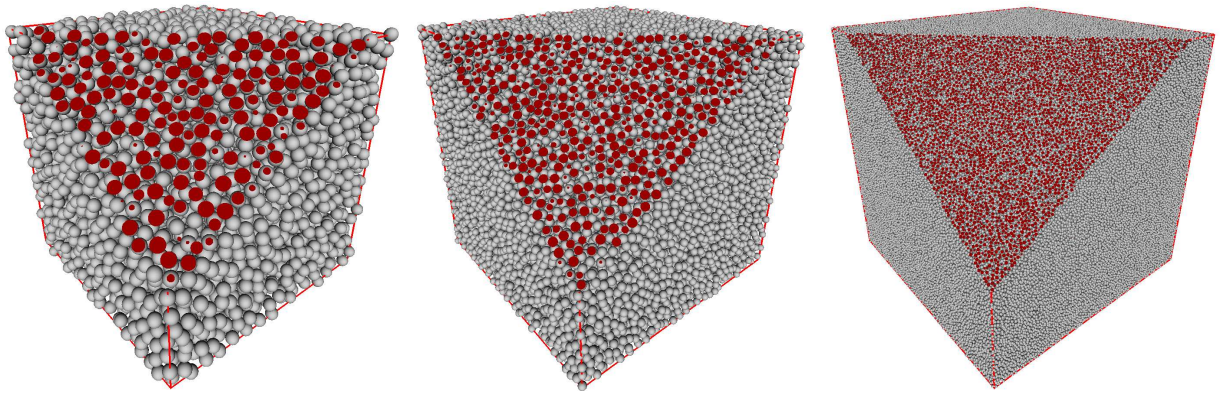

Fig. 2. Three-dimensional Poisson-disc sampling on the unit cube. From left to right, the distribution radii are 0.05, 0.03 and 0.01. The number of samples is 6 360, 28 416 and 736 682, respectively. The time taken to generate the samples with our algorithm was 1.539s, 8.172s and 326.300s, respectively.

is because of its blue noise spectral properties. A blue noise spectrum is characterised 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-disc process. Figure 3 shows this power spectrum. The frequency response of a typical lowpass anti-aliasing filter is also shown in Figure 3. The gap in the Poisson-disc spectrum for low frequencies (with a width that varies with the inverse of the radius) allows the anti-aliasing filter to recover the original signal whose spectrum is centred around the DC spike. Copies of the spectrum from the original signal are also spread along the high frequencies by the Poisson-disc 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 uniform 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-disc distribution for the areas outside the fovea. Such a result helps

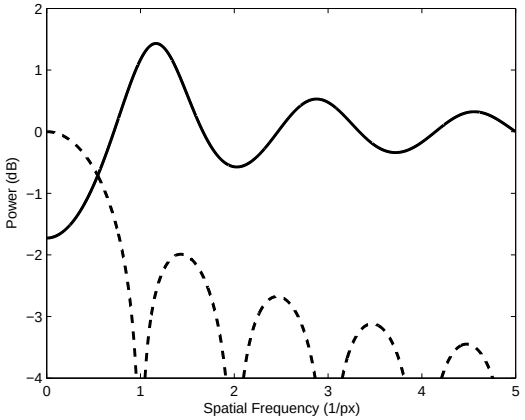

Fig. 3. The average power spectrum of a one-dimensional Poisson-disc process corresponds to a blue noise spectrum. The dotted line shows the frequency response of a typical low pass filter.

to explain why visual perception is so forgiving of low power noise artifacts and, therefore, why Poisson-disc sampling is useful for image sampling and anti-aliasing. The simpler Poisson sampling, in contrast, presents a flat spectrum across all frequencies and is not recommended for image sampling since it cannot separate the low frequencies of the signal from the high frequencies of the aliases.

Cook [1986] suggested that Poisson-disc sampling could be used as part of a distributed ray tracing algorithm, not only to perform anti-aliasing but also to generate such effects as motion blur, depth of field, smooth shadows, glossy reflection and transparency. Unfortunately, methods for generating Poisson-disc distributions were inefficient at the time and Cook relied instead on *jittered sampling* as a cheaper 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-disc sampling, with a low frequency gap that is smaller and less well resolved.

The applications of Poisson-disc sampling to ray tracing 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-disc distribution in a canonical hypercube $[0, 1]^n$, for a 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 non-uniform distribution leads to visually better results when compared with the domain warping method. A non-uniform Poisson-disc 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 $r(\mathbf{x})$. Ostromoukhov et al. demonstrated the application of non-uniform distributions to perform importance sampling of environment maps while Kopf et al. used them to perform image dithering in real time.

Graphical objects can be distributed in space based on a Poisson-disc sampling. This has been used to distribute ink strokes for non-photorealistic rendering [Deussen et al. 2000; Secord et al. 2002]. 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

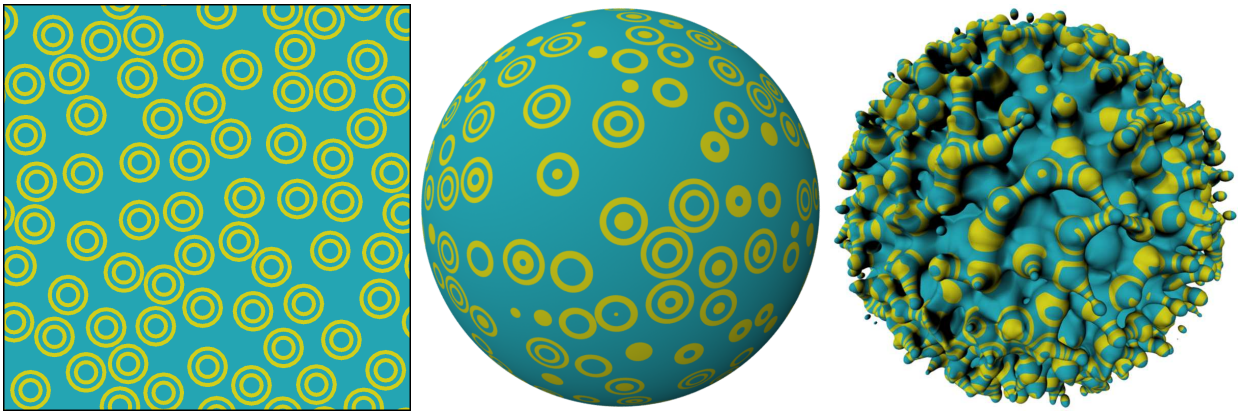

Fig. 4. A procedural object distribution function used to generate a two-dimensional texture (left), a solid texture (middle) and both a solid texture and a hypertexture (right).

object can be bounded by a disc or a sphere of radius $r/2$ then a Poisson-disc distribution with radius r will guarantee that no interference exists between the copies. Lagae and Dutré have proposed a procedural object placement function by creating an infinite aperiodic distribution of Poisson-disc samples on the plane [Lagae and Dutré 2005; 2006a]. The same authors later extended their procedural object placement function to three dimensions [Lagae and Dutré 2006b]. Figure 4 shows simple examples of procedural placement of a radially symmetric primitive. More complex examples can include pseudo-random rotations and scalings that are applied when instancing every copy of a primitive. Poisson-disc 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-DISC SAMPLING

We present here an introduction to the methods that have been developed to generate Poisson-disc distributions. The reader is referred to Lagae and Dutré [2007] for a more in-depth survey of these methods. We make a distinction between *accurate methods*, *approximate methods* and *tile based methods* for Poisson-disc sampling. Accurate methods obey conditions (1a) and (1b), generating correct Poisson-disc 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-disc 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 pre-computation 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 quasi-periodic tilings because each tile is copied repeatedly across space. If the sampling space is displaced, several copies of the tiles overlap. This quasi-periodicity 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-disc sampling methods are the fastest because all samples have already been generated at run time. 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-disc sampling conditions. Accurate methods are the slowest since they

are required to enforce exactly both conditions for Poisson-disc sampling. Despite the existence of many fast approximate and tile based sampling algorithms, the need for algorithms that can generate Poisson-disc 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 for Poisson-disc 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 sampling mechanism by trial and error – a large number of samples is attempted but only a small percentage of them is inserted into the distribution with the others being rejected. The algorithm cannot guarantee that a dense 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-disc sampling. Its inefficiency lead to the development of approximate sampling algorithms. The situation changed recently with the development of two 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. These spatially guided methods were developed specifically for two-dimensional sample distributions and do not extend well to higher dimensions. Both methods have $O(N \log N)$ time complexity for N samples as their data structures are organised in the shape of a binary tree.

One 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 r with the original sample at the centre. 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 used as weights. This ensures that sample placement is done with a uniform probability distribution – the tree is randomly traversed top to bottom using the area weights to control 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 r, which can be reduced to four fundamental cases. A sampling method by trial and error 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 r. This is a more complex procedure than its two-dimensional equivalent and cannot be reduced to a small number of simple cases.

The other 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 centred at distinct points. The available area around each sample can be represented as the 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 trial and error sampling strategy is avoided as sampling inside a scalloped sector is always guaranteed to generate a valid Poisson-disc sample. It is not known how the scalloped sector data structure can be extended to three dimensions.

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-disc 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 that they generate embedded streams of Poisson-disc samples. If the sample sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}$ is approximately Poisson-disc with radius r_i, the larger sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_{i+1}\}$ is also approximately Poisson-disc with radius $r_{i+1} < r_i$, up to a maximum sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. These embedded streams are attractive because they allow several Poisson-disc distributions with different radii to be generated from a single run of the sampling algorithm.

The best candidate algorithm works by trying mi samples when placing the i-th 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 unfavourable 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-disc 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 r and r_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 r and r_i, respectively, not considering the presence of other nearby samples that may reduce this allowable area.

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

3.3 Tile Based Methods

The first tile based Poisson-disc sampling methods used *Wang tiles* and were proposed by Hiller et al. [2001] and Cohen et al. [2003]. Wang tiles have colours assigned to their edges in specific ways. A Wang tile can only be placed next to another if they share the same colour along the common edge. This allows aperiodic tilings of the plane to be created. The generation of Poisson-disc 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 colour. 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

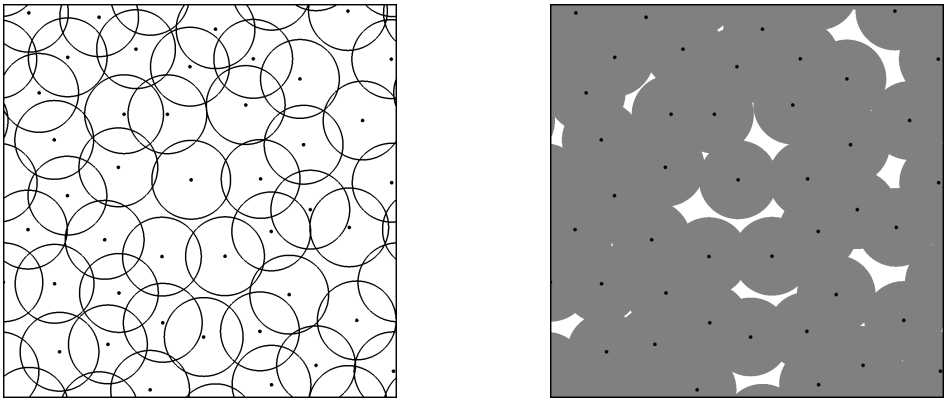

Fig. 5. The left image shows an incomplete Poisson-disc distribution visualised with discs of radius r. The right image shows the same distribution with gray shaded discs. New samples can be inserted in the white areas.

fly in a consistent way to any arbitrary point in space. This leads to the creation of infinite aperiodic tilings of Poisson-disc samples. Lagae and Dutré [2006a] later introduced procedural tiling rules for *corner tiles*. Corner tiles have colours 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 non-uniform Poisson-disc sampling have been proposed based on tile distributions. Kopf et al. [2006] apply subdivision rules to Wang tiles, where tiles can have smaller tiles inside of them, 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 2007]. 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.

4. POISSON-DISC SAMPLING BY SUBDIVISION REFINEMENT

Our method for Poisson-disc sampling performs a subdivision refinement of the allowable space for the insertion of new samples. It can be used in all applications of Poisson-disc sampling that were discussed in Section 2 with the exception of the direct generation of nonuniform distributions. The method only requires the specification of the distribution radius $0 < r \leqslant \sqrt{n}$ as a starting parameter. Given that samples are generated inside the unit hypercube $[0,1]^n$, values $r > \sqrt{n}$ 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. Like with all other Poisson-disc 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 discs of radius r instead of the $r/2$ half-discs that were shown in Figure 1. The larger discs, shown on the left of Figure 5, are now intersecting but a property still holds that no disc contains any sample other than the sample at its centre. The same discs are gray shaded on the right of Figure 5. The white areas in this image represent the empty space where new samples can be inserted. Once the unit square becomes uniformly coloured in gray, the distribution is jammed.

A spatial subdivision data structure is used to mark the allowable insertion space. In two dimensions, this

```
initialise tree with hypercube [0, 1]^n;
while tree not empty
  Select node from tree;
  if node is a candidate
    generate sample inside node;
    output sample;
    Update tree with sample;
```

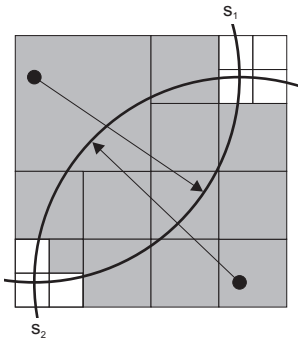

Fig. 6. The main algorithm for Poisson-disc sample generation. The diagram on the right shows an example. The gray shaded nodes have already been pruned from the tree. The top right node is a candidate node.

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 main algorithm is shown in pseudo-code form in Figure 6. The routine `Select` and `Update`, invoked by the main algorithm, will be further explained in the following sections. The algorithm is initialised by placing a root node with dimensions $[0, 1] \times [0, 1]$ in the tree at subdivision level 0. For purposes of sample insertion, 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 discs of any previously inserted samples, otherwise it is a potential candidate. Figure 6 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 disc of one of the samples and which can be discarded. All the remaining leaf nodes in the tree are potential candidates. Samples can only be inserted in candidate nodes as these are the only ones where the random placement of a sample is guaranteed not to violate the minimum distance constraint relative to all other previous samples.

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 two different events can occur based on whether the leaf is a candidate or a potential candidate. If the leaf is a candidate, a new sample is randomly inserted inside with a uniform probability density and the tree is updated to account for the presence of the new sample. If, on the other hand, the selected node is a potential candidate no sample insertion takes place. The `Select` routine internally subdivides the tree whenever it reaches a potential candidate leaf node. Subdivision can proceed while the areas of the nodes do not change. The `Select` routine then returns once a candidate node is reached through subdivision. The subdivision must stop, however, once a node is reached that is to be pruned from the tree. Pruning implies a change in the total area represented by the tree and requires that a new tree traversal be executed in a subsequent iteration. The proposed algorithm implicitly assumes a sampling method by trial and error, similar to the dart-throwing algorithm. The difference is that in dart-throwing the rejection of a sample represents a wasted computational effort. In our case, even during iterations where samples are not inserted, the subdivision tree is always refined and the algorithm continually converges towards a dense distribution.

A maximum subdivision level condition is important to prevent the algorithm from becoming locked in an infinite loop in situations where three or more discs are intersecting at the same point. Figure 7 illustrates this scenario. Although the node shown in this figure is covered by the discs, none of the three discs by themselves provide a complete coverage. The node is deemed to be subdivided and the same problem is

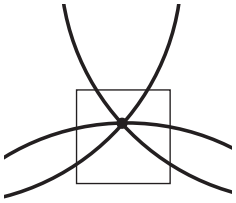

Fig. 7. Three or more discs intersecting at a common point cause a situation where a node is subdivided indefinitely. A maximum subdivision level must be applied to force the algorithm to terminate.

going to occur for all the descendants that contain the point of intersection of the discs. The consequences of enforcing a maximum level of subdivision can be formalised by stating that the algorithm generates distributions obeying the following condition:

$$\exists \varepsilon > 0, \forall \mathbf{x} \in D, \exists \mathbf{x}_i \in X : \ \|\mathbf{x} - \mathbf{x}_i\| < r + \varepsilon. \tag{3}$$

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

4.1 Selecting Nodes from the Tree

The selection of candidate leaf nodes from the tree for the purpose of sample insertion is shown in pseudo-code in Figure 8. The routine `Select` is recursive and accepts as arguments a current node in the tree, a reference variable *selnode*, where the selected node 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 *selnode* is initialised with a null node.

The `Select` routine begins by checking if the current node is a leaf. If it is and if it is also a candidate node then it is immediately selected. No pruning occurs so $\delta = 0$ is returned. If the maximum subdivision level has been reached then the leaf node is to be discarded and the pruned area is $\delta = 2^{-nl_{MAX}}$. Otherwise, the node is subdivided and a return is made if some of the child nodes were pruned. Pruning of child nodes invalidates the areas of the nodes that have been used while traversing the tree so far and no further traversal is possible, through recursive invocation of `Select`, during the current iteration of the main algorithm.

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 either when the current node is not a leaf or when it is a leaf that has been subdivided but for which no pruning occurred. The `Select` routine is then recursively invoked on the chosen child node. The remaining part 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 `Select` 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 `Select`, 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.

```
Select(node,selnode,l)
if node is a leaf
  if node is a candidate
    let selnode = node;
    return 0;
  if l = l_MAX return 2^-nl;
  let δ = Subdivide(node);
  if δ > 0 return δ;
let child of node be randomly chosen;
let δ = Select(child,selnode,l + 1);
if δ < area of child
  decrement area of child by δ;
else
  discard child from the tree;
return δ;
```

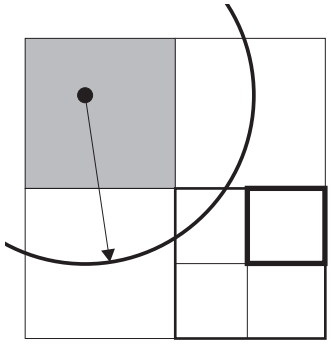

Fig. 8. The `Select` algorithm. The diagram on the right shows an example. Initially, there are three partial candidates and one pruned node. The node on the lower right, shown with medium thickness, is randomly selected with a $3:1$ probability. Because it is a partial candidate it is then subdivided. The child shown with maximum thickness is randomly selected with a $4:1$ probability and is accepted as it is a candidate.

```
Subdivide(node)
let δ = 0;
generate children of node;
for each child
  for each sample in node's list of samples
    if child is intersected by sample's disc
      add sample to child's list of samples;
    else if child is contained in sample's disc
      discard child;
      increment δ 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 δ;
```

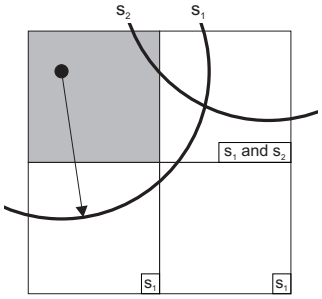

Fig. 9. The `Subdivide` algorithm. The diagram on the right shows an example for a parent node that intersects samples s_1 and s_2. The gray shaded node is discarded. The small boxes for each leaf node indicate which samples intersect that node.

4.2 Subdividing Nodes in the Tree

The pseudo-code for the `Subdivide` routine is shown in Figure 9. This routine is invoked from within `Select` 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 node of a subdivided node must be compared against the samples that have already been inserted in the distribution. This process is made efficient by storing in each node a list of the samples whose discs intersect with it. 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 disc 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 disc completely contains

```
Update(node,sample)
if node is outside sample's disc
  return 0;
if node is inside sample's disc
  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 δ_T = 0;
forall children of node
  let δ = Update(child,sample);
  if δ < area of child
    decrement area of child by δ;
  else
    discard child from the tree;
  increment δ_T by δ;
return δ_T;
```

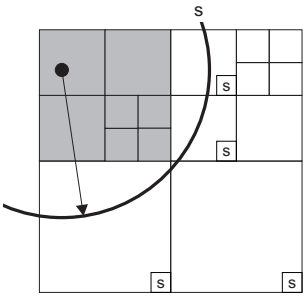

Fig. 10. The `Update` algorithm. The diagram on the right shows an example for the insertion of a new sample s. The gray shaded nodes are pruned from the tree. Leaf nodes that intersect with the new sample are marked with a small box.

the child node, that child node can be discarded and the value of δ incremented correspondingly. The list of samples for each node also indicates the candidate status of the node. Nodes with non-empty lists of samples are potential candidates, otherwise they are candidates.

The tests for intersection or containment between a node and a disc 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 the Appendix. 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 disc 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.

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. It accepts a current node in the tree as an argument, together with the sample that has just been inserted. Similar to the `Select` routine, `Update` is first invoked for the node at the top of the tree and, for each invocation, returns the area of the nodes that have been pruned underneath the current node. The routine first checks the node against the disc of the sample. If the node is completely outside the disc, an early return is made and no pruning occurs. If the node is completely inside the disc 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 intersects the disc of the new sample. 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 `Select` 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_T and returned to higher nodes

of the tree.

4.4 Time Complexity

The procedures `Subdivide`, `Select` 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. Selecting a node from the tree and updating the tree with a new sample take logarithmic time in order to traverse the tree all the way to the leaves. These two logarithmic times are dominant in the algorithm since node subdivision is invoked from within the node selection procedure. As a consequence, for a distribution with N samples, the total time complexity for inserting a new sample is $O(N \log N)$.

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 copies of the hypercube are placed side by side to create an infinite periodic tiling that is still a valid Poisson-disc 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 non-periodic distributions. In the original non-periodic 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 $D = \{-1, 0, +1\}$, it is possible to write the following by incurring a slight abuse of notation:

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

This pseudo-code 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 after the insertion of every new sample. Many of these invocations terminate early, however, in the cases where the disc of the offset sample $s + d$ does not intersect with the unit hypercube at the root of the subdivision tree.

4.6 Memory Management

One drawback of the proposed Poisson-disc sampling method is the memory occupancy of the required subdivision tree, which implies $O(N \log N)$ space complexity. This problem becomes significant for higher dimensions and also for smaller values of the radius r, when the tree needs to be refined down to a large number of subdivision levels. For a sufficiently small value of r, the algorithm stops working once the available memory is exhausted[1]. To overcome this problem we divide the unit hypercube into a tiling of m^n smaller hypercubes and generate Poisson-disc distributions for each smaller hypercube at a time. We enforce a minimum ratio $\rho = r/(1/m) = rm$ between r and the lateral size $1/m$ of a tile. Table I shows appropriate values of ρ that were found by testing the algorithm on machines with 1, 2 and 4 Gigabytes of main memory. These values may have to be increased, however, if the algorithm is to be run as part of a larger application.

[1]The method by Jones [2006] also has $O(N \log N)$ space complexity and suffers from the same problem. The methods by Dunbar and Humphreys [2006] have $O(N)$ space complexity and are better equipped to handle distributions with small radii.

Table I. The minimum ratio ρ between distribution radius and lateral tile size in two, three and four dimensions.

	Memory		
n	1Gb	2Gb	4Gb
2	0.05%	0.032%	0.024%
3	1.2%	0.86%	0.71%
4	6.2%	5.1%	4.6%

For a given value of r, supplied as a parameter to the algorithm, the number of tile divisions along each of the coordinate axes is given by $m = \lceil \rho/r \rceil$, using the values for ρ from Table I.

Tiles in the unit hypercube are selected one at a time in raster scan order for Poisson-disc generation. The Poisson-disc properties must be enforced across the tiles in order to avoid any noticeable artifacts at the tile boundaries. Rather than computing Poisson-disc distributions inside hypercubes with volume $(1/m)^n$, distributions are computed inside larger hypercubes with volume $(1/m + 2r)^n$ as shown in Figure 11. This corresponds to an offset of the hypercube for each tile by a distance of r and allows sharing of information between neighbouring tiles. The subdivision tree used in the main algorithm of Figure 6 is initialised with the offset hypercube. All the Poisson-disc samples that may already exist in the neighbouring tiles and whose discs either intersect with or are contained in the offset hypercube are used as starting samples for the new distribution. The samples are passed along to the algorithm as part of the tree's root node, which at this stage is also a leaf node and, therefore, possesses a list of samples. Once the distribution for a tile is complete, each sample that has been placed outside the tile, i.e. in the shaded area of Figure 11, is checked against the corresponding tile neighbour to which it belongs. A sample in the shaded area is discarded if it belongs to a neighbouring tile that has not been generated yet. This prevents samples that are influenced by internal tile boundary artifacts from being inserted into the distribution. In the opposite case, the sample is accepted since it fills up some empty space that remained when the neighbouring tile was generated and had some of its exterior samples discarded. This sharing of samples inside regions that are at a distance r from the tile boundaries enables the generation of a uniform Poisson-disc distribution from a sequence of spatially distinct runs of the Poisson-disc sampling algorithm.

There is some computational overhead involved in the generation of Poisson-disc distributions when the unit hypercube is tiled. Some samples are generated and later discarded in order to avoid the above mentioned boundary artifacts. The overhead is given by the ratio of discarded samples to accepted samples and is, in turn, related to the volume of the shaded region in Figure 11. This ratio is approximately given by $n(m-1)r$ for the entire unit hypercube if we ignore the corner areas of the shaded regions, such as the one in Figure 11. In a 3D setting, this approximation considers only the face adjacencies between tiles, which vary with $O(r)$, and ignores the edge and vertex adjacencies, which vary with $O(r^2)$ and $O(r^3)$, respectively. For small values of r, the n-dimensional "face" adjacencies will always dominate. Figure 12 shows a plot of the approximate computational overhead. It is bounded above by nr and has discontinuities at the transition points $r = \rho/k$ for $k \in \mathbb{N}$. Furthermore, $1/m \geqslant r$ must hold for the tiling mechanism to work, otherwise it would be necessary to query more than the immediate neighbours of a tile. This, in turn, means that $\rho \leqslant 100\%$ must hold. As can be seen in Table I, ρ increases with n. There is a maximum n, therefore, beyond which ρ will be larger than 100% and tiling will not be possible. This maximum number of dimensions must be found by experimentation and depends on the amount of main memory of the available hardware.

4.7 Specifying the Total Number of Samples

There are three different ways of supplying initial parameters to a Poisson-disc sampling algorithm. The basic parameters are the distribution radius r and the total number of samples N that is desired. One can

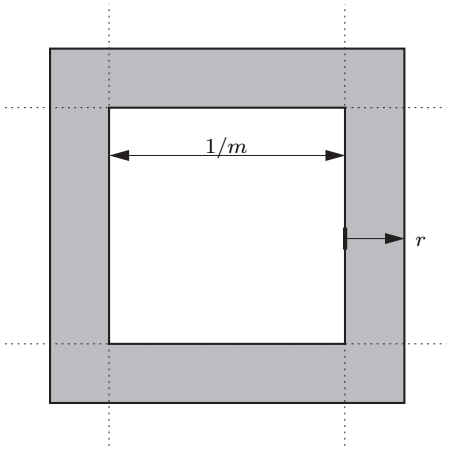

Fig. 11. A two-dimensional tile with a lateral size equal to $1/m$, part of a total of m^2 tiles. Samples are generated inside the square with a lateral size of $1/m + 2r$, offset by r from the tile.

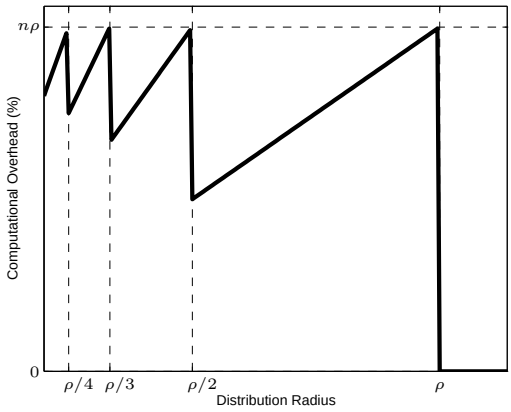

Fig. 12. The approximate computational overhead incurred by the hypercube tiling method in n dimensions. Transitions occur at the points ρ/k, with $k \in \mathbb{N}$, where the number of tiles m^n changes.

start a Poisson-disc sampling algorithm by specifiying r or N or both in conjunction. The specification of r is used when exact control over the distance between particles is required. The specification of N usually leads to sample distributions that are not dense as it is difficult to control the sampling process so that it becomes jammed exactly after the N-th sample has been inserted. The radius r is the starting parameter for the proposed sampling algorithm in order to generate dense 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 dense 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*. A distribution of N samples has a maximum packing ratio when the samples are placed deterministically according to a crystalline grid. In two dimensions, for example, this grid is hexagonal. The packing ratio $\delta_n \in (0, 1)$ is the maximum percentage of n-space that is occupied by the hyperspheres of radius $r/2$ centred around the samples. Packing ratios 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 $[0, 1]^n$, the maximum distribution radius is then:

$$r_{MAX} = 2 \sqrt[n]{\frac{\delta_n}{N} \frac{\Gamma\left(\frac{n}{2} + 1\right)}{\pi^{n/2}}}. \tag{4}$$

The previous equation takes into account the volume of a n-dimensional hypersphere with radius $r_{MAX}/2$, which is equal to $\pi^{n/2}(r_{MAX}/2)^n/\Gamma(n/2 + 1)$ where Γ is the gamma function. It is now possible to define a relative radius for the distribution as a parameter $\epsilon \in [0, 1]$ such that the absolute radius is:

$$r = \epsilon\, r_{MAX}. \tag{5}$$

When $\epsilon = 0$ is specified, no distance constraints are enforced and a Poisson distribution with N samples is generated. When $\epsilon = 1$ is specified, a deterministic placement of N samples is achieved, having the densest possible packing ratio. It must be remarked, however, that no random sample placement algorithm

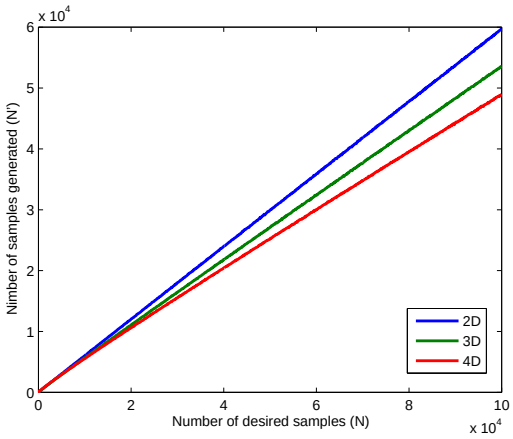

Fig. 13. The total number of samples generated N' against the desired number of samples N in two, three and four dimensions.

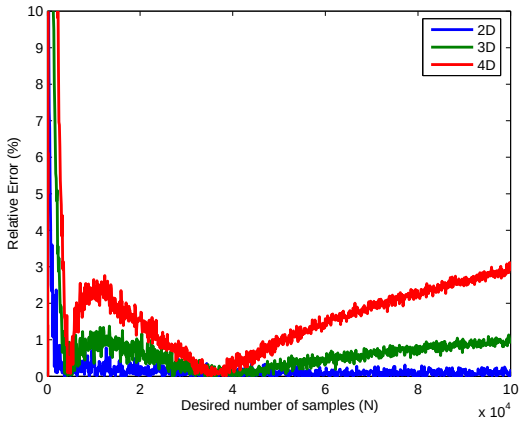

Fig. 14. The relative error between the number of samples generated and the desired number of samples N in two, three and four dimensions.

Table II. The parameters a and b in the linear regression $N = aN' + b$.

n	a	b
2	1.6758	-92.5539
3	1.8700	-510.4279
4	2.0508	-1260.7089

can attain this result. Poisson-disc sampling algorithms can only generate N samples exactly for relative radii that don't significantly exceed 0.8. For larger values of ϵ, the probability increases of the distribution becoming jammed before N samples can be inserted. The case $\epsilon = 1$, in particular, is always guaranteed to generate dense distributions with a number of samples that is smaller than N.

Figure 13 shows plots of the actual number of samples placed N' against the desired number of samples N in two, three and four dimensions for $\epsilon = 1$. Except for small deviations, the graphs are practically linear. A linear regression of the three curves leads to an inverse relation $N = aN' + b$ with the parameters shown in Table II. Since we are only interested in generating dense distributions, we can work with the case $\epsilon = 1$ and use the linear relations to manipulate the number of samples so that the actual number of samples generated is as close as possible to the desired number:

$$r = 2\sqrt[n]{\frac{\delta_n}{\max(aN+b, N)}\frac{\Gamma\left(\frac{n}{2}+1\right)}{\pi^{n/2}}}. \tag{6}$$

This equation gives a distribution radius where the number of samples has been increased internally to account for jamming. The maximum operator prevents the internal number of samples from becoming negative for small N where the linear regression results are less accurate. The resulting radius is then supplied as the starting parameter to the proposed Poisson-disc sampling algorithm.

Figure 14 shows the relative error $|N' - N|/N$ between the number of samples generated N' and the desired number of samples N after the corrected radius (6) has been used. Except for very small values of N, for which no correction is performed, the relative error stays below 5%. It is possible to further improve

Table III. Timing results in seconds for several Poisson-disc sampling algorithms in two dimensions.

	N		
Algorithm	1 000	10 000	100 000
Dunbar (accurate)	0.596s	5.588s	55.899s
Jones (accurate)	0.251s	2.715s	30.816s
Gamito (accurate)	0.055s	0.413s	5.148s
Dunbar (approximate)	0.006s	0.058s	0.578s

on these results by using the plots of Figure 13 as lookup tables to retrieve N for use in equation (4), given the number of samples N', so that the linearisation error is removed. For best results, the plots should be obtained by averaging many independent runs of the algorithm in order to reduce small random fluctuations.

5. PRELIMINARY RESULTS

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

—The Voronoi decomposition algorithm by Jones [2006].

—The accurate scalloped sector algorithm by Dunbar and Humphreys [2006].

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

Although the dart-throwing algorithm of Dippé and Wold [1985] is the basis for all spatially guided accurate Poisson-disc sampling algorithms, it cannot be used for comparison purposes because it does not have a guaranteed termination time when used to generate dense distributions. The source code for the algorithms used in the comparison has been made publicly available by their respective authors. The approximate sampling algorithm by Dunbar and Humphreys 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-disc sample generation are relaxed. For simplicity, we refer to the algorithms by the name of their respective first authors in the timing results shown as part of this section.

Table III shows the timing results of all algorithms for the computation of two-dimensional distributions with N samples. 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 our proposed algorithm, the technique described in Section 4.7 was used to generate distributions close to the desired number of samples, followed by extrapolation. The number of samples generated was 991, 10 033 and 99 909, respectively. The timings were obtained on a dual AMD Athlon MP2600 2.1GHz machine with 4Gb of main memory. The approximate sampling algorithm by Dunbar and Humphreys 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.

APPENDIX. Intersection Testing between a Hypersphere and a Hypercube

Arvo [1990] presented three boolean methods for testing the intersection between a hypersphere and a hyperbox in n dimensions, depending on whether the hypersphere and the hyperbox should be treated as surfaces or as volumes. Recently, Larsson et al. [2007] improved on the solid-solid intersection method of Arvo by providing early exits from the routine as soon as a decision about the intersection state can be

[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.

made. Rather than a boolean test, we need an intersection test for our Poisson-disc 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 entirely outside the hypersphere.

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

A hypersphere in $\mathbb{R}^n$ is defined by a centre $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ and a radius r. 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 $\mathbf{x} \in T$:

$$d_{\text{MIN}}(\mathbf{c}) \triangleq \min_{\mathbf{x} \in T} (d(\mathbf{x}, \mathbf{c})) \quad \text{and} \quad d_{\text{MAX}}(\mathbf{c}) \triangleq \max_{\mathbf{x} \in T} (d(\mathbf{x}, \mathbf{c})), \tag{A.7}$$

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 computed with:

$$d_{\text{MIN}}(\mathbf{c}) = \sqrt{\sum_{i=1}^{n} d_{i_{\text{MIN}}}^2(c_i)} \quad \text{and} \quad d_{\text{MAX}}(\mathbf{c}) = \sqrt{\sum_{i=1}^{n} d_{i_{\text{MAX}}}^2(c_i)}, \tag{A.8}$$

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 rules:

`In`: $d_{\text{MAX}}(\mathbf{c}) < r$.

`Out`: $d_{\text{MIN}}(\mathbf{c}) > r$.

`Over`: $d_{\text{MIN}}(\mathbf{c}) \leqslant r \leqslant d_{\text{MAX}}(\mathbf{c})$.

Following common practice, we compare the squares of the distances with the square r^2 of the radius to avoid having to compute the square roots in equation (A.8). Figure 15 shows the pseudo-code 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 $\sum_{j=1}^{i} d_{j_{\text{MIN}}}^2(c_j) > r^2$ for some $i < n$, then we know that $d_{\text{MIN}}^2(\mathbf{c}) > r^2$ and a return code of `Out` can be issued without having to wait for the remaining iterations to complete. The same reasoning applies if $d_{j_{\text{MIN}}}(c_j) > r$ for any j.

The intersection test is where our Poisson-disc sampling algorithm spends most of its time and the efficiency of this test is critical to determine the efficiency of the whole algorithm. The pseudo-code of Figure 15 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 vectorised 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.

REFERENCES

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

Baddeley, A. and Møller, J. 1989. Nearest-neighbour Markov point processes and random sets. *International Statistical Review 57,* 2 (Aug.), 89–121.

Cohen, M. F., Shade, J., Hiller, S., and Deussen, O. 2003. Wang tiles for image and texture generation. *ACM Transactions on Graphics (SIGGRAPH '03 Proceedings) 22,* 3 (July), 287–294.

```
let d²_MIN = 0;
let d²_MAX = 0;
for i = 1, 2, ..., n
  let d_{i_MIN} = t_{i_MIN} − c_i;
  if d_{i_MIN} > 0
    if d_{i_MIN} > r return Out;
    let d²_MIN += d²_{i_MIN};
    if d²_MIN > r² return Out;
    let d²_MAX += (t_{i_MAX} − c_i)²;
    continue
  let d_{i_MIN} = c_i − t_{i_MAX};
  if d_{i_MIN} > 0
    if d_{i_MIN} > r return Out;
    let d²_MIN += d²_{i_MIN};
    if d²_MIN > r² return Out;
    let d²_MAX += (c_i − t_{i_MIN})²;
    continue
  let d²_MAX += max² (c_i − t_{i_MIN}, t_{i_MAX} − c_i);
if d²_MAX > r² return Over else return In;
```

Fig. 15. The intersection test between a hypercube and a hypersphere.

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

Deussen, O., Hanrahan, P., Lintermann, B., Mech, R., Pharr, M., and Prusinkiewicz, P. 1998. Realistic modelling and rendering of plant ecosystems. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 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. *Computer Graphics Forum (Eurographics 2000 Proceedings) 19,* 3 (Aug.), 40–51.

Dickman, R., Wang, J.-S., and Jensen, I. 1991. Random sequential adsorption: Series and virial expansions. *Journal of Chemical Physics 94,* 12 (June), 8252–8257.

Dippé, M. A. Z. and Wold, E. H. 1985. Antialiasing through stochastic sampling. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, B. A. Barsky, Ed. Vol. 19. 69–78.

Dunbar, D. and Humphreys, G. 2006. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics (SIGGRAPH '06 Proceedings) 25,* 3 (July), 503–508.

Dutré, P., Bala, K., and Bekaert, P. 2006. *Advanced Global Illumination*, 2nd ed. AK Peters Ltd, Wellesley, MA.

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

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.

Jones, T. R. 2006. Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics 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 Transactions on Graphics (SIGGRAPH '06 Proceedings) 25,* 3, 509–518.

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

Lagae, A. and Dutré, P. 2006a. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Transactions on Graphics 25,* 4 (Oct.), 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. 2007. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27,* 1 (Mar.), 114–129.

Larsson, T., Akenine-Möller, T., and Lengyel, E. 2007. On faster sphere-box overlap testing. *Journal of Graphics Tools 12,* 1, 3–8.

Leneman, O. A. Z. 1966. Random sampling of random processes: Impulse processes. *Information and Control 9,* 4 (Aug.), 347–363.

Lewis, J.-P. 1989. Algorithms for solid noise synthesis. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, J. Lane, Ed. Vol. 23. ACM Press, 263–270.

Lloyd, S. P. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory 28,* 2 (Mar.), 129–137.

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 *Computer Graphics (SIGGRAPH '87 Proceedings)*, 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 *Computer Graphics (SIGGRAPH '91 Proceedings)*, T. W. Sederberg, Ed. Vol. 25. ACM Press, 157–164.

Ostromoukhov, V. 2007. Sampling with polyominoes. *ACM Transactions on Graphics (SIGGRAPH '07 Proceedings) 26,* 3 (July), 78.

Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics (SIGGRAPH '04 Proceedings) 23,* 3, 488–495.

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, Aire-la-Ville, Switzerland, 215–226.

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

Snyder, D. L. 1991. *Random Point Processes in Time and Space*, 2nd ed. Springer-Verlag, New York.

Ulichney, R. A. 1988. Dithering with blue noise. *Proceedings of the IEEE 76,* 1 (Jan.), 56–79.

Weisstein, E. W. Hypersphere packing. From MathWorld – A Wolfram Web Resource.

Worley, S. P. 1996. A cellular texture basis function. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, H. Rushmeier, Ed. Vol. 30. ACM Press, 291–294.

Yellot, Jr, J. I. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 221*, 382–395.