SAND2011-0260C

Maximal Poisson-Disk Sampling

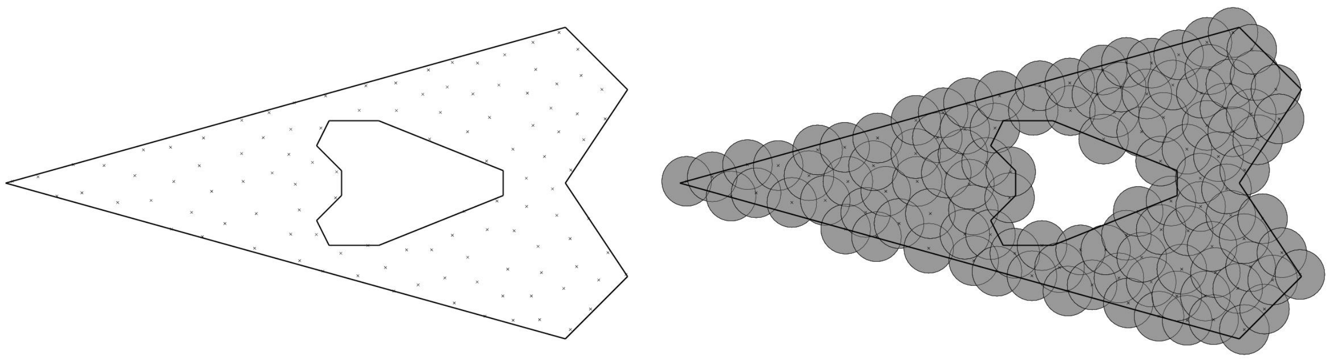

Figure 1: *Maximal Poisson-Disk Sampling of a non-convex domain with a hole (left). The associated Grey-shaded disks show that this Poisson-disk sampling is indeed maximal (right).*

Abstract

We solve the problem of generating a uniform Poisson-disk sampling that is both maximal and unbiased. The method is based on classical dart-throwing with a background grid of square cells for efficiency. In the first phase classical dart-throwing covers much of the domain. A second phase calculates the connected components of the remaining uncovered voids, and uses their geometry to efficiently place unbiased samples that cover them. Our second phase converges quickly, overcoming a common difficulty in dart-throwing methods. Our algorithm is simple, easy to implement, and can handle non-convex domains. The memory and expected running time is linear in the output size, the number of points in the final sample. Our serial implementation shows this is achieved in practice; and we also have a parallel implementation.

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

Keywords: poisson disk, maximal, provable convergence, linear complexity, sampling, blue noise

1 Introduction

Maximal Poisson-disk sampling distributions are useful in many applications. In computer graphics these distribution are desired because the randomness avoids aliasing, and they have the blue noise property. Blue noise means the inter-sample distances follow a certain power law, with high frequencies more common. The lack of low frequency noise produces visually pleasing results for rendering, imaging, and geometry processing [Pharr and Humphreys 2004]. The bias-free property is crucial also in fracture propagation simulations. In this process, a random point cloud is required to minimize the effect of the dynamic re-meshing on the direction of the crack growth. A maximal distribution improves the quality bounds and performance of meshing methods such as Delaunay Triangulation [Attali and Boissonnat 2004].

Poisson-disk sampling is a process that selects a random set of points, $X = \{x_i\}$, from a given domain, $\mathcal{D}$ in n-dimensional space. The samples are at least a minimum distance apart, satisfying an empty disk criterion. In this work we restrict to the uniform case, where the disk radius, r, is constant regardless of location or iteration. The maximal condition requires that the disks are simultaneously closely packed together, in the sense that the sample disks cover the whole domain. Any sampling process that is maximal must terminate, since no more points can be added to the sample set. Bias-free means that the expected number of sample points inside any sub-domain is proportional to the area of the subdomain. This is usually achieved by ensuring that the probability of selecting a point for the next sample is equal to the probability of selecting any other point, provided these points are not already inside some prior sample's disk.

$$\text{Bias-free:} \quad \forall x_i \in X, \forall \Omega \subset \mathcal{D} : P\left(x_i \in \Omega\right) = \int_{\Omega} \mathrm{d}\omega, \tag{1a}$$

$$\text{Empty disk:} \quad \forall x_i, x_j \in X, x_i \neq x_j : ||x_i - x_j|| \geq r, \tag{1b}$$

$$\text{Maximal:} \quad \forall x \in \mathcal{D}, \exists x_i \in X : ||x - x_i|| < r. \tag{1c}$$

Despite the desirability of this distribution, it has been quite challenging for the community to discover an efficient algorithm that satisfies all three conditions. For a detailed survey of Poisson sampling methods see Lagae and Dutre [Lagae and Dutre 2008]. Virtually all existing methods either solve a relaxed version of the problem or require unbounded computational resources. That is, the classical dart-throwing [Dippe and Wold 1985; Cook 1986] approach is unbiased but the probability of the next candidate point satisfying (1b) vanishes as the algorithm progresses, requiring an infinite amount of time to become maximal. Tile-based methods improve the performance, but sacrifice the bias-free condition. For example Wang tiles [Cohen et al. 2003; Lagae and Dutre 2005] requires a biased Voronoi relaxation step to satisfy the empty-disk condition. Penrose tiles [Ostromoukhov et al. 2004; Ostromoukhov 2007] is another example where each tile has a single sample, but Voronoi relaxation is required to reduce sampling artifacts. Another class of methods improves efficiency by computing samples on the fly [Mitchell 1987; Jones 2006; Dunbar and Humpherys 2006; Bridson 2007]. However, these methods are biased and require relatively large storage. Dunar et al. [Dunbar and Humpherys 2006] proposed a linear-time advancing front method where each new sample is picked from a region near to prior samples. Each new point has the same distance to its nearest neighbor, which violates the bias-free condition. Grid-based methods have emerged recently and are very efficient. Wei [Wei 2008] proposed a parallel sampling method that employs a sequence of multi-resolution uniform grids in the dart throwing process. This method is not bias-free and terminates without achieving a maximal distribution. To improve the ability to reach a maximal distribution, White et al. [White et al. 2007] uses a tree-based method to capture the remaining void and select new samples. The memory requirement of the algorithm has been further improved by a similar method proposed by Gamito and Maddock [Gamito and Maddock 2009]. However, tree-based refinement methods require unbounded memory storage if a maximal distribution is desired.

In this paper, we present a simple, yet very effective algorithm to solve the maximal Poisson-disk sampling problem. Our algorithm inherits many desired properties from Wei's algorithm [Wei 2008] such as simplicity and efficient parallel implementation using GPUs. Moreover, it generates bias-free maximal distributions over non-convex domains while consuming limited resources. To our knowledge, this is the first practical algorithm that simultaneously satisfies all the requirements of a maximal Possion-disk sample. The sampling process is achieved through two phases. For efficiencty we use a background grid of square cells covering the whole domain. Each cell can accommodate a single sample. In the first phase darts are thrown into these cells. The initial darts are unlikely to overlap so the algorithm starts very fast, but slows down as more darts are placed, so we switch to the second phase. The first phase leaves many small empty voids bounded by circles and grid cells. These are approximated by convex polygonal voids. During the second phase, darts are thrown directly into the voids, with probability proportional to the relative areas of the voids, which maintains the bias-free condition. The algorithm is capable of tracking the remaining voids in the domain up to round-off error. A maximal distribution is achieved when the domain is completely covered, leaving no room for new points to be selected. The serial implementation of our algorithm is capable of generating one million samples from a square domain in less than 10 seconds. Moreover, our algorithm is capable of handling non-convex domains which is the typical input in many meshing applications.

HERE WE SHOULD EMPHASIS ON THE PARALLEL IMPLEMENTATION CONTRIBUTION AND RESULTS ....

In the rest of this paper, we describe our algorithm in gradual steps. In section (2) we go over the various steps of the two-dimension sequential algorithm. The 3d sequential implementation is presented in Section (3). Parallel implementation aspects are discussed in Section (4). Finally application examples are presented in Section (5) to demonstrate the efficiency of the proposed method and the quality of the output distributions.

2 Two dimensional sequential sampling

Our two phase-algorithm utilizes an active pool of cells to guide the dart throwing process. During Phase I, each cell in the active pool is square-shaped and can accommodate a single sample. Once a dart is thrown successfully into a randomly-selected cell, this cell is invalidated and removed from the active pool. An *invalid cell* is a cell that does not have any room for a new sample. If a thrown dart violates the empty-disk condition, the associated sample is rejected and that trial is considered as a *miss*. The first phase ends after a given number of successive misses, N. At this point, the area the remaining void, Ω, is relatively small compared to the total area of the domain, $\mathcal{D}$. This fact is implied by the probability of achieving N successive misses in Equation 2.

$$P\left(x_{i=1,2,\ldots,N} \notin \Omega\right) = \left(1.0 - \frac{\int_{\Omega} \mathrm{d}\omega}{\int_{\mathcal{D}} \mathrm{d}\omega}\right)^{N} \tag{2}$$

The remaining voids at the beginning of Phase II usually consists of small regions distributed all-over the domain. We loop over the remaining valid cells and modify their boundaries such that we have a better representation of the remaining void inside each cell. This process reduces the area of the targeted domain to be almost the same as the area of the remaining void. Virtually all the methods in the literature have hard time trying to select a point from a small void in the domain and this becomes more challenging as the void gets smaller. It is quite interesting that this is not the case in our method. As the void gets smaller, its linear representation has almost the same area so the chances of inserting a point into that void is actually better.

Our algorithm consists of the following steps:

1. Generation of a background grid and identifying invalid and boundary cells,
2. Dart throwing procedure via a set of square cells,
3. Generation of linear representation of the remaining voids,
4. Dart throwing procedure via a set of polygonal cells.

2.1 Generation of a background grid

The input of this algorithm is a set of edges defining the input domain. External boundary edges are oriented in a counter-clock wise direction while the edge bounding a hole in the domain are oriented in a clock-wise direction. This orientation is crucial for identifying the interior of the domain. Another constraint is that each edge must have a larger length compared to the radius of the output distribution. Our algorithm can also accommodate point set input, prescribed sample points, embedded in a bounded domain. We start the algorithm by generating a uniform grid covering a bounding box of the input domain. Each cell in this grid is identified using a single index. The spacing of the uniform grid is given by $\frac{r}{\sqrt{2}}$. Hence, each cell can only accommodate a single sample. Note that storing the coordinate of the grid lower left corner, the spacing as well as the number of rows and columns is sufficient to identify any cell using its index. This information can be stored using a few variables independent of the output distribution. Each cell in that uniform grid is associated with two classifications. A *valid cell* is a cell that has room for a new sample to be selected, and a *boundary cell* is a cell intersecting one or more of the input edges. These two classifications are stored using two boolean arrays to get the best performance of the algorithm. Sampling a million points in a square takes three million cells, which consumes less than 0.2 MB of memory.

After generating the uniform grid, we identify the boundary cells. For simplicity we re-use some of our main-algorithm machinery. We generate points uniformly along each edge, and locate the hit cell for each point. Some cells might be missed because an edge could graze a corner and have a short length inside the cell. These missed cells are neighbors of hit cells, and are recovered by checking the intersection of the edge with the sides of the hit cell, using only integer operations. The cells interior to the domain are

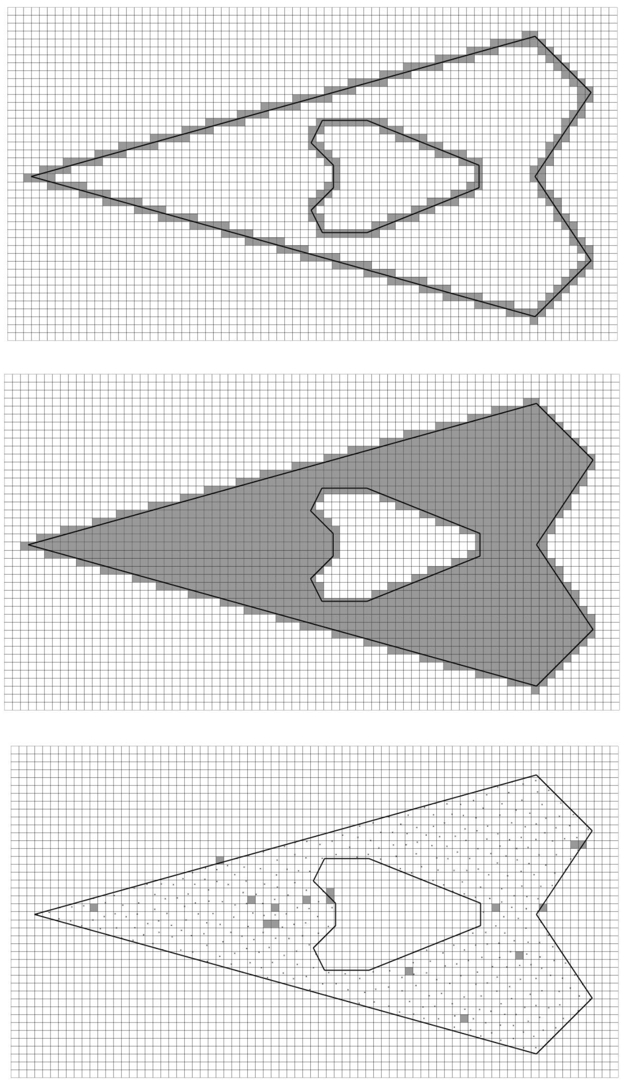

Figure 2: *Phase I of Our algorithm. Boundary cells are first detected (Top), then utilized to identify valid cells (Middle). The first phase ends leaving few valid cells behind (Bottom).*

distinguished from those exterior to the domain using a flood-fill algorithm. Exterior cells are invalid by default, and interior cells are marked valid. This background grid and flood-fill technique is demonstrated using Figure 3.

For the purposes of simplifying the proofs, we assume that the intersection of a cell with the domain is connected. If not, we use Reiman sheets, duplicating a cell once for each connected component, with the cell containing the appropriate boundary pieces. In practice this step is unecessary, and we merely keep track of all the connected components the same way we keep track of disconnected voids.

2.2 Dart Throwing using squared-shaped cells

Since all the cells have the same area, The dart throwing procedure can be executed by selecting a cell randomly from the active pool and then we draw a random sample from the selected cell. If this process was not successful i.e. the selected sample violates the empty-disk condition, we count that iteration as a miss and throw

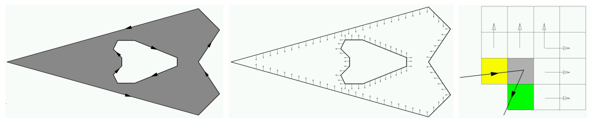

Figure 3: *Boundary edges are oriented based on their type (Left). This orientation governs the propagation of the valid cells signal (Middle). Some cells may have two propagating directions (Right).*

another dart. Note that the empty-disk check is executed in constant time since we need to test the selected sample against the points already inserted in the two layers surrounding the selected cell. If the selected cell was a boundary cell, the selected point has to be checked against the associated edge to ensure that it lies in the domain. If the process was successful, we accept the selected sample and invalidate the associated cell. This loop is terminated after reaching a given number of successive misses, N. In our implementation, we set $N = 500$. At this point we deduce that the chance of utilizing the square-shaped cells in inserting more point is quite low. Thus we switch to Phase II.

2.3 Generation of the linear representations of the remaining voids

For each valid cell we capture each remaining void using a linear representation. This linear representation is achieved by the following steps, which are illustrated in Figure 4:

1. Construct an initial polygon defining the boundaries of the valid cell with respect to the domain interior.

2. Iterate over the neighbor discs and for each disc do the following:

   (a) Retrieve any corner of the polygon that lies inside that disc. If the disc does not contain any polygon corner, proceed to the next disc.

   (b) Insert two corners at the intersection of the disc and the two polygon edges with endpoints inside the disc

   (c) Delete all the polygon corners and edges that lies inside the disc excluding its boundaries.

   (d) Adjust the location of the new polygon corners to account for a disc that we already iterated over.

3. Split the generated polygon iteratively till each polygon contains one isolated void at most. Note that a cell can contain a maximum of three isolated. This step is illustrated in Figure 5.

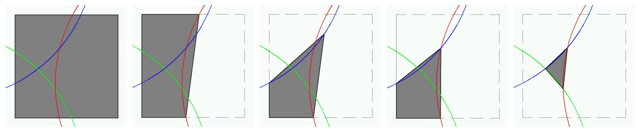

Figure 4: *Generation of the linear representation of a void entrapped between three circles. A polygon is constructed initially using the cell boundaries. Three boolean subtractions are utilized to retrieve a better linear representation of that void. During these linear subtractions the position of the polygon are adjusted to account for old circles.*

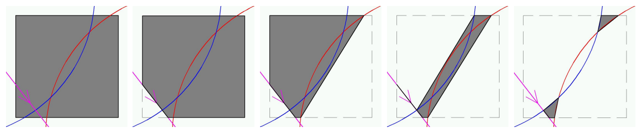

Figure 5: *Generation of the linear representations of two void entrapped between two circles and a boundary edge. A polygon is constructed initially using the cell boundaries. It is then modified to respect the boundaries of the domain. After a polygon is generated it split into two to cature the isolated voids in this cell.*

2.4 Dart Throwing using linear polygonal cells

This is similar to the previous dart throwing algorithm. However, the random selection of a cell from the active pool takes into account the relative area between the cells. After selecting a cell, we choose a random point from that cell and utilize the background grid to check whether the selected point violates the empty-disk condition or not. If the process was a success, the cell is removed from the pool and the remaining cells are updated keeping the area of the cells in the pool very close to the area of the remaining voids. This property increases the probability of successful dart throwing as the remaining voids get smaller. The algorithm terminates when the active pool is empty. In order to have a better performance, we implemented this step iteratively, where we just mark a cell to be invalid instead of removing it from the selection pool. This eliminates the need to update the relative areas of the remaining valid cells since we can select an invalid cell, which will be a miss in that case. After a given number of successive misses, we remove all the invalid cells from the selection pool and update the relative areas of the remaining valid cells. This process continues till we have an empty pool which indicates that a maximal distribution has been achieved.

WE NEED TO PROVE THAT THE ALGORITHM WILL ALWAYS CONVERGE ... probabilistic approach !!!

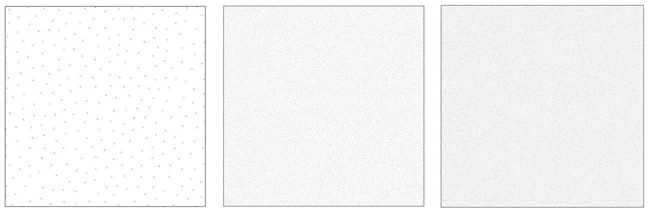

Figure 6: *Three maximal Poisson distributions using a unit square domain and three different densities. Our serial implementation is capable of generating 100,000 maximal bias-free Poisson samples/second.*

2.5 Correctness and Linear Complexity

Let n be the number of darts in the domain after the algorithm terminates. We first show that we do not have too many cells.

Theorem 1. *The total number of cells intersecting the interior of the domain $|\mathcal{C}|$ is $\Theta(n)$.*

Proof. $|\mathcal{C}| = \Omega(n)$ because each cell contains at most one dart. For the other direction, we charge the empty cells to full cells, those containing a dart. For interior cells, every point of every cell is covered by at least one disk. The area of each disk is πr^2 and each cell is $r^2/2$. So the ratio of interior cells to darts is at most 2π. For each boundary cell (intersecting the boundary of the domain), pick any point of the cell in the interior of the domain. There is at least one disk covering that point. That disk is inside some 4×4 grid of cells, so can cover at most 16 boundary cell points. □

Lemma 2. *The expected fraction of a boundary cell that is interior to the domain is 0.5.*

Proof. The correctness of this lemma relies on several assumptions about the domain. We assume r has been chosen so that any cell contains at most one vertex, or at most one edge bounding the domain. We assume that the fraction of cells containing domain vertices is small; if not, then we can add a constant to the probability of a miss in Phase I, and we transition early to Phase II where these cells are handled well. For any cell containing a boundary edge, we assume that which of the two sides is interior to the domain are equally likely, so the expected fraction interior is 0.5. □

In the following we assume that cells external to the domain have been discarded and we are only generating darts for cells containing some portion of the interior of the domain.

2.5.1 Phase I misses

Here we prove that the Phase I dart-throwing algorithm is bias free and each throw has constant complexity. When Phase I completes, the expected fraction of the domain covered by darts is xxx and the expected running time is xxx.

ZZYK

Let b be the ratio of the number of boundary cells to interior cells. b depends on both the geometry of the domain and r.

The expected area covered by a dart is $\pi r^2 - \delta$, where δ = xxx something dependent on b, and the number of successful darts so far because darts may overlap

TODO:

Likelihood of a dart miss in Phase I.

If the dart is in a boundary cell, the expectation of it falling outside the domain is 0.5.

First consider a dart in an interior cell, where all 21 of its template neighbors are interior as well.

The expected fraction of misses is

TODO: Bound the total number of misses to the number of successive misses.

2.5.2 void convexity and complexity

For the purposes of assigning a disk center to a unique square, squares are considered open on their minimal extremes, as in Figure 7. We call such squares *half-open squares*.

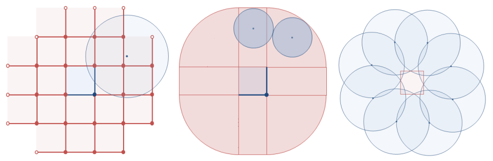

Figure 7: *Left. Any r-disk intersecting the central half-open square is assigned to a unique square within this template. Center. Any such r-disk induces an $r/2$-disk entirely inside this region. Right. 8 disks may overlap a square forming a single remainder region.*

Lemma 3 (Civilization[Meier and et al.] template)**.** *r-disks that intersect a half-open square are assigned to one of 21 squares, up to two squares away, within a 5×5 grid of squares with the corner squares removed, as in Figure 7.*

Proof. Only these squares have points that are less than r away from the center square. The corner squares of the 5×5 grid have a corner point that is exactly distance r from a corner point of the center square, but any r-disk centered at one of these corners is either assigned to a closer square within the template, or the disk intersects the closed but not half-open central square. □

This shows that checking if a dart is in any disk is a constant time operation, that any void is bounded by a constant number of disks, and that any square contains a constant number of voids. The bounds provided by Lemma 3 may be tightened by using area, angle, and distance arguments. For linearity in Phase II, it remains to show that a constant fraction of each polygonal void is outside any circle.

Lemma 4 (disks in square by area)**.** *No more than 15 r-disks can intersect a square.*

Proof. An empty-disk system of r-disks induces a system of *non-intersecting* $r/2$-disks with the same centers but half radius. Any r-disk overlapping the square has its $r/2$ disk completely within distance $3r/4$ of the square. The region that is at most distance $3d/4$ from a square has area just under $5.77r^2$. Each $r/2$-disk covers a disjoint subset of this reagion, of area $\pi r^2/4$. Hence at most $15 = \lfloor 5.77r^2/(\pi r^2/4) \rfloor$ disks can "fit" close enough to the square to intersect it. □

For voids we will improve the 15-disk bound to 9. Figure 7 is a construction showing that 8 disks can bound a void.

A *void* V_r is one connected component of the non-empty intersection of a square, together with the closed complement of some d-disks. V_r is an arc-gon. A *polygonal void* V_p is the convex hull of V_r. For convenience in the proofs, we retain *flat* vertices v of V_r as vertices of V_p when the angle of V_p at v is $180°$. Note that V_r is closed and bounded, and V_p is an outer approximation to V_r. The *polygonal angle* at a vertex y of V_r will mean the interior angle between segments $\overline{xy}$ and $\overline{yz}$ where x, y, and z are consecutive vertices of V_r. $\overline{xy}$ denotes the line seqment between x and y and $|xy|$ denotes the straight-line distance between x and y.

Our next series of lemmas shows that all the vertices of V_r appear as vertices of V_p, and we bound the size and shape of voids. For simplicity we assume that a square is completely interior to the domain. At the end we relax this assumption and note that the changes to the results are slight.

Lemma 5 (kites)**.** *The angle subtended by a chord $\overline{xy}$ is twice the angle α between the circles tangent at x and the chord. And $\alpha =$* $\arcsin(|xy|/2r)$.

Proof. Similar triangles; see Figure 8. □

Theorem 6 (convex corners)**.** *The polygonal angle at a vertex of V_r is at most* $180°$.

Proof. See Figure 8. We have two cases. In the first case vertex y is at the intersection of a circle C and square edge e. At worst x and z are also on e, in which case the angle is $180°$. In the second case y is at the intersection of two circles C_x and C_z. The angle between the tangents of C_x and the line between the points of intersection between the two circles is at most $60°$, achieved when c_x lies on C_z. The angle between the tangent of C_x at y and the chord $\overline{xy}$ is

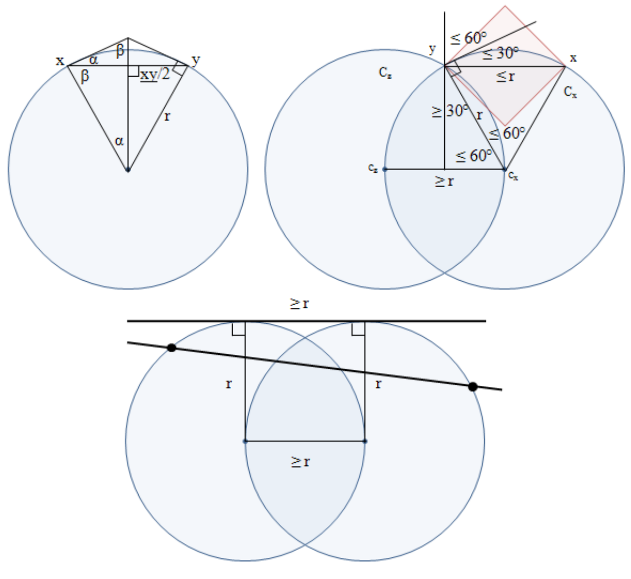

Figure 8: *Left, some chord-angle identities. Right, an upper bound on chord lengths implies an upper bound on polygonal angles. Bottom, only one circle bounding a void can intersect a square side twice.*

$\arcsin(|xy|/4r)$ from Lemma 5. Since the chord must lie inside the square, $|xy| \leq r$, and this angle is at most $30°$. The angles between the circles-intersection line and $\overline{yz}$ is also at most $90°$, so the sum of these angles is $\leq 180°$. (This second case might be tightened to $150°$, since not both chords can be of length r.) □

Corollary 7 (naturally convex)**.** *All vertices of V_r are on the boundary of V_p. Boundary edes of V_p are chords of circles and subsegments of square sides.*

Corollary 8 (one arc)**.** *A circle contributes at most one arc to V_r.*

Proof. The exterior (relative to the circle) polygonal angle between any three points on a circle is reflex, which implies any polygon containing three points of a circle would have a reflex angle somewhere. Since V_p is convex and has no reflex angles, at most two of its vertices can lie on any one circle. □

Lemma 9 (convex centers)**.** *The centers of circles bounding V_r must be in convex position.*

Proof. Note any the circle center is outside V_r, else the square would be covered by that circle. Suppose that three circles C_x, C_y, and C_z touch V_r, but that y is not in convex position, meaning that for some point p of V_r on C_y, $\angle zyp + \angle pyx > 180°$. Assume C_y intersects both of the other circles. Consider the arc of C_y between the intersection points that touches V_r, and assume it is shorter than half the circle perimeter. (This arc is unique by Corollary 8.) Then the arc's chord is longer than r, the diagonal of the square. Hence the other two circles are too far away from one another to both intersect V_r. If the three circles do not intersect, or the arc is longer than half the circle perimiter, then the other two circles are even farther away. See Figure 8. □

Lemma 10 (10 arc sides)**.** *Less than* 10 *circles bound V_r.*

Proof. We show that for r-disks bounding V_r, the distance between the centers of the two farthest-apart circles c_x and c_y must be $> 3r$, so not both can overlap a square with diagonal r. By symettry and

Lemma 9 the closest c_x and c_y can be is if all circles are arranged on a regular n-gon, with side length r. By Lemma 5 the circumscribed circle C for this n-gon has radius $R = r/(2\sin(180/n))$. For $n \geq 10$, we have $R > 1.6r$. For even n, the two farthest apart disk centers are diametrically opposed on C. For odd n, they are slightly closer; $\angle c_x c c_y = 180(1 - 1/n)$ and $|c_x c_y| = 2R\sin(\angle c_x c c_y/2) = r\sin(90(1 - 1/n))/\sin(180/n)$. This last is monotonically increasing with n, and for $n \geq 11$ is $> 3.5r$. □

Perhaps a stronger argument could show that 9 is impossible. In any event, we have a construction in Figure 7 which shows that 8 is possible. That construction may be modified by moving some of the circles outward, exposing some of the square sides to be included in the void V_r. Indeed, four of the circles may be positioned so that each intersects one of the square sides twice, with both intersection points forming flat vertices of V_r.

Lemma 11 (8 square sides)**.** *The square contributes at most* 8 *sides to* V_r*, and only* 4 *if flat sides are ignored.*

Proof. Any circle intersecting a side exactly once (and non-tangent) contains one of the corners of the square, and so does not increase the number of subsegments of the side bounding V_r. See Figure 8. There can only be one circle that intersects one of the four sides twice or is tangent to it as follows. Consider the points of outer tangency for a supporting line to two d-disks not containing each others centers. The tangent points are at same distance from each other as the disk centers are from each other, which is at least d. Hence any line through these disks has farthest points of intersection $> d$ apart. The length of the side of a square is $d/\sqrt{2} < d$. □

Putting the prior lemmas together we have the following theorem.

Theorem 12 (few arc-gon sides)**.** *The number of sides of* V_r *is at most* 17*, and at most* 13 *if flat sides are removed. Figure 9 realizes a void with 16 sides, and Figure 9 shows a void with 12 non-flat sides.*

The preceeding considered only squares that did not contain the boundary of the input domain, but most of the proofs only relied on squares being contained in a circle of radius $r/2$. For boundary squares, we note that they may have at most two edges of the input domain (sharing a common vertex), and their seqments on the boundary of V_r are of length $< r$, so that the number of sides increases by at most 2.

2.5.3 area ratio of arc-void to polygonal-void

We now consider the shape of the voids, specifically the ratio of the area of V_r to V_p, since that determines the expected number of dart-misses in Phase II.

Theorem 13. *The ratio of the area of* V_r *to the area of* V_p *is at least a constant. THE CONSTANT IS UNKNOWN AND THE PROOF WOULD NOT PRODUCE A TIGHT BOUND, BUT IT IS A START. MAYBE A NON-LINEAR BOUND IS POSSIBLE AS THE AREA GETS SMALLER.*

Proof. Consider the circles defining the boundary of a void. We include circles intersecting a square side twice, because those circles affect the valid area for placing a dart. Consider the Vornoi cells of the circle centers; actually consider the weighted Vornoi region of the circles [Edelsbrunner and Shah 1992]. Assume for now that the remainder region is bounded entirely by r-circles, and truncate the Voronoi cells at the polygonal void V_p.

For any circle C, its Voronoi cell will contain the circle chord on the polygon boundary χ, the arc-boundary s, and a part of the interior of V_r. The reasons are as follows. Let VC be the cicles trunctated Voronoi cell, and VS_{ext} its partition outside C, and VS_{int} its part inside C. Recall from Lemma 9 the circle centers are in convex position and can be considered in order around the boundary of the void. Since only the consecutive circles around the void may overlap with C (else the void would not be connected), the chord is not inside any other circle so it is in VC. Also, the Voronoi line of equal distances between C and a non-consecutive circle lies strictly outside C. Since by Lemma 10 there are at most a constant number of circles (¡10), there are a constant number of straight sides bounding VC. All of these bounding sides lie outside VS_{ext} as well. At worst these sides approach tangency with C, and form a 9-sided polygonal outer approximation to the arc. Since the arc s has constant curvature, the area of VS_{ext} is at least a constant fraction of VS_{int}. We do not work out the exact constant because this bound is not very tight; for example much fewer than 9 circles can be packed close enough to be nearly tangent with C.

Now relax the assumption that the remainder region is bounded entirely by circles. Treat the lines supporting the square sides or domain boundary as infinite-radius circles centered at infinity, and all the arguments of the prior paragraph still hold. The area ratio bound constant can be reproduced by assigning the Voronoi regions of the infinite-radius circles to the closest r-circle, since for the infinite-radius circles the arc-gon and polygon are identical. □

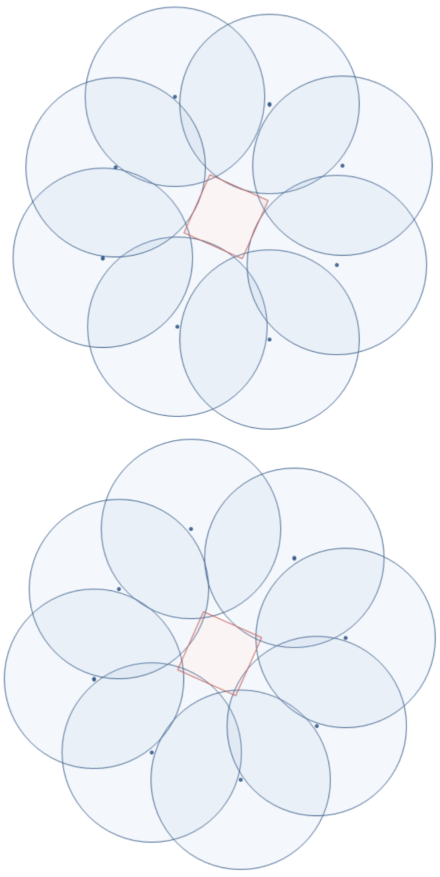
Figure 9: *Left, a void with 16 sides. Right, a void with 12 nonflat sides.*

2.5.4 number of voids per cell

We now consider the number of voids that may appear within a square. Since at most a constant number of circles intersect a square, combinatorics implies the number of voids is constant, but that bound is very weak, so we improve it here.

We call two voids *adjacent* if they each have a vertex a_{xy} and b_{xy} that is the intersection of the same pair of circles C_x and C_y. Those circles are called *consecutive*. We first consider three sided remainder regions, and label their features as in Figure 10.

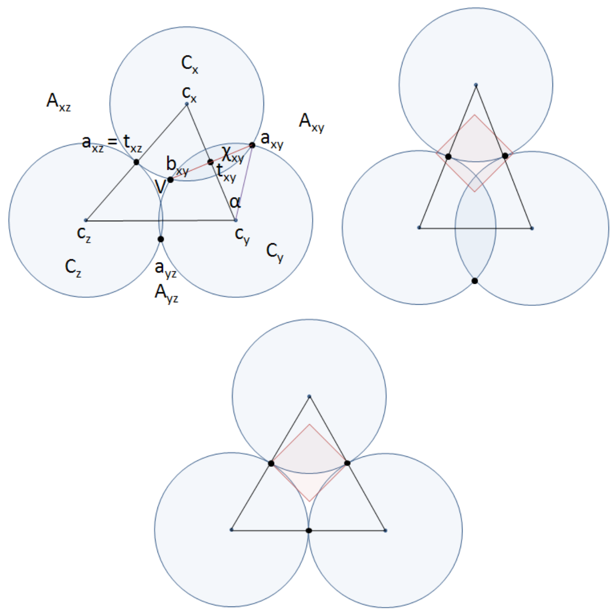

Figure 10: *Top, labeling of a three-sided void. If C_x and C_z are tangent, then $t_{xz} = a_{xz} = b_{xz}$ coincide. Middle, the 3-sided void with smallest distance between an adjacent pair of voids. Bottom, the 3-sided void with smallest distance between the second-closest pair of adjacent voids.*

If we keep the distance between two pairs of circles constant and vary the third by moving one of the circle centers, we observe the following inverse relationships about the distances between pairs of circle centers and pairs of void vertices.

Lemma 14 (circle void distances)**.** *If a_{xy}, a_{yz} and a_{xz} are in the same cell, then*

$$|a_{xy}a_{xz}| \uparrow, |a_{xy}a_{yz}| \downarrow, |a_{xz}a_{yz}| \downarrow \iff |c_yc_z| \uparrow$$

$$|a_{xz}a_{yz}| \uparrow, |a_{xz}a_{xy}| \downarrow, |a_{yz}a_{xy}| \downarrow \iff |c_xc_y| \uparrow$$

$$|a_{xy}a_{yz}| \uparrow, |a_{xy}a_{xz}| \downarrow, |a_{yz}a_{xz}| \downarrow \iff |c_xc_z| \uparrow$$

Proof. By symmetry it suffices to show the first two relationships. First, if C_x is tangent to both C_y and C_z then $|a_{xy}a_{xz}| = |c_yc_z|/2$ by similar triangles. Otherwise, $|t_{xy}t_{xz}| = |c_yc_z|/2$ by similar triangles. By Lemma 9 $\angle t_{xy}c_xt_{xz}$ is less than $180°$, so $|t_{xy}t_{xz}| \uparrow \iff |a_{xy}a_{xz}| \uparrow$. Second, we wish to show $|a_{xy}a_{yz}| \downarrow \iff |c_yc_z| \uparrow$. By similar triagles $|t_{xy}t_{yz}| = |c_xc_z|/2 = \text{constant}$. Also by Lemma 5 $|a_{yz}t_{yz}| \downarrow \iff |c_yc_z| \uparrow$. The requirement that both a_{xy} and a_{yz} are in the cell bounds the chord length to r so by Lemma 5 $\angle a_{xy}c_ya_{yz} \leq 120°$; in particular this angle is non-reflex which extends $|a_{yz}t_{yz}| \downarrow \iff |c_yc_z| \uparrow$ to $|a_{yz}a_{yz}| \downarrow \iff |c_yc_z|$. □

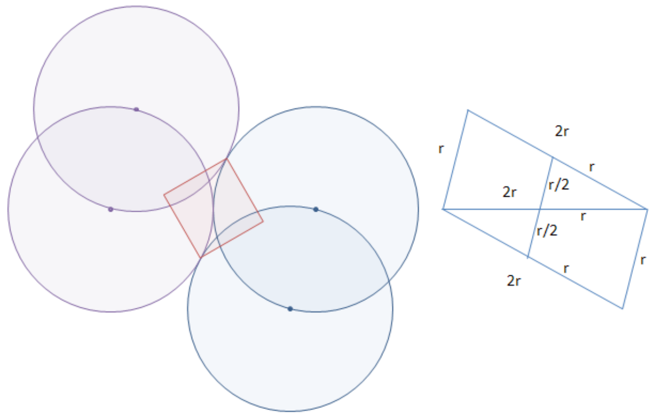

Figure 11: *The configuration minimizing the distance to an adjacent void for two voids simultaneously. The circle centers form a parallelogram, with vertices from four voids along the diagonal of a square. Only three voids can fit strictly inside a cell.*

Lemma 15. *For three sided voids, the distance between consecutive adjacent voids is at least $r/2$. For other voids, the distance between consecutive adjacent voids is at least r.*

Proof. Consider moving the center c_y of circle C_y while keeping all other circles of the void fixed, and keeping C_y overlapping with the same two circles. Using Lemma 14, first varying $|c_yc_x|$ then $|c_yc_z|$ shows that the minimum $|a_{xy}a_{yz}|$ is achieved when C_y is tangent to its consecutive circles. Similar triangles shows that this distance is half the distance between the consecutive circle centers, $|c_xc_z|$. For three sided voids, $|c_xc_z| \geq r/2$, and for all others $|c_xc_z| \geq r$. See Figure 10. □

Corollary 16. *For a void with four or more sides, only two adjacent voids can be in the same cell as the void, and only one strictly inside.*

Proof. The square diagonal is r, so only one pair of points at distance r can be placed inside it. □

Theorem 17. *For a three-sided void, only two adjacent voids can be strictly inside the same cell.*

Proof. We consider the configuration minimizing the second-closest pair of adjacent regions, and show that this distance is large. WLOG $|a_{xy}a_{xz}| \leq |a_{xz}a_{yz}| \leq |a_{yz}a_{xy}|$. We seek to show the first inequality is equality by contradiction. Suppose the first inequality is strict. Then Lemma 14 shows that increasing $|c_yc_z|$ increases $|a_{xz}a_{yz}|$ and decreases $|a_{xz}a_{yz}|$ and $|a_{xy}a_{yz}|$, a contradiction to the configuration being optimal. So $|a_{xy}a_{xz}| = |a_{xz}a_{yz}| \leq |a_{yz}a_{xy}|$. Again Lemma 14 shows that increasing $|c_xc_z|$ will decrease both $|a_{xz}a_{xy}|$ and $|a_{xz}a_{yz}|$, so the optimal configuration has maximal $|c_xc_z|$, or $|c_xc_z| = 2r$. Hence all the disks are at distance $2r$ from one another, and each of the three adjacent cell verticex pairs are distance r apart; see Figure 10. Since the square only has diagonal lenght r, not all three of these vertices can fit. □

Lemma 18. *NEED SOMETHING ABOUT ONLY NEEDING TO CONSDIER POINTS OF CIRCLE INTERSECTION, THEY ARE CLOSEST TO THE FIRST VOID.*

Lemma 19. *NEED SOMETHING ABOUT ONLY NEEDING TO CONSIDER ADJACENT VOIDS. ¡something about only needing to consider adjacent voids¿ If an adjacent void is not in the same cell,* *the... If voids are in the same cell, then two of them are adjacent. non-adjacent voids are farther apart than adjacent ones...*

Theorem 20. *A cell can contain at most three voids.*

Proof. Consider one of the voids in a cell. If it has four or more sides, then Corollary 16 shows that only one more can fit - NEED TO SHOW THAT NON-ADJACENT SIDES ARE EVEN FARTHER APART. Figure 11 shows two three-sided voids, each with two tangent pairs of circles and one maximally overlaped pair of circles. The circle centers form a parallelogram with side lengths r and $2r$. Center the square at the point of tangency between the opposite circles; the other two points of tangency are distance $r/2$ from this center tangency. See Figure 11. By Lemma 15 any other arrangement has adjacent voids farther apart from the center point, so cannot fit in the square. Hence four voids strictly interior to the square is impossible. By moving the square off-center it can contain three voids in its interior. □

Acknowledgements

We thank the U.S. Department of Energy, Office of Science and Sandia's Computer Science Research Institute for supporting this work. We thank Joseph E. Bishop, Mike Stone and Vitus Leung (Sandia) for useful discussions.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

References

ATTALI, D., AND BOISSONNAT, J.-D. 2004. A linear bound on the complexity of the delaunay triangulation of points on polyhedral surfaces. *Discrete Comput. Geom. 31*, 3, 369–384.

BRIDSON, R. 2007. Fast poisson disk sampling in arbitrary dimensions. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, 22.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *SIGGRAPH '03*, 287–294.

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

DIPPE, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. In *SIGGRAPH '85*, 69–78.

DUNBAR, D., AND HUMPHERYS, G. 2006. A spatial datastructure for fast poisson-disk sample generation. *ACM Transactions on Graphics (SIGGRAPH '06 Proceedings) 25*, 3, 503–508.

EDELSBRUNNER, H., AND SHAH, N. R. 1992. Incremental topological flipping works for regular triangulations. In *Proceedings of the eighth annual symposium on Computational geometry*, ACM, New York, NY, USA, SCG '92, 43–52.

GAMITO, M. N., AND MADDOCK, S. C. 2009. Accurate multidimensional poisson-disk sampling. *ACM Transactions on Graphics 29*, 1, 1–19.

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

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

LAGAE, A., AND DUTRE, P. 2008. A comparison of methods for generating poisson disk distributions. *Computer Graphics Forum 27*, 1, 114–129.

MEIER, S., AND ET AL. Sid meier's civilization, http://www.civilization.com. Official Website. versions I, II, III, and IV.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *SIGGRAPH '87*, 65–72.

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

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Transactions on Graphics (SIGGRAPH '07 Proceedings) 26*, 3, 78–1–78–6.

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

WEI, L.-Y. 2008. Parallel poisson disk sampling. *ACM Transactions on Graphics (SIGGRAPH '08 Proceedings) 27*, 3, 1–9.

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