SAND2012-1152C

k-d darts: Sampling by k-Dimensional Flat Searches

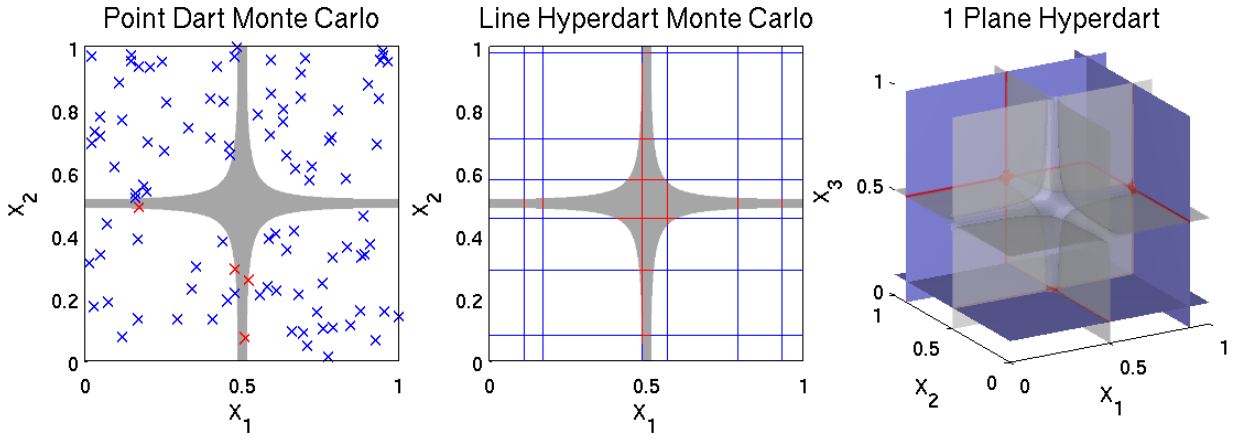


Figure 1: *Sampling a long and thin subregion (gray) using point (left) and line (center) k-d darts in 2d, and plane k-d darts in 3d (right). Point samples (0-d darts) may be cheap to compute and sample, but in many applications where the region of interest is small compared to the sampling space, they may often contribute nothing to the final result. Instead, we propose taking samples of higher dimension, or k-d darts. Their greater expense is offset by a higher hit rate, and they can offer significant performance benefits over point samples in a variety of applications.*

Abstract

In this paper we formalize the notion of a k-d dart, which is a point sample extended to searches along k-dimensional directions. We propose to use k-d darts to sample a function's properties in high dimensions, such as its integral or an exemplar above a threshold. K-d darts can be split or joined depending on whether they meet at a common point, which can influence the bias in the final outcome and the efficiency of the sampling process. We demonstrate that k-d darts are better than point-wise samples in high dimensions depending on the characteristics of the sampling domain, e.g. if the subregion of interest is long and thin and the search is not too expensive.

To illustrate the utility of k-d darts, we present three concrete application scenarios: maximal Poisson-disk sampling, high-quality rasterization of depth-of-field blur, and an estimation of the probability of failure from an emulator for potential use in uncertainty quantification. The three scenarios indicate that the rate of convergence while using k-d darts is potentially much higher that traditional point-sampling approaches, with a corresponding reduction in noise.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: sampling, line search, dimension, Poisson-disk, uncertainty quantification, thin regions, rendering, depth-of-field

1 Introduction

In many graphics and other applications, we are interested in estimating some property of a continuous function over a domain, and the function is not directly analyzable because there is no simple, cheap, and explicit formula for it. For example, the function might be a continuous representation of an image, a complicated graphics rendering convolution, or the output of a parallel physics simulation. *Sampling* is the process of randomly selecting a subset of a domain. The function is evaluated at these subsets, and the overall function properties are estimated based on those values.

In typical sampling processes, the samples are points. However, as the following literature survey describes, a recurring challenge is to deal efficiently with the case that the interesting part of the domain is very small compared to the entire domain. For example, suppose we are interested in estimating the volume of the domain where the underlying function is negative. If this subdomain has a very small volume, only a correspondingly very small fraction of uniform sample points will land in it; see Figure 1. Consequently, our sampling process will be expensive, since most of the samples we take will not contribute to the value we are trying to compute.

To better address this problem, we propose the k-d dart. The key idea behind the k-d dart is that rather than take point (0-dimensional) samples in the domain, we instead take samples of higher dimension. While these samples may be more expensive to compute, they are more powerful, and generate better results for the same amount of effort.

For each sample, instead of evaluating the function at a single point, we evaluate the function along a set of higher-dimensional flats (e.g. lines, planes . . . hyperplanes). A k-d dart denotes the set of flats that form a single sample. An important case of this is where flats are one-dimensional lines. Using our previous example, we may find the points along the line where the function value is zero, and partition the line into segments where the function value f is strictly positive or negative, and estimate the volume where $f < 0$ from the negative-interval lengths. In some cases it is expedient to consider a single sample point, together with all the axis-aligned lines through it. This is called a *joined* k-d dart. In other cases, the statistical dependence of these intersecting lines biases the functional estimate, and *split* k-d darts are needed: each of the lines (flats) is chosen independently.

The purpose of this paper is to formally introduce this technique.

We focus on three motivating applications for concreteness. This both eases the explanation of the technique and shows its use to the graphics community. For all of these, the space is of moderately high dimension, and 1-dimensional line searches are particularly effective. These applications are completing a maximal Poisson-disk sampling in high (e.g. 10) dimensions, rendering depth-of-field blur in four dimensions, and estimating the probability of failure (e.g. 1e-4) from a function emulator in about six dimensions.

2 Previous Work

Our work generalizes sampling, which has been most thoroughly explored, both in graphics and in other application domains, in the application of maximal Poisson-disk sampling (Section 2.1). Our interest here is in applications with large dimensionality, typically beyond the 2d or 3d focus of most previous work. Of course traditional rendering applications make wide use of sampling in a variety of different forms (Section 2.2), many of them with high dimension. In non-graphics applications, the field of uncertainty quantification attempts to quantify the error range of a particular computation, typically by performing the computation many times with a well-distributed selection of input parameters that span the input domain, and because of its importance, we use it as another motivating example (Section 2.3).

2.1 Maximal Poisson-Disk Sampling

Poisson-disk sampling is a popular graphics technique to distribute a set of points in a domain. The points are random, which helps avoid visual artifacts. The points have a minimum distance between them, which also helps to use the point budget efficiently. In a maximal sampling, disks around the points overlap to cover the whole domain, leaving no room to add another point. Poisson-disk sampling patterns generate a blue-noise-like spectrum that is well-suited for the human visual system, but computing such patterns is a challenge.

Many solutions have been proposed, and often achieving maximality is the most challenging part. Many modern methods utilize some form of a background quadtree grid and a refinement procedure to track the remaining voids (uncovered regions) [Dunbar and Humphreys 2006; Gamito and Maddock 2009; Ebeida et al. 2011; Jones and Karger 2011]. This can be made efficient in dimensions up to about 5 [Ebeida et al. 2012].

Uncertainty quantification (UQ) motivates MPS sampling in higher dimensions, e.g. 20. No MPS methods in the literature scale to these dimensions due to the so-called "curse of dimensionality." Classical dart throwing [Dippé and Wold 1985; Cook 1986] is not strongly dependent on dimension, but as the number of accepted samples increases, the runtime for the next sample becomes prohibitive and the algorithm must terminate well before maximality.

Consider sampling a unit box with disk radius r in dimension d. Some issues are fundamental to the problem, independent of any specific algorithm or application. The size of a maximal sampling n is indeterminate, but its lower $n_{\min}$ and upper bounds $n_{\max}$ grow exponentially in $1/r$ and d; see the extensive literature on packing density. The number of disks that can touch another disk grows exponentially in d; see kissing number. (The Lattices catalogue summarizes both [Nebe and Sloane 2012].)

The curses of dimensionality for grid-based methods go beyond these and are manifold:

1. The base grid size grows exponentially and faster than the output point set.
2. A grid cell is refined into 2^d subcells.
3. The number of nearby cells that might contain a conflicting sample grows faster than the kissing number.
4. The ratio of misses/hits grows exponentially with the dimension [Ebeida et al. 2012].

Item 1 arises because the size of the base grid is usually chosen such that each cell can accommodate at most one point: the diagonal of base squares are r and the side length is $r/\sqrt{d}$. Worse, at maximality, the number of empty base cells grows exponentially with the dimension. These empty cells increase the time, and especially the memory requirements, to prohibitive levels. One possible solution is to choose a base grid level with *side* length $2d$, so that every square must contain one point. However, due to Item 2, this approach just defers the problem until cells are required to be refined a couple of times to represent voids. Because of the kissing number, representing voids through geometric constructions [Ebeida et al. 2011] or explicit arrival times [Jones and Karger 2011] do not appear to be viable solutions at high dimension.

Some approximate methods [Wei 2008] put an upper bound of the number of misses per cell, which partly addresses Item 4). The drawback is that the sampling is not maximal. How far it is from maximality has not been analyzed, but volume arguments suggest that for a fixed box size, the number of allowed misses must grow exponentially in d to bound the linear distance between an uncovered point and a sample's disk.

While some of these issues affect runtime, the real curse is the memory requirements for quadtrees. The quadtree method with the best memory scaling by dimension [Ebeida et al. 2012] seems unlikely to extend to even $d = 10$ in the near future. The consequence of the lack of computation and memory scalability is that today's maximal-Poisson-disk point sampling techniques are insufficient for sampling functions of large dimensionality.

2.2 Rendering

High-quality rendering is an important application of multi-dimensional sampling, since photorealistic effects like motion-blur, depth-of-field and soft shadows can be expressed as integrals over multiple dimensions. Classical techniques often employ stochastic point sampling to estimate these integrals. Noise-free rendering using point sampling can require a large number of samples, which can be extremely expensive for complicated scenes.

In this paper we present an alternative to point sampling by formalizing multi-dimensional darts as samples for computing such estimates. However, a few graphics applications do employ multi-dimensional samples in limited and specific scenarios. The OpenGL accumulation buffer [Haeberli and Akeley 1990] uses a form of k-d darts for motion blur: each output pixel is an aggregate of several input pixels, each of which may span a 2d region (pixel area) for a constant shutter time. These 2d snapshots are essentially 2d-darts in 3d space.

Jones and Perry [2000] experimented with using 1d darts for anti-aliased polygon rendering. They shoot single-dimensional darts across a pixel's surface, analytically compute triangle coverage for each of them, and then average them to obtain final pixel colors. In Section 5, we take a similar approach to compute analytical coverage over the lens aperture for 4d depth of field integral. However, unlike their method, our line darts lie in u, v space. They do not span multiple pixels which is both better suited to our screen-space-to-lens-space transformation and parallel-friendly.

Recent research has also shown promise in rendering high-quality

motion-blur using multi-dimensional samples. Gribel et al. [2010] present the use of 1d darts in the x, y, t domain to analytically render motion blur effects in a scene. In their more recent work, Gribel et al. [2011] proposed the use of 2d-darts in the x, y, t domain for motion-blur. They also extend their implementation to render motion-correct ambient occlusion.

Our paper generalizes the idea of multi-dimensional darts for sampling, and it is this generalization that helps us design renderers that use k-d darts in different configurations. We present one such configuration in Section 5 which discusses an extension of Gribel et al.'s work for rendering depth-of-field effects. We use 1d darts to compute efficient and high-quality depth-of-field rendering.

While we demonstrate just one configuration of k-d darts for this application, we emphasize the possibility of several different strategies for employing k-d darts for depth-of-field as well as other effects. In general, k-d darts offer a sampling process that converges much faster than point sampling and has a much lower amount of noise. They can add bias to the process, but careful design can minimize the impact on the final rendering.

2.3 Uncertainty Quantification

Random sampling is one of the oldest [Hall 1873] and most robust methods for uncertainty quantification (UQ). Although it existed long before then, UQ first became a useful tool for solving problems with the advent of computers, specifically during the Manhattan Project of World War II. At that time, Nicholas Metropolis and Stan Ulam named it "Monte Carlo" (MC) sampling as a reference to Monaco's famous casino.

The principal use of MC is to approximate a high-dimensional integral with a sample mean. An relevant example is to compute the probability, i.e. the mean rate of occurrence, that a system's output will lie in a "failure region." The primary drawback of MC is the slow rate at which the sample mean converges to its true value. The Central Limit Theorem states that if X_i is a sequence of independent identically distributed random variables (i.e. independent draws from the same distribution) with finite mean μ and variance σ^2, then as the number of samples, N, approaches infinity the sum of the sequence mean converges to the normal distribution, $\mathcal{N}\left(N\mu, N\sigma^2\right)$. With minor manipulation it follows that

$$\begin{aligned}\lim_{N\to\infty} \frac{\sum_{i=1}^{N}(X_i-\mu)}{\sqrt{N}} &\sim \mathcal{N}\left(0,\sigma^2\right),\\ \lim_{N\to\infty} \frac{\sum_{i=1}^{N}(X_i-\mu)}{N} &\sim \mathcal{N}\left(0,\frac{\sigma^2}{N}\right),\end{aligned}$$

and hence that, in the limit of an infinite number of samples, the standard deviation of the error, also known as the standard error, in the *computed* mean is

$$\sigma_{err} = \frac{\sigma}{\sqrt{N}}. \tag{1}$$

Although this rate of convergence is very slow, the number of dimensions, d, does not appear in Eq. 1. MC's primary advantage is that it is not subject to the so-called "Curse of Dimensionality": the number of samples required for a given fidelity of representation of an unknown function is exponential in d. Significant effort has been invested in developing variants of MC with faster rates of convergence. Some examples include (1) Jittered Sampling (JS) [Liu and Stanley 1965], a space-filling form of MC that randomly perturbs nodes of a tensor product grid; (2) Latin Hypercube Sampling (LHS) [McKay et al. 1979; Owen 1992], often called the more descriptive "N-rooks sampling" in the graphics community; (3) Multi-Jittered Sampling (MJS) [Chiu et al. 1994], a two-dimensional combination of JS and LHS; (4) Binning Optimal Symmetric Latin Hypercube Sampling (BOSLHS) [Dalbey and Karystinos 2010], which generalized MJS to higher dimensions and a number of samples that is not exponential in d; (5) Importance Sampling (IS) [Glynn and Iglehart 1989], which involves preferentially sampling where it is beneficial to do so and compensating by reducing the weight of preferred points; and (6) Markov Chain Monte Carlo (MCMC) [Gilks et al. 1996], briefly summarized as random or drunkard's-walk MC with a carefully constructed path preference. Our work has a similar goal to these methods; in this paper we show that using k-d darts instead of traditional point darts also has the potential to accelerate the convergence of a MC sample mean for certain types of problems.

3 K-d dart Method

Now that we have seen a text description of the method (Section 1) and where it might be useful (Section 2), we define it more formally.

We are interested in analyzing a function f, defined over the domain $\mathcal{D} \subset \mathbb{R}^d$. Often $\mathcal{D} = [0,1]^d$.

A *flat* F is a k-dimensional subspace of the domain, $\{p\} : p_o + \vec{t}v$. The flat contains point p_o. Here v is an axis-aligned normal vector: $v_j = 1$ for j in some indicator set J and 0 otherwise, where $|J|$ is the dimension of the flat. Also $\vec{t}$ is some arbitrary scaling vector and $\vec{t}v$ denotes the vector that is their componentwise product. Thus p_o and v define the flat.

A k-d dart s is a sample defined by a set of flats. The v are the $\binom{d}{k}$ possible zero-one vectors with $|J| = \binom{d}{k}$. For a line-dart, the v are the d elementary vectors. For a *joined k-d dart*, all the flats contain a common point p_o. For a *split k-d dart*, the point for each flat is independent.

To choose a joined dart uniformly at random from $[0,1]^d$, we choose the d coordinates of p_o uniformly at random from $[0,1]$. For a given flat, only the coordinates of p_o where v is zero matter, so we only need to generate those coordinates when specifying each flat for a split k-d dart.

The typical algorithm using k-d darts uses the following approach. Generate N independent darts. (Adaptive sampling is also possible.) For each dart, perform searches to generate the (approximate) imprint of f along each flat. Use a measure of the imprint to estimate the global measure of f. Since this outline is so generic, we next describe three concrete applications.

The meaning of the domain is application-specific. In rendering, the domain combines a 3d scene space with time and lighting to produce a 5d domain. For maximal Poisson-disk sampling, the domain is typically the unit cube. For uncertainty quantification (UQ), the domain is an abstract parameter space, describing the possible configurations of a realized system. For example, one dimension might indicate the range of temperatures that a bridge is exposed to, another the weight load, and others the spatially varying material strengths of concrete.

Each application is concerned with estimating some property of f. In rendering, f is the contribution of one ray (photon) to a pixel, and we seek to estimate the contribution of all photons over a discrete time interval for each pixel. In maximal Poisson-disk sampling (MPS), uncertainty quantification, and other geometric applications, f is an indicator function specifying whether that point of the domain is inside or outside some region of interest. For MPS, f indicates whether a point is inside a previously selected Poisson-disk. For UQ, $f < 0$ indicates where a manufactured system fails,

and the actual value of f indicates how far the system is from failing.

4 Maximal Poisson-Disk Sampling (MPS)

Algorithm 1 A classical dart throwing algorithm using line darts.

```
while maximality estimates are inadequate do
  generate a line dart s^1
  for all i =permutation(1..d) do
    generate line segment(s) g = s_i^1 ∩ domain
    for all samples p do
      subdivide g = g \ D(p)
    end for
    if g ≠ ∅ then
      count s^1 as a hit
      select sample uniformly from g
      skip to next line dart
    end if
  end for
  count s^1 as a miss
end while
```

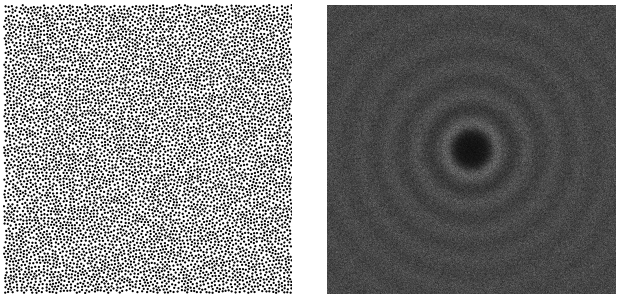
Using k-d darts, we provide a practical solution for the MPS problem in high dimensions, where traditional methods do not scale (Section 2.1). Our implementation is a variant of traditional dart-throwing, which (in its classical formulation) throws point darts into the domain, keeping only those darts that hit uncovered regions. However this algorithm converges so slowly as to be impractical, especially in high dimensions, because any individual point dart is unlikely to hit an uncovered region. Our implementation uses both point and line darts. Casting line darts into the domain is much more likely to intersect an uncovered region, making our implementation much more practical. Algorithm 1 specifies our implementation in detail: briefly, we cast line darts into the domain and intersect them with already accepted samples in the domain to generate a set of segments that are not covered by any samples, then uniformly sample across those segments with a point dart.

The memory requirements are only $O(nd)$, which is what is required to represent the output point cloud because each sample has d coordinates. Only $2n + d$ floats are needed for scratch space for each line dart g. The runtime is $O(dn \log n + nd^2)$ per dart throw. The number of throws is a function of V, the acceptable void volume (the fraction of the domain that is uncovered by samples), and the miss rate; the community lacks the tools to analytically bound the miss rate, but we show that for line darts it can be made to be reasonable, or at least more reasonable than the alternatives.

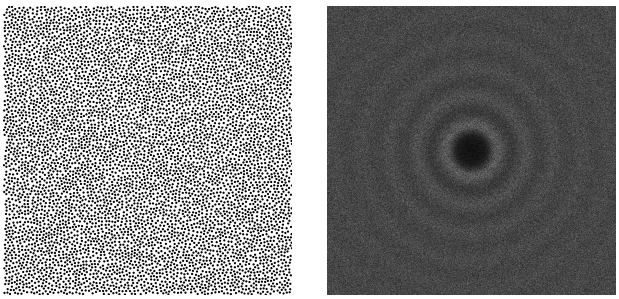
The drawbacks are that the process is biased, but the output point cloud bias is small according to the standard measures: FFT spectrum, power, and anisotropy. The output is not maximal, but its deviation can be estimated.

In order to check the potential bias of line darts, we generated 2d samples using both line darts and point darts. Figure 2 and Figure 3 compare the blue-noise properties associated with the two resulting point clouds; the bias introduced by line darts is, at least for these three metrics, indistinguishable from traditional point darts. In this test problem, the acceptable void volume was less than 10^{-6}. Line darts inserted 173,944 points in 33.241 seconds while point darts inserted 171,341 points in 235.947 seconds.

The increased efficiency of our approach is due to the superior ability of line darts to search the space to find uncovered points. Only extremely simple and one-dimensional data structures are needed. The cost of throwing a line dart is nearly the same as a point dart.

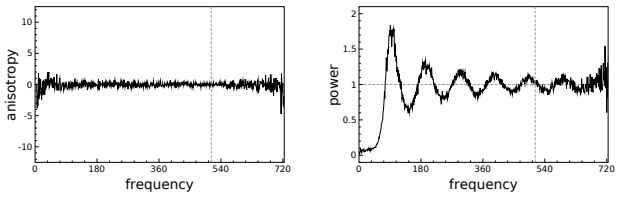


(a) Line darts

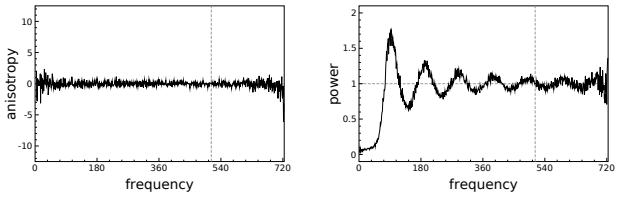


(b) Point darts

Figure 2: *FFT spectrum (right) for the MPS point cloud (left) generated via line darts (top) and point darts (bottom).*



(a) Line darts



(b) Point darts

Figure 3: *Radial anisotropy and mean power estimates for line darts (top) and point darts (bottom), averaged over ten samplings.*

4.1 Implementation Details

k-d tree. To speed up the iteration over inserted points, we utilize a k-d tree that requires low memory since every inserted point is associated with two points only.

Line darts. A split line dart generates d sets of $d-1$ coordinates, in $O(d^2)$ time and space. (A line dart just takes $d-1$ random coordinates.) At the time of this submission, we have implemented the method for joined darts. The maximality estimates for coverage are better for split darts because they are independent.

Computation of the segment array g. Beyond storing the accepted samples, our only data structure is an array of segments in the line l of the line dart. Each stored segment is inside the domain but outside the sample disks processed so far. The set g may be implemented as a one-dimensional linked list storing the ith coordinate of the starts and ends of segments: $g = [a_0 b_0 a_1 b_1 \dots a_q b_q]$ where $q \leq n$ if the domain is convex.

To update g for a new disk $D(c)$ centred at sample c, we first compute $g' = D(c) \cap l = [b', a']$.

Using binary search in $O(\log n)$ time, we find the position in g where b' and a' should appear. If they are beyond a_0 or b_q, the disk does not intersect g. If b' and a' both lie between a_j and b_j, the latter segment is split into two: $a_j b' a' b_j$. Otherwise if b' only lies between a_j and b_j, then a segment is trimmed by replacing b_j by b'; a similar step applies for a'. Any endpoints between b' and a' are covered by the disk and discarded. The updates are $O(1)$ time for a linked list.

Choosing a point from g uniformly can be done by adding the lengths of the segments L, choosing a random number between 0 and L, and using binary search to find the corresponding point on g.

Maximality estimates Suppose that having a remaining void volume V is acceptable, and the probability of hitting the void with a k-dimensional dart (point, line, or higher k) is P_k. Then on average, one out of P_k^{-1} darts will hit the void. Our current terminating condition is to stop after $m = \text{ceil}\,(1/P_k)$ misses. The probability of finding a void of size V is therefore

$$P_{\text{find}} = 1 - (1 - P_k)^m \,. \tag{2}$$

If being able to find the void with a specified probability P_{find} is required, then Eq. 2 can be solved for m instead of P_{find}. For point, i.e. $k = 0$, darts, $P_0 = V$. Assuming a hypercube domain, the worst (hardest-to-find) case for k-d darts is when the void is a hypercube with edge length $b = V^{1/d}$. For joined k-d darts, the worst case P_k is

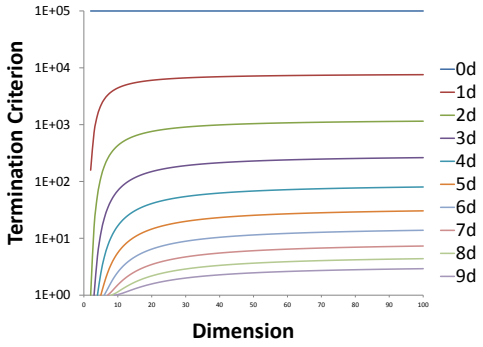
$$P_k = \sum_{i=0}^{k} \binom{d}{i} b^{d-i} (1-b)^i. \tag{3}$$

Figure 4 shows the values of m for different k values using $V = 10^{-5}$ and $V = 10^{-8}$. For lower values of V, using a higher-dimension dart might be more efficient.

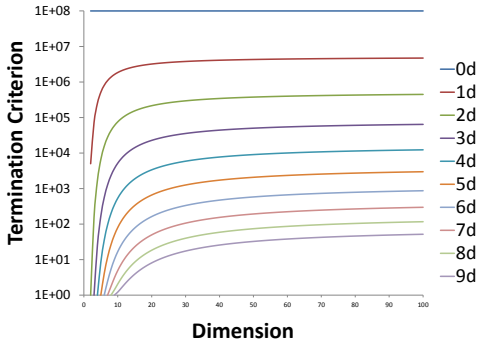
For joined line darts (our current implementation), this simplifies to $P_1 = V + d\left(V^{1-1/d} - V\right)$. For split line darts, the worst case P_1 is $P_1 = 1 - \left(1 - V^{1-1/d}\right)^d$. This analysis suggests that split darts will be even more capable at finding voids than joined darts.

4.2 Experimental Results

We tested our code using 2, 4, 10, and 20 dimensions using point darts and line darts. Figure 5 demonstrates the execution time and the number of inserted points for different values of the acceptable void ratio, V. For low dimensions (e.g. d = 2–4), line darts are more efficient if $V < 10^{-4}$, while for high dimensions, line darts seem to take the upper hand at lower values. Note that for all experiments, line darts were able to insert more points regardless of V and d.



(a) $V = 10^{-5}$



(b) $V = 10^{-8}$

Figure 4: *Number of successive misses required for termination in the case of using joined darts.*

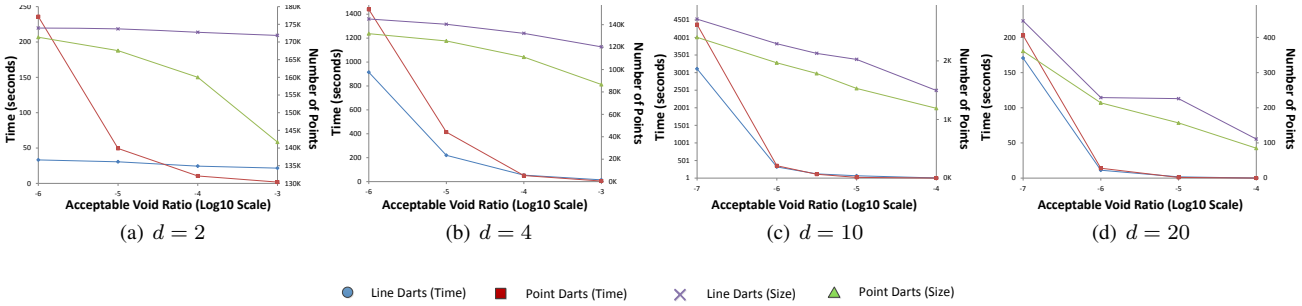
5 Depth-Of-Field Using k-d darts

In this section we present an application of k-d darts for fast and high-quality rendering of depth-of-field in computer-synthesized images. Depth-of-field or defocus is a common visual effect caused due to the finite aperture of the lens in a camera. Objects near the focal plane of the camera appear sharp, while those away from the focus (near or far) appear blurred or out of focus.

Mathematically, computing a pixel's color in the presence of depth-of-field can be expressed as a four-dimensional integral over the pixel's spatial (x,y) and lens aperture (u,v) dimensions. In most high-quality renderers, this integration is calculated using a Monte Carlo integration over many point samples. Practically this method suffers from a low rate of convergence, and reducing noise for a good quality image usually requires a very large number of samples per pixel. This results in a very expensive operation, and presents itself as an important problem in computer graphics.

k-d darts offer themselves as a great alternative in such scenarios, since they promise faster convergence as well as low noise. Thus, we use k-d darts to present an efficient solution to rendering high-quality depth-of-blur. Instead of using point samples for reconstruction, we use 1d (line) darts, thrown in the (x,y,u,v) space. We compute coverage for these darts using a method inspired by Gribel et al. [2010; 2011]'s work on rendering motion blur. However, we use line darts in the u, v domain (instead of x, y, t domain) to compute depth-of-field blur. In that sense, our work can be considered an extension to Gribel et al. [2010] for a broader four dimensional problem. Note that the use of "line sample" by Gribel et al. [2011] our terminology is a 2d or plane dart.

For each triangle and pixel sample (x,y), we throw several line darts



(a) $d = 2$ (b) $d = 4$ (c) $d = 10$ (d) $d = 20$

Figure 5: *Execution time and number of inserted points for dimension d as a function of the acceptable void ratio, V. Smaller V (toward the left on the graphs) is desirable since it approaches the maximal condition; as V becomes smaller, using line darts has an increasing performance advantage. Given the termination condition of the same acceptable void volume, the line-dart approach also consistently finds more points than the point-dart approach for all void ratios.*

in the u, v plane, analytically computing and storing the triangle's coverage along the line dart. Once we have processed all triangles, we resolve the list of coverages for each line dart to obtain the final sample color, which we use to compute the final pixel color. Note that for efficiency reasons, we use a combination of split and joint k-d darts, throwing several u, v line darts per pixel sample (x,y).

5.1 Triangle Edge Equations u, v Domain

For a given triangle, we start by computing a signed radius of circle-of-confusion (CoC) for each vertex, obtained using the following expression [Hammon Jr. 2008]:

$$CoC = A \cdot \left(\frac{f(z - z_f)}{z(z_f - f)} \right)$$

where A and f are the camera aperture and focal length, respectively, and z_f and z indicate the respective depths of the focal plane and the given vertex. The latter is simply the w coordinate of the vertex in clip-space.

Given the lens u and v coordinates ($(u, v) \in [-0.5, 0.5]^2$), we assume a linear apparent motion of the vertex screen coordinates. For a given screen-space vertex x_i,y_i ($i \in 0, 1, 2$):

$$\begin{aligned} x_i^u &= x_i + c_i u \\ y_i^v &= y_i + c_i v \end{aligned}$$

where c_i represents the circle of confusion for the vertex.

We insert the above in the equations for testing whether an oriented triangle edge i (between vertices i and j) covers a point (x,y):

$$ES_i(x, y, u, v) = (y - y_i^v)(x_j^u - x_i^u) - (x - x_i^u)(y_j^v - y_i^v) \geq 0$$

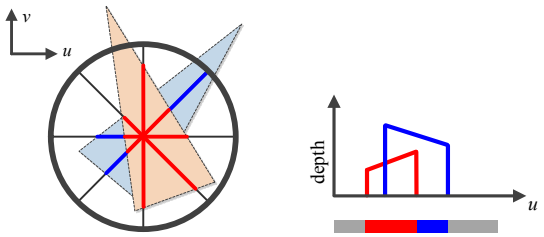
where ES_i represents the edge-sum at (x,y) for the ith edge.

The above expands to:

$$\begin{aligned} ES_i(x, y, u, v) &= (y - y_i - c_i v)(x_j - x_i + u(c_j - c_i)) \\ &- (x - x_i - c_i u)(y_j - y_i + v(c_j - c_i)) \geq 0 \end{aligned}$$

and can be further simplified to get:

$$\begin{aligned} ES_i(x, y, u, v) &= ES_i(x, y, 0, 0) \\ &+ u(c_j(y - y_i) - c_i(y - y_j)) \\ &- v(c_j(x - x_i) - c_i(x - x_j)) \geq 0 \\ ES_i(x, y, u, v) &= C_i(x, y) + uA_i(y) + vB_i(x) \geq 0 \end{aligned}$$



(a) Line darts in u, v domain. Bright portions of the samples indicate coverage. (b) Resolving depth for a sample.

Figure 6: *Our technique for computing analytical coverage using line darts. The u, v domain with transformed triangles and line darts is shown in (a), and depth resolution per sample is shown in (b). Without line darts, we would have to sample many more points within the sampling region to achieve equivalent quality.*

which is linear in u and v. Thus, to compute coverage for a given pixel sample, each triangle edge in screen-space can be transformed into another edge in the lens space. We throw line darts in this space, computing coverage as described in the next subsection.

Note that C_i is simply the edge-sum in a non-defocus scenario, and A_i and B_i can both be partially computed during triangle setup, and updated using simple multiply-accumulate operations on a per-pixel-sample basis.

5.2 Analytical Coverage in the u, v Domain

Knowing the edge equations in the u, v domain, we can now compute their coverage along a line dart. Figure 6 summarizes our technique for determining coverage.

For each pixel sample, we instantiate several line darts along u or v directions. Our current implementation considers darts thrown in a wagon-wheel fashion, although extension to arbitrary darts is straightforward. Note that a wagon-wheel pattern biases the reconstruction towards the center of the lens, which may be desirable depending on the application.

Rendering consists of testing each incoming triangle against potentially covered line darts. For each pixel sample in the triangle

Point Darts	#darts	1	4	16	64	169
	Time (sec)	1.26	4.35	15.15	58.22	150.53
Line Darts	#darts	2	4	8	12	16
	Time (sec)	4.55	8.91	18.44	27.61	37.97

Table 1: *Performance of our point sampler against our line sampler. #darts indicates the number of darts (point or line) thrown per pixel. Each line dart is clearly several times costlier than a point dart, although only a few are required for high quality.*

bounding box, we use equations from Section 5.1 to transform triangle edges to the u, v domain.

From here on, we follow Gribel et al. [2010]'s approach to construct and resolve line darts, and direct the interested reader to their paper for more details. For each line sample, we analytically compute any portion covered by the triangle. A per-line-sample queue stores covered portions with color as well as depth information. Once all triangles have been processed, we resolve the final color for each sample by sweeping across its dimension while aggregating triangles closest in depth, and then using all pixel samples to compute the final color for the pixel.

5.3 Implementation and Results

To test our formulation, we designed a simple CPU-based renderer capable of rendering triangle scenes using traditional triangle rasterization. To this renderer, we integrated the following:

- A stochastic sampler based on point-darts
- A sampler based on line-darts

Resulting images for a basic scene are shown for quality comparison in Figure 7. Also, Table 1 shows the performance of our samplers.

Clearly, the renderer based on k-d darts is virtually free of noise with just two darts in u, v space. However, dart orientations introduce some obvious bias, which quickly resolves as we use more than 12 line darts. In contrast, a stochastic point sampler suffers from severe noise even with 64 samples per pixel. At 169 samples per pixel, the noise is reduced but still evident. Furthermore, at this point the stochastic renderer is 4–5 times slower than the k-d dart renderer with 12–16 line darts. Note that the slight difference in the blur magnitudes is due to our choice of a wagon-wheel line dart pattern (Figure 6).

Clearly, throwing a line dart is much more expensive than throwing a point dart. However, line darts improve bias significantly faster and are less noisier even with a few samples per pixel.

6 Probability of Failure, Uncertainty Quantification (UQ)

Uncertainty quantification (UQ), in this context, samples the result of a complex high-dimension computation that characterizes a particular system to estimate the probability of that system's failure. In typical UQ application domains, the function is complex, sampling the function is expensive, and the failure region is small compared to the size of the domain, so efficient sampling is crucial. K-d darts can make this computation more efficient. This application demonstrates that k-d darts are applicable even for complex functions.

6.1 UQ: Error Analysis

Let $I(\underline{x})$ be the indicator function for a system failure, such that $I(\underline{x}) = 1$ if failure occurs at the point $\underline{x}$ and zero otherwise. Then the probability, or mean rate of occurrence, of failure is $\mathrm{E}(I(\underline{x})) = P$, where $\mathrm{E}()$ is the "expectation" or mean over the domain $\underline{x} \in \mathcal{X}$. If the probability that a system failure occurs using point-dart MC is P, then the true variance of $I(\underline{x})$ is

$$\begin{aligned} \sigma^2 &= \mathrm{E}\left((I(\underline{x}) - P)^2\right) \\ &= \mathrm{E}\left(I(\underline{x})^2\right) - 2\mathrm{E}\left(I(\underline{x})\right)P - P^2 \\ &= \mathrm{E}\left(I(\underline{x})^2\right) - P^2 = P - P^2. \end{aligned} \tag{4}$$

Thus the point-dart MC standard error is

$$\sigma_{\mathrm{err_{PD}}} = \sqrt{\frac{P - P^2}{N}}. \tag{5}$$

Eq. 1 still applies when hyperdarts are used instead of point-darts, but the definition of the σ in the numerator and N in the denominator changes. The analytical variance σ^2 is now defined as the average over all $\binom{d}{k}$ components or flats of a d-dimensional hyperdart. The N in the denominator is replaced with $\binom{d}{k}N$ where N is now defined to be the number of hyperdarts. Also, because the Central Limit Theorem now requires all *components* of samples to be independent, *split* hyperdarts must be used. Even with these stipulations, the standard error for a d-dimensional hyperdart, $\sigma_{\mathrm{err_{kD}}}$, depends on the problem under consideration, This is because the indicator function is no longer a binary function; it can now take on any real value between 0 and 1 (inclusive). This can significantly reduce the analytical variance, σ^2. The value of $I(\underline{x})$ for a specific hyperdart is the fraction of it that intersects with the failure region, and that depends on the *shape* of the failure region. If the failure region was shaped so that P of every component of every possible k-hyperdart intersected with the failure region, then the standard error would be zero, but other than for the special cases of $k = d$, $P = 1$ or $P = 0$ this is highly unlikely. However, if one assumes the domain $\mathcal{X}$ is a hypercube, it is straightforward to compute the standard error for certain extreme cases. Intuitively, 3 extreme cases should be considered:

1. a hyper-rectangle failure region that covers P of 1 dimension and all of the $d - 1$ other dimensions,
2. a hypercube failure region that covers $P^{1/d}$ of all d dimensions, and
3. a hypersphere failure region.

Only the first two are considered here because they are far easier to compute than the third. For extreme case 1, $\binom{d-1}{k-1}$ components of the hyperdart have $\sigma^2 = 0$; the other $\binom{d-1}{k}$ have the same σ^2 as point-dart MC. This averages to

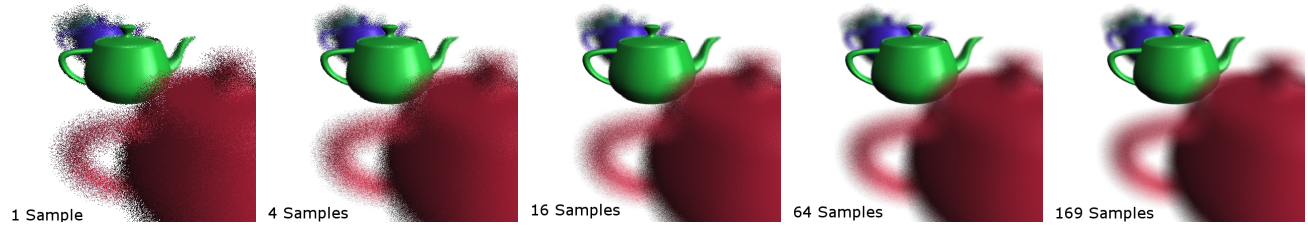
$$\sigma^2_{\mathrm{HD1}} = \frac{\binom{d-1}{k}}{\binom{d}{k}}\left(P - P^2\right) = \frac{d-k}{d}\left(P - P^2\right). \tag{6}$$

The standard error is then

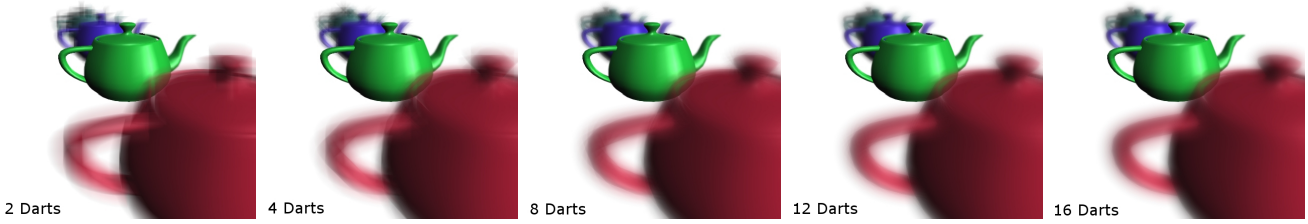
$$\sigma_{\mathrm{err_{HD1}}} = \sqrt{\frac{(d-k)(P - P^2)}{d\binom{d}{k}N}}. \tag{7}$$

For extreme case 2, the analytical variance is the same in all directions:

$$\sigma^2_{\mathrm{HD2}} = P^{1-k/d}\left(P^{k/d}\right)^2 - P^2 = P^{1+k/d} - P^2. \tag{8}$$



(a) Images rendered using our point sampling renderer.



(b) Images rendered using our renderer based on line darts.

Figure 7: *Depth-of-field results using conventional point sampling vs our k-d darts method. Our k-d darts method is able to produce a high-quality, relatively noise-free image using only a few line-samples, at the cost of some bias.*

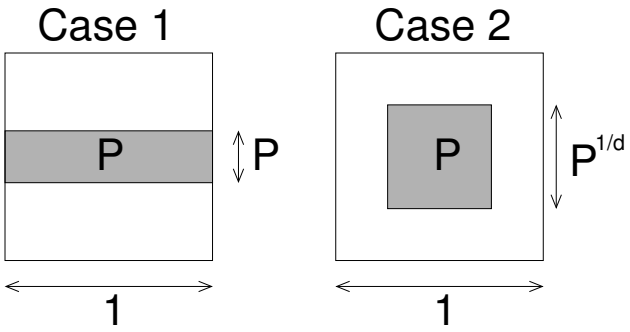


Figure 8: *Left: Extreme Case 1: the failure region covers P of one dimension and all of $d-1$ dimensions. Right: Extreme Case 2: $P^{1/d}$ of all d dimensions are covered.*

The standard error is then

$$\sigma^2_{\text{err}_{\text{HD2}}} = \sqrt{\frac{P^{1+k/d} - P^2}{\binom{d}{k} N}}. \tag{9}$$

Thus, theoretically, a single d-dimensional hyperdart can exactly compute the probability of failure. This is reflected by the standard error equaling zero for both extreme case 1 (Eq. 7) and 2 (Eq. 9) when $k = d$. For $0 < k < d$, under the assumption that the cost of each flat is paid separately, the primary advantage of hyperdart MC is the reduction of the analytical variance in the numerator of the standard error. However, this significant increase in accuracy is associated with an increased per-dart cost, where cost is defined as the number of potentially very computationally expensive simulations. Note that a shared cost implementation, one in which the simulations performed for earlier flats are leveraged to reduce the number of simulations required by later flats would result in a significant cost reduction. This would allow hyperdart MC to *benefit* from an increased number of dimensions, which is why k-d darts are potentially powerful tools for uncertainty quantification.

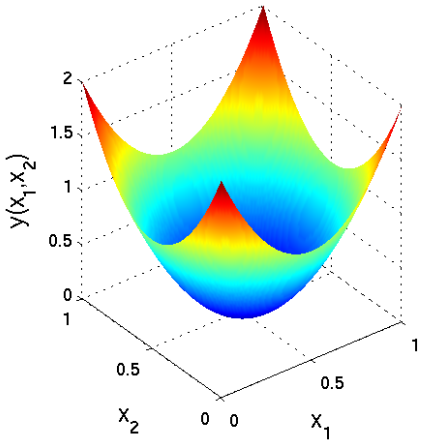


Figure 9: *Plot of the "circular parabola" function, Eq. 10, for two inputs.*

6.2 UQ: Results

We now demonstrate the potential of k-d darts for uncertainty quantification, by using line-dart Monte Carlo to determine the failure probability for the simple $d = 2$ dimensional "circular parabola" test function plotted in Figure 9. Its governing equation is

$$y(\underline{x}) = \sum_{i=1}^{d} (2x_i - 1)^2, \; 0 < x_i < 1. \tag{10}$$

Our MATLAB implementation of line-darts makes the following assumptions:

- Failure is defined as the output of a true function (read as "simulator") $y = f(\underline{x})$ being either above or below a threshold value.

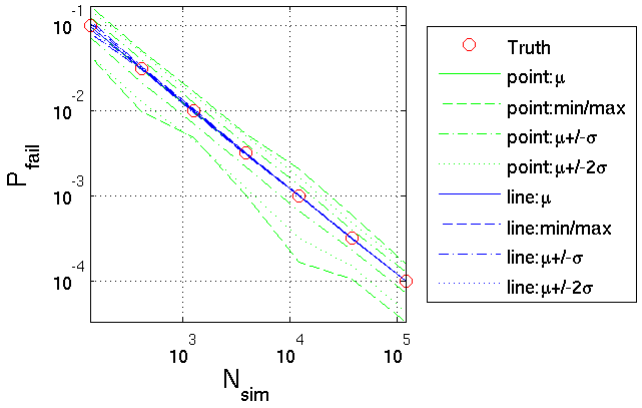


Figure 10: *Comparison of point-dart and line-dart Monte Carlo for the 2D circular parabola problem. The number of line-darts thrown was* $\text{ceil}(2/P_{fail})$*. Point-dart MC used the same number of simulations as line-dart MC. In the plots* μ *and* σ *are the mean and standard deviation respectively of 100 trials. The true failure probabilities were specified directly and the corresponding thresholds were calculated from the area of the circular failure regions. The failure probabilities calculated by line-dart MC are dramatically closer to the true values than those calculated by point-dart MC.*

- $f(\underline{x})$ is C^1 continuous everywhere on the domain $\underline{x} \in \mathcal{X}$.
- The simulator is a computationally expensive black box function, whose first derivative of $f(\underline{x})$ can be efficiently calculated using a numerical technique such as continuous adjoints or automatic differentiation [Griewank 1988]. Here, "efficiently" means the cost of evaluating a the true function plus its first derivative is comparable to the cost of evaluating only the function. "Expensive" means that all other computational costs are negligible compared to the cost of running simulations.

The implementation handles each component or flat of a split line-dart separately and uses a 1D cubic Hermite spline a response surface for each component. The 1D optimization starts by simulating at both ends of the one-dimensional domain and fitting a single-segment cubic Hermite spline. The spline is rebuilt with an increasing number of segments after each simulation is added. If a segment of the spline intersects with the threshold then a simulation is performed at that intersection. If the spline segment does not intersect the threshold, then the point on the spline closest to the threshold is evaluated. If there are multiple equally good candidates on a single segment, then the one closest to the segment's center is evaluated. A segment's best candidate will only be evaluated if it is between the segment's break points (not inclusive). The 1D optimization stops when no segment proposes a valid candidate.

If the projection of the true function onto the flat is cubic or lower order polynomial, such as for the circular parabola problem, then this implementation uses the minimum number of simulations possible within the class of methods that pays the cost of each flat separately. Here minimum means 4 simulations per flat that intersects the failure region twice and 3 simulations per flat that intersects it one or zero times.

Figure 10 shows that the failure probabilities calculated by line-dart MC are dramatically more accurate (closer to the true values) than those calculated by point-dart MC for the 2D circular parabola test function when the same number of simulations are used for each approach. Specifically, the minimum, maximum, and mean plus or minus two standard deviations for 100 trials of line-dart MC were well within the mean plus or minus one standard deviation for 100 trials of point-dart Monte Carlo. This demonstrates the potential for significantly greater accuracy of hyperdart relative to point-dart MC. However, hyperdart MC will only be competitive with point-dart MC for generic high-dimensional problems if the simulation cost of flats can be shared, i.e. if the cost of later flats can be reduced by taking advantage of the simulations performed for earlier flats. It may be possible to accomplish this with a global-surrogate-based optimization method that concurrently locates the intersection of all flats with the failure region's isosurface.

7 Conclusions

In this work we generalize higher-dimensional sampling, introduce the k-d dart, and demonstrate its effectiveness in a number of applications.

Line darts seems to be quite significant for solving the maximal Poisson-disk sampling problems in higher dimensions. Below a given acceptable void ratio V, they are clearly more efficient than point darts. To achieve the same relaxed maximal condition (e.g the distance between any point in the domain is less than $r + \epsilon$) independent of the number dimensions, d, the acceptable void ratio has to decrease exponentially with d. This makes line darts even more significant as the number of dimensions increases. Our results suggests that even applying higher dimensional-darts might be useful in some cases. The bias introduced by the line darts seems to be negligible. Our line-dart algorithm has is optimal with regard to the memory requirements enabling the production of large samples in higher dimensions. In one case we were able to produce a point cloud with 43527 points, $d = 30$ in less than three hours.

In the context of uncertainty quantification, hyperdart Monte Carlo sampling has the potential for dramatically greater accuracy per cost than point-dart Monte Carlo. A optimization/root-finding method that leverages earlier simulations to reduce the computational cost of later flats is needed to realize this potential for general high-dimensional problems.

For depth-of-field, using k-d darts is extremely beneficial in reducing noise. Though each 1D dart requires more processing than point samples, we only need a few to render a high-quality image. This results in our k-d dart method outperforming point sampling. We expect that a parallel GPU implementation of this method would get real-time performance, which we plan for future work. Since k-d darts result in a low-noise image with bias, and point sampling gives noisy bias-free images, a hybrid sampling implementation may mitigate artifacts from both of these sources.

K-d darts are an example of higher dimensional hyper-plane sampling; however, there are a variety of other higher-dimensional sampling techniques, such as sampling along the path of a circle (or hyper-sphere) with varying radii, or sampling along a z-pattern. We plan to explore some of these possible sampling techniques for graphics applications to study what advantages, if any, we can exploit from their characteristics.

References

CHIU, K., SHIRLEY, P., AND WANG, C. 1994. Multi-jittered sampling. *Academic Press Graphics Gems Series*, 370–374.

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

DALBEY, K. R., AND KARYSTINOS, G. N. 2010. Fast generation of space-filling Latin Hypercube Sample designs. In *Proceed-*

ings of the 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA 2010–9085.

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

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

EBEIDA, M. S., PATNEY, A., MITCHELL, S. A., DAVIDSON, A., KNUPP, P. M., AND OWENS, J. D. 2011. Efficient maximal Poisson-disk sampling. *ACM Transactions on Graphics 30*, 4 (July), 49:1–49:12.

EBEIDA, M. S., MITCHELL, S. A., PATNEY, A., DAVIDSON, A. A., AND OWENS, J. D. 2012. A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Computer Graphics Forum, Proc. Eurographics 31*, 2, tbd.

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

GILKS, W., RICHARDSON, S., AND SPIEGELHALTER, D. 1996. *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

GLYNN, P., AND IGLEHART, D. 1989. Importance sampling for stochastic simulations. *Management Science*, 1367–1392.

GRIBEL, C. J., DOGGETT, M., AND AKENINE-MÖLLER, T. 2010. Analytical Motion Blur Rasterization with Compression. In *High-Performance Graphics*, 163–172.

GRIBEL, C. J., BARRINGER, R., AND AKENINE-MÖLLER, T. 2011. High-Quality Spatio-Temporal Rendering using Semi-Analytical Visibility. *ACM Transactions on Graphics 30* (August), 54:1–54:11.

GRIEWANK, A. 1988. On automatic differentiation. Tech. rep., Argonne National Lab., IL (USA).

HAEBERLI, P. E., AND AKELEY, K. 1990. The accumulation buffer: Hardware support for high-quality rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, 309–318.

HALL, A. 1873. On an experimental determination of pi. *Messenger of Mathematics 2*, 113–114.

HAMMON JR. , E. 2008. Practical post-process depth of field. In *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 583–605.

JONES, T. R., AND KARGER, D. R. 2011. Linear-time Poisson-disk patterns. *Journal of Graphics, GPU, & Game Tools* (to appear). arXiv:1107.3013v1.

JONES, T. R., AND PERRY, R. N. 2000. Anti-aliasing with line samples. Technical report, MERL – A Mitsubishi Electric Research Company, June.

LIU, B., AND STANLEY, T. 1965. Error bounds for jittered sampling. *IEEE Transactions on Automatic Control 10*, 4, 449–454.

MCKAY, M., BECKMAN, R., AND CONOVER, W. 1979. Comparison the three methods for selecting values of input variable in the analysis of output from a computer code. *Technometrics 21*, 2 (May), 239–245.

NEBE, G., AND SLOANE, N., 2012. A catalogue of lattices. `http://www.math.rwth-aachen.de/~Gabriele.Nebe/LATTICES/index.html`.

OWEN, A. 1992. A Central Limit Theorem for Latin Hypercube Sampling. *Journal of the Royal Statistical Society. Series B (Methodological) 54*, 2, 541–551.

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