Optimizing Kronecker Sequences for Multidimensional Sampling

Mayur Patel
Prodigy Education

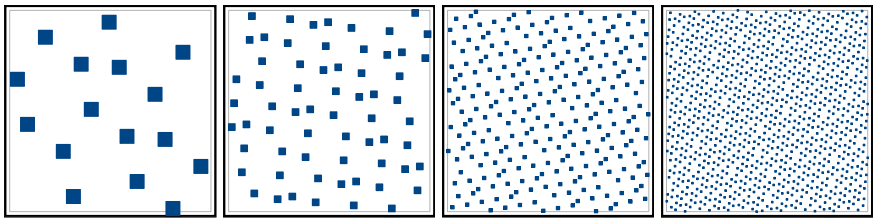


Figure 1. From left to right, we show 16, 64, 256, and 1024 points generated in the unit square using the two-dimensional Kronecker sequence defined by these irrational values: $\frac{\sqrt{506598872547596}}{29147227}$, $\frac{\sqrt{107882942223468}}{28993644}$. The subject of this paper is the method by which we found this and other useful Kronecker sequences in dimensions greater than one.

Abstract

We review the use of Kronecker sequences for sampling applications. To apply them to multiple dimensions, we develop and execute a method to find irrational numbers that will produce good-quality results. Finally, we provide empirical evidence that the irrationals we found outperform those in current use and that they perform respectably against other sample generation techniques.

1. Introduction

Sampling is a fundamental operation in computer graphics. The rapid generation of high-quality sample points remains a topic of active interest. Among other uses, samples drive the integration methods that have become standard in the synthesis of imagery. There are a number of properties that we believe make a sample generator useful.

 ISSN 2331-7418

- We want to generate samples rapidly.
- We want the code for generating samples to be simple. Bugs that find their way into sample generation may be difficult to detect because there may be no immediate error in the resulting program execution. However, by having simple implementations, we reduce the probability of such bugs occurring.
- We want the sample generator to be scalable to higher dimensions with a minimal increase in the implementation complexity.
- We prefer sample generators that are progressive. In other words, it is useful that the generator does not need to know how many samples will be generated before sampling begins.
- Of course, we want the samples to be of high quality.

Stratification is very popular in computer graphics for improving the quality of point sets generated randomly for sampling. The approach involves subdividing the sample space uniformly and distributing samples evenly across these areas. Frequently, one sample is randomly placed within each subdivision. Jittered sampling has long been a popular technique, presumably because it offers a good compromise between ease of implementation and quality [Shirley 1991]. More advanced stratification methods have been developed for producing sets of high-quality samples with a corresponding increase in the complexity of the implementation [Kensler 2013; Christensen et al. 2018].

A number of practical methods have emerged for calculating point sets according to the Poisson-disc distribution. Typically, these require more memory and more complex implementations than other methods due to the use of spatial data structures [Dunbar and Humphreys 2006; Bridson 2007].

Tile-based techniques involve dividing the sample space into perfectly packed tiles and associating sample placements with these tiles. Primitives that define tiles are saved and later fit together to produce larger collections of samples. Effort is invested to minimize artifacts where the tiles meet [Kopf et al. 2006; Ostromoukhov 2007; Ahmed et al. 2017].

Sobol sequences and Halton sequences are specific instances of digital nets. Sample generation involves dividing the interval from zero to one in increasingly finer steps. Such techniques have enjoyed success in sampling applications, but they require careful selection of parameters for generating high-dimensional sequences.

Lattice generators space samples with regularity in the unit torus: for example, using an additive recurrence. Recent work has shown that rank-1 lattices can be extended to produce samples in high dimensions with a reasonable investment of effort [Liu et al. 2021].

Techniques that generate finite point sets tend to result in better sampling than techniques based on infinite sequences [Dick et al. 2013]. This is because the points in the set can be optimized relative to one another to guarantee coverage of the space. However, progressive sample generators are becoming increasingly important in modern computer graphics work flows where renders may use adaptive sampling or progressive refinement to improve interactivity for the user.

Sampling is a vast topic, and there are useful surveys that cover this material in greater detail [Keller et al. 2012; Keller 2013; Dick et al. 2013; Keller et al. 2019].

In this paper, we review a lattice generator that has received little attention in computer graphics relative to other methods: Kronecker sequences (Figure 1). These sequences use one irrational value per dimension to produce uniformly distributed values. They can be implemented with a few lines of code using a compact state even for high dimensions. This code can run at extreme speeds on contemporary hardware. Our main contribution is to present lookup tables of irrational numbers that are suitable for driving Kronecker sequences of various dimensions, and we discuss the means by which we found them. We demonstrate empirically that the quality of samples generated with these methods is comparable to other techniques.

2. Kronecker Sequences

Weyl's Equidistribution Theorem demonstrates that irrational numbers can be used to produce values that are uniformly distributed. This result is well known in number theory [Kuipers and Niederreiter 1974]. The sequence can be defined by an additive recurrence:

$$S[0] = k_0, \quad S[i] = \{S[i-1] + \alpha\}, \tag{1}$$

where $S[i]$ is the value of the 1D sequence at index i and α is the irrational number that defines the sequence. The expression $\{x\}$ indicates the fractional part of x, or x modulo one. The constant k_0 is a Cranley–Patterson rotation [Cranley and Patterson 1976] that allows for variation between instances of the sequence. This sequence generates values that are uniformly distributed in the range $[0, 1]$. Such sequences are known as *Kronecker sequences*.

Kronecker sequences in s dimensions can be calculated by using s linearly independent irrational values and calculating each dimension independently using the given formula.

On initial inspection, Kronecker sequences seem to have the properties that we seek in a sample generator. Samples can be generated rapidly on contemporary hardware, especially when SIMD instructions are available. Multidimensional implementations are compact and easy to code. Samples may be generated progressively.

For all these advantages, Kronecker sequences have one big caveat. Kronecker sequences are uniformly distributed in the limit, but short subsequences are not always well-distributed over the full volume of the unit hypercube.

Unfortunately, there is no theory that currently exists to help us select irrational values that will produce optimal Kronecker sequences in dimensions greater than one [Kuipers and Niederreiter 1974; Niederreiter and Winterhof 2015]. However, there have been attempts to develop theory that would help us choose useful values, even if they are not optimal. Authors have suggested that powers of e using rational exponents would be good choices [Zinterhof 1996]. Transcendental numbers of other forms have also been suggested [Zhu 2007]. Metallic irrationals, a generalization of the golden ratio, have also been recommended [Chi 2013]. In practice, these sets are still vast and it is difficult to find specific values that perform well in real-world applications. However, one specific generalization of the golden ratio resulted in concrete values that exceeded the quality of previous efforts [Roberts 2018b]. We will use those results as a basis of evaluating our findings later in this paper.

Optimal 1D Kronecker sequences have been extended to multidimensional applications by using permutations or space-filling curves to map the optimal 1D sequence to higher dimensions [Schretter et al. 2012; Schretter et al. 2016]. These methods sacrifice some important advantages of Kronecker sequences, such as performance and simplicity.

In this paper, we propose an empirical method for finding sets of irrational values that produce good-quality Kronecker sequences. We measure the quality of the results to demonstrate the utility of the method.

We make no claim that the collections of irrational values we present will produce *optimal* Kronecker sequences, but we believe that the values we propose will perform well when used for real-world computer graphics applications, especially when compact and high-speed sample generation is of special interest.

3. Finding Irrational Numbers for Kronecker Sequences

In this section, we present a method for finding irrational values that will produce high-quality Kronecker sequences. Our method consists of three elements. First, we create a metric suitable for estimating the uniformity of a point set. Second, we demonstrate a method for aggregating the metric for a sequence that is used to produce point sets of many different sizes. Finally, we describe an iterative refinement method to improve the aggregate metric value.

To begin, we define a metric to estimate the uniformity of a point set. Many have observed that maximizing the minimum distance between samples has the effect of increasing the quality of those samples [Keller 2006; Schlömer et al. 2011; Keller et al. 2019]; in fact, this is a driving principle in many sample generators based on the Poisson-disc distribution. We exploit this observation to develop a simple test that can be used to calculate a quality metric for a set of irrational numbers that drive a Kronecker sequence.

Let's assume that we have a set of points P in the unit hypercube. The set P has N points. Furthermore, let's assume we have a region of interest R_M within the unit hypercube, where the expected number of samples from P within the region for a uniformly distributed point set is M. If $G_e(R_M, M)$ is the ideal Euclidean distance between two points in R_M and $G_m(R_M, M, P)$ is the closest measured Euclidean distance between any two points in R_M, then the metric is given by

$$J(R_M, M, P) = \left(\frac{G_e(R_M, M) - G_m(R_M, M, P)}{G_e(R_M, M)} \right)^2 . \tag{2}$$

When estimating the ideal expected distance between points, $G_e(R_M, M)$, we draw upon existing work for the hypersphere packing problem, imagining that one of our points lies in the center of each hypersphere. The challenge is to pack uniformly sized hyperspheres optimally into Euclidean space without overlap using a regular lattice structure, minimizing the volume of the space outside any hypersphere. In dimensions 2–8, the optimal solution is known [Weisstein 2021]. In other dimensions, we approximate a solution by centering hyperspheres at the corners of a hypercube lattice. The fraction of the volume of the space that is enclosed by hyperspheres is called the packing density, given by δ_s for dimension s. If M is the number of points expected in the region of interest R_M, V_{R_M} is the volume of the region of interest, and the function $Q_s(v)$ gives the radius of a hypersphere of dimension s given its volume v, then the ideal expected distance is given by

$$G_e(R_M, M) = 2 \cdot Q_s \left(\frac{\delta_s \cdot V_{R_M}}{M} \right) . \tag{3}$$

We use a region of interest in our calculation of $J(R_M, M, P)$. Without a region of interest, the time complexity of Equation (2) is $O(NlgN)$ owing to the complexity of the closest distance algorithm. In cases where the region of interest is used, then the complexity becomes $O(MlgM) + O(N)$. If we use a small region of interest, where M is much less than N, then the complexity becomes linear with N. We will exploit this later as an important optimization.

Sets of irrationals that drive Kronecker sequences that produce lower metric values are more desirable. As presented, Equation (2) only determines the suitability of the irrational set for a specific number of samples. For general applications, we want sets of irrationals that produce good metric values when producing any number of samples.

We use the metric from Equation (2) to evaluate a set of irrationals over a number of different sample counts in Algorithm 1.

We establish the maximum number of samples to search on line 1 and the minimum on line 2. For our experiments, we used 2^s as the lower bound, which is a typical lower bound for sampling s dimensions.

Algorithm 1 Metric for a sequence of s dimensions.

1: $N_1 \leftarrow$ maximum number of samples to test
2: $N_0 \leftarrow$ minimum number of samples to test, 2^s
3: $n \leftarrow$ Fibonacci sequence (start at smallest value >= N_0)
4: $KS \leftarrow$ Kronecker sequence (s irrationals)
5: $m \leftarrow 0$
6: **while** n::value() <= N_1 **do**
7: $M \leftarrow$ min(n::value(), 2^{2s})
8: $R_M \leftarrow$ region expected to contain M samples
9: $r \leftarrow J(R_M, M, KS$ ::make samples(n::value()$)$
10: $m \leftarrow m + r$
11: n::advance to next value()
12: **end while**
13: **return** m

On lines 3, 6, and 11, we use a Fibonacci sequence to generate the exact sample count to test for each iteration. We found that the irregularity between adjacent values in the sequence, particularly at low values, helped to prevent our search from favoring Kronecker sequences whose quality of uniformity oscillated strongly at a particular frequency relative to the number of samples generated.

On lines 7 and 8, we calculate a region of interest where we expect to find a potentially small fraction of the total number of samples generated. This is an important optimization that makes the entire search feasible on modern hardware. The performance bottleneck changes from being the closest distance calculation to being sample generation itself.

In our implementation, we used a hypercube centered within the unit hypercube as our region of interest. On line 9, we collect the metric from Equation (2), where M is the expected number of samples in the test. In our implementation, we seek to test 2^{2s} samples, which gives us some margin above the minimum sample count required to sample the dimension in question. In Figure 2, we show the regions of interest for various sample counts in two dimensions. Because we are using a relative error calculation, when we aggregate the metric values in Algorithm 1 on line 10, we have a strong expectation that the metric values will fall within the same order of magnitude. This means that the results of every iteration are weighted approximately equally throughout the test.

The idea of using a region of interest is inspired by the observation that the relative positions of points from a Kronecker sequence repeat in different areas within the unit hypercube. Testing all the samples would result in a high degree of redundant calculations in most situations. Because we are using the Fibonacci sequence to generate sample counts, we know that there are approximately 68% more samples

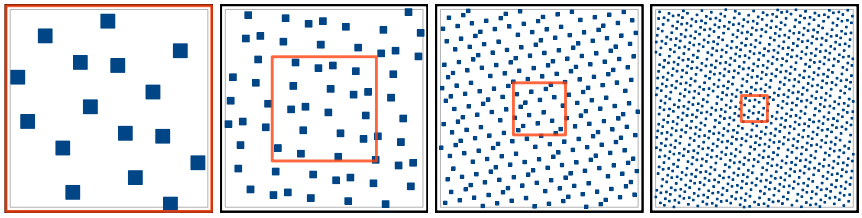


Figure 2. From left to right, we show sets of 16, 64, 256, and 1024 samples and the regions of interest our algorithm uses to estimate their quality.

added between iterations of the test. This adds samples relatively slowly to the test set, so we can have some confidence that we are testing at a frequency sufficient to catch significant changes in the configuration of the relative positions of samples in the sequence even in a small region of interest.

We aggregate metrics from different iterations on lines 5, 10, and 13. In effect, this gives us the root mean square relative error (RMSRE) of the closest distances between samples in the sequence for all the iterations in the execution. In practice, it is convenient for us to skip the last division and square root operation in the RMSRE calculation. This is possible because metrics are only compared when they have been executed on the same sets of samples, and because metric values are only used for comparison. Our omission of operations does not change the rank order of the metrics. Use of the root mean square is known to favor consistently good estimates, punishing tests that show strong outliers in the observations.

As presented, Algorithm 1 will evaluate the quality of a Kronecker sequence driven by a specific set of irrationals, but we still need to provide the irrational values themselves for the test. We start with random values. As we iterate, we add very small random values to the irrationals and we test the new values to discover improvements. The addition of small random amounts to the irrationals will be referred to as a *value adjustment*. The process is presented in Algorithm 2.

The biggest difference between the two algorithms are on lines 11–19 of Algorithm 2. At each iteration of the search, we generate a selection of alternatives to the original Kronecker sequence, which are random variations of the original irrational values. Each new alternative Kronecker sequence is tested, and if it performs better than the current one with the sample counts encountered so far, then the current one is replaced. In our experiments, we tested value adjustments of the irrationals of a sequence four times in each dimension. This is demonstrated on lines 11 and 12.

It may not be obvious how one selects initial irrational values at random or how one applies value adjustments to irrational values. Quadratic irrationals have good theoretical properties for 1D Kronecker sequences [Niederreiter and Winterhof 2015]; so, like others [Chen et al. 2003], we used them in our search for multidimensional sequences. On line 4, we create a new Kronecker sequence by selecting random

Algorithm 2 Search for s-dimensional Kronecker sequences.

$N_1 \leftarrow$ maximum number of samples to test
$N_0 \leftarrow$ minimum number of samples to test, 2^s
$n \leftarrow$ Fibonacci sequence (start at smallest value > N_0)
$KS \leftarrow$ Kronecker sequence (s irrationals)
$m \leftarrow 0$
while n::value() <= N_1 **do**
  $M \leftarrow$ min(n::value(), 2^{2s})
  $R_M \leftarrow$ area expected to contain M samples
  $r \leftarrow J(R_M, M, KS$ ::make samples(n::value()$)$
  $m \leftarrow m + r$
  for $i = 0$ to $4s$ **do**
    $C \leftarrow KS$::value adjust irrational values(i)
    $m_C \leftarrow$ **Algorithm 1** (C, n::value())
    if $m_C < m$ **then**
      $KS \leftarrow C$
      $m \leftarrow m_C$
      $i \leftarrow 0$
    end if
  end for
  n::advance to next value()
end while
return KS

fractional values and fitting quadratic irrationals to them, of the form:

$$\alpha = \frac{\sqrt{j}}{k}, \quad \sqrt{j} < k, \quad j, k \in \mathbb{Z}^+. \tag{4}$$

In our implementation, k can be stored in approximately 25 bits. We use an integer j of approximately 50 bits, which is convenient on modern hardware given the representation of double-precision floating-point values and the widespread use of 64-bit processors. Of course, one need not preserve the full precision of these values in practice. Using shorter bit-words, the maximum length of the sequence that can be generated will be affected proportionally.

We execute value adjustments similarly. The floating-point value of the irrational is offset by a random amount proportional to the expected distance between sample points at the current sample density. Then, we generate a new quadratic irrational to approximate this value.

Though we could have only calculated floating-point values that approximate irrational values, we believe that there will eventually be utility in fitting these quantities

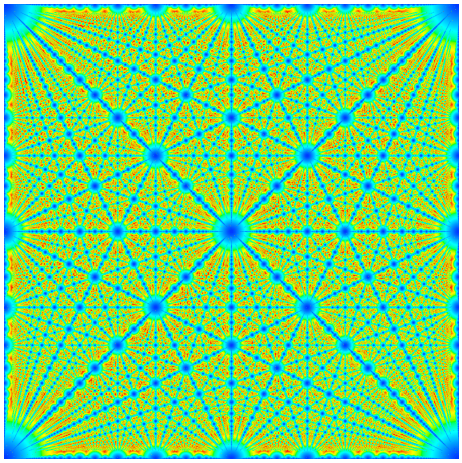


Figure 3. Heat-map visualization showing the quality of Kronecker sequences driven by irrationals in the unit square. Hot colors, especially red, indicate coordinates of desirable pairs of irrationals, as indicated by our metrics and methods.

to true irrational values. The theoretical benefit remains to be proven. For the average engineer, there will be no difference in how these values are ultimately used in practice.

We recognize that our technique is one of local refinement and not necessarily one of true exhaustive search. In order to explore diverse irrational sets in the hypercube, we run Algorithm 2 a very large number of times with distinct initial irrational values for each execution. The appendix reports the irrationals that produce the Kronecker sequences with the best metrics among all executions. Because each invocation of Algorithm 2 can be run independently, this provides a very good opportunity for massively parallel execution with fairly modest memory requirements.

In Figure 3, we present a heat-map visualization of the unit square, showing where desirable irrational pairs can be found in two dimensions.

Clearly, there is a regular pattern underlying the data in the figure. This makes us optimistist that a theory may some day be developed to help us choose optimal irrational sets analytically. We acknowledge that irregularities or inaccuracies in the image could represent flaws in our methods.

4. Uniformity Metrics

4.1. Comparing Kronecker Sequences

We used the methods described in the previous section to discover sets of irrational numbers whose specific values are given in the appendix. In this section, we will compare the Kronecker sequences produced from these irrationals (K21-2, K21-3, K21-4) to the Kronecker sequences from Roberts in dimensions 2–4 (R2, R3, R4)

[Roberts 2018b]. In computer graphics, there is often the need to sample 2D and 3D space, as well as 3D space+time, so these dimensions are of the greatest interest to computer graphics practitioners.

We use the star discrepancy and arbitrary edge discrepancy [Dobkin and Mitchell 1993] as metrics of quality in two dimensions. In all dimensions, we report the diaphony and the closest Euclidean distance between points in each set.

Diaphony is a measure of uniformity of point distribution closely related to discrepancy [Hellekalek and Niederreiter 1998]. The diaphony of N samples in s dimensions can be calculated with $O(s \cdot N^2)$ time complexity. Because the diaphony calculation scales better than the discrepancy metrics to both higher dimensions and larger point sets [Doerr et al. 2014; Burkardt 2012], we use the diaphony in these situations to compare Kronecker sequences.

The closest Euclidean distance is interesting to observe because the optimizations described in the previous section are driven by distance. We acknowledge that the Euclidean distance becomes less useful in high dimensions due to the Curse of Dimensionality.

In Figure 4, we graph the results for Kronecker sequences in two dimensions. In the following table, we analyze those results over different sample-count ranges.

	Samples	Star	Arb. Edge	Diaphony	Distance
K21-2 better than R2	$[2, 256]$	56%	63%	39%	36%
	$[257, 512]$	95%	29%	100%	18%
	$[513, 1024]$	90%	66%	100%	92%
	$[1025, 2048]$	52%	52%	68%	18%

Interestingly, the K21-2 sequence did not frequently beat the R2 sequence with respect to the closest distance metric. For the other metrics, the K21-2 sequence was modestly better.

Figure 5 graphs the diaphony and closest distances for the Kronecker sequences in three dimensions; Figure 6 graphs them for 4D sequences. The results are not definitive.

	Samples	Diaphony	Distance
K21-3 better than R3	$[2, 256]$	9%	92%
	$[257, 512]$	20%	100%
	$[513, 1024]$	100%	41%
	$[1025, 2048]$	97%	36%
K21-4 better than R4	$[2, 256]$	8%	73%
	$[257, 512]$	38 %	100%
	$[513, 1024]$	100%	75%
	$[1025, 2048]$	100%	100%

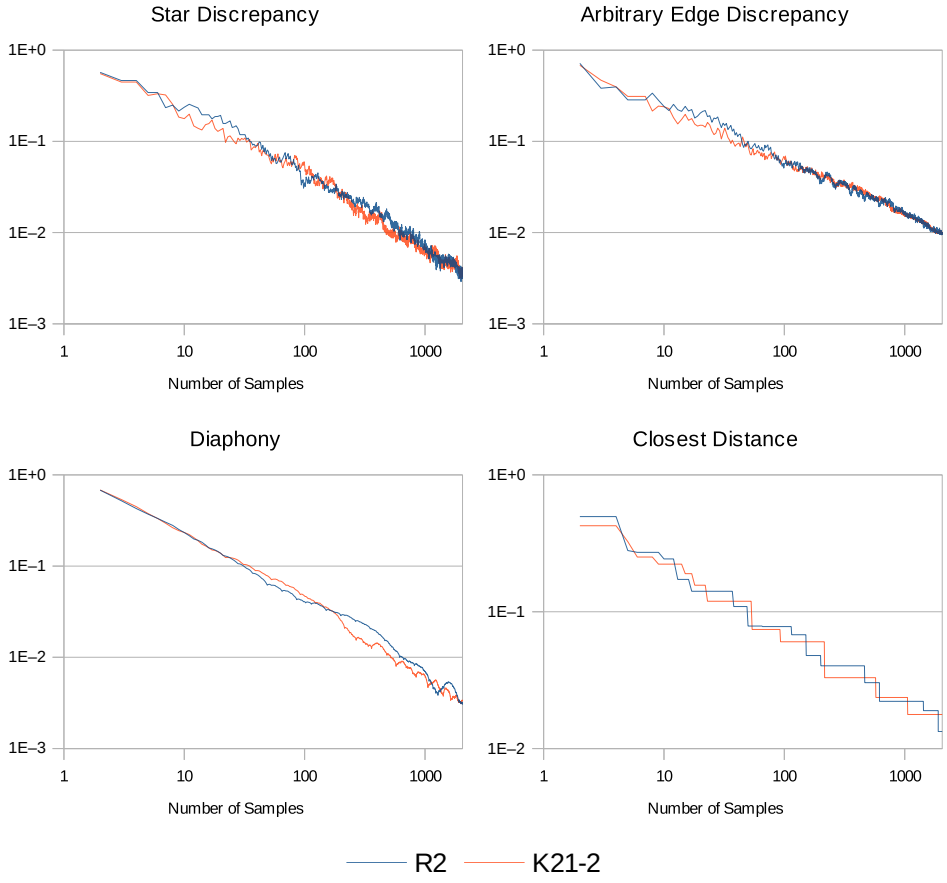


Figure 4. Various quality metrics for the K21-2 and R2 Kronecker sequences. The axes of the charts are logarithmic, base 10.

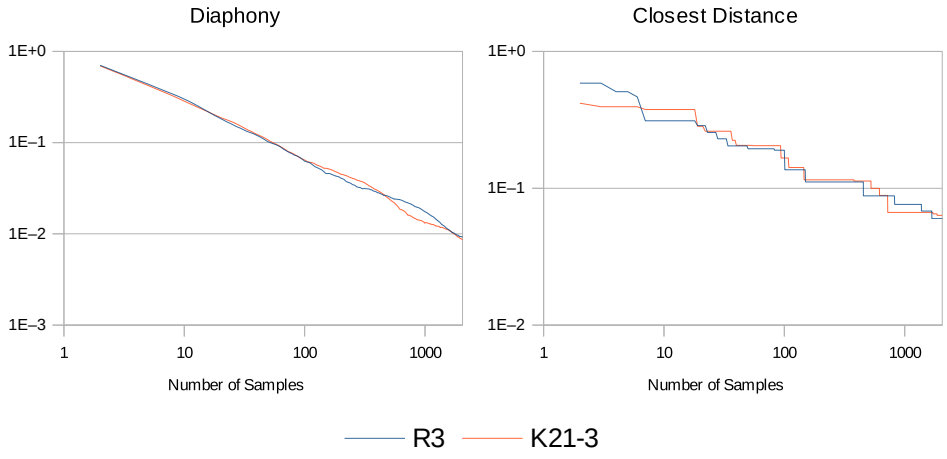


Figure 5. Quality metrics of Kronecker sequences in three dimensions.

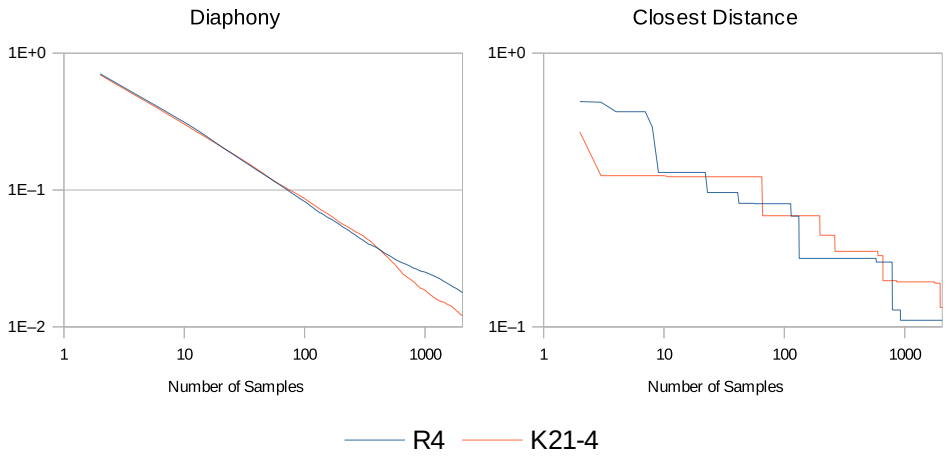


Figure 6. Quality metrics of Kronecker sequences in four dimensions.

4.2. Comparing against Other Generators

In this section, we compare the metrics of uniformity for the K21-2 Kronecker sequence against those of other progressive sample generators in two dimensions.

In our comparison, we use Grünschloß's implementation of the Halton23 sequence [Grünschloß 2021] and Burley's implementation of the Sobol sequence with Owen scrambling [Burley 2020] to represent good digital nets. These implementations are also known to be fast, so they will be useful during performance testing as well. We use the PMJ02 generator [Christensen et al. 2018] as a good progressive stratified sampler, and we use the ART generator [Ahmed et al. 2017] as a good progressive tile-based sample generator. To provide a baseline, we also use a generator that produces uniformly random sample points.

Figure 7 graphs metrics of uniformity for the test set, with the following results.

	Samples	Star	Arb. Edge	Diaphony	Distance
K21-2 best	$[2, 256]$	12%	7%	35%	79%
	$[257, 512]$	29%	0%	90%	95%
	$[513, 1024]$	38%	0%	95%	100%
	$[1025, 2048]$	22%	0%	92%	100%

The K21-2 sequence underperformed with respect to the arbitrary edge discrepancy, but these results could be anticipated. Because Kronecker sequences are defined by an additive recurrence, they will produce subsets of samples that are colinear. When lines are used to divide the space, during the calculation of the arbitrary edge discrepancy, the samples that fall onto the line drive up the value of the discrepancy.

With the other uniformity metrics, the performance of the K21-2 sequence was respectable.

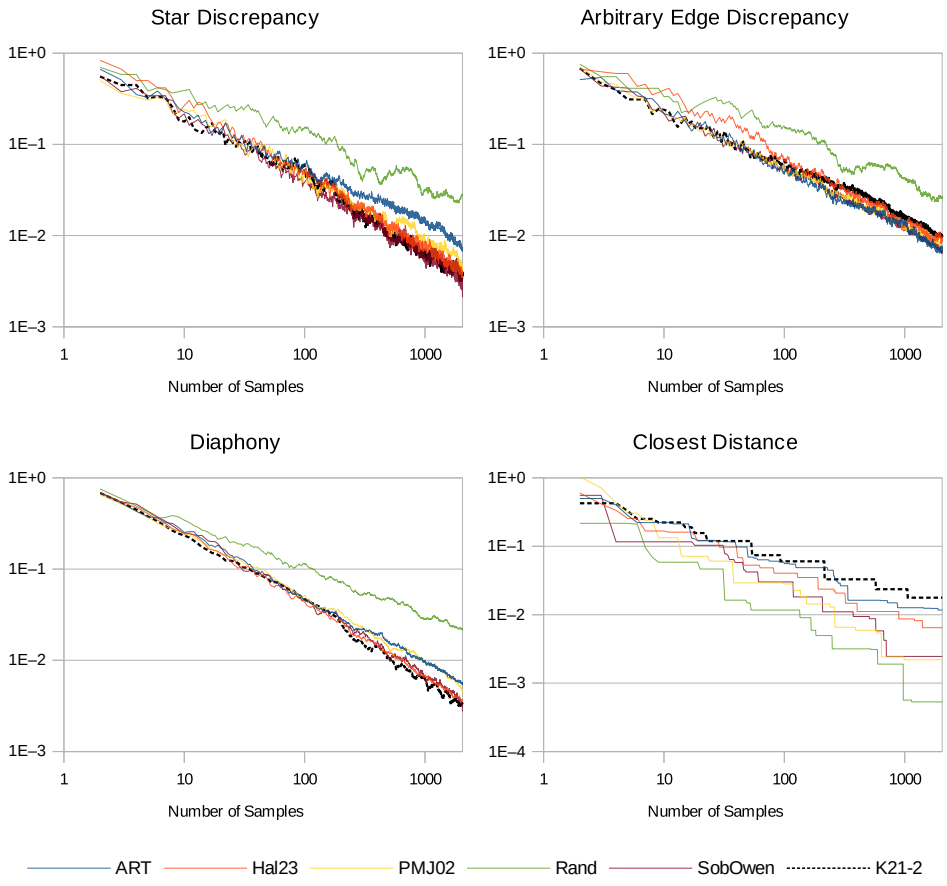


Figure 7. Metrics of uniformity for a variety of progressive sample generators.

5. Empirical Testing

5.1. Function Integration

In this section, we compare the quality of the K21-2 sample generator against various others when used to integrate a collection of simple 2D functions. For our tests, we used sample counts in the range $[2, 2048]$.

For each sample count, we used each generator in 25 trials, where the generator produced points that were used to integrate the function of interest. The ART, Halton23, R2, and K21-2 generators used a different uniformly random Cranley–Patterson rotation for each trial. The PMJ02, Sobol–Owen, and uniformly random generators used different random seeds for each trial. We report the RMS error over the full set of trials for each sample count in Figure 8. The following table summarizes an analysis of these results over different sample-count ranges.

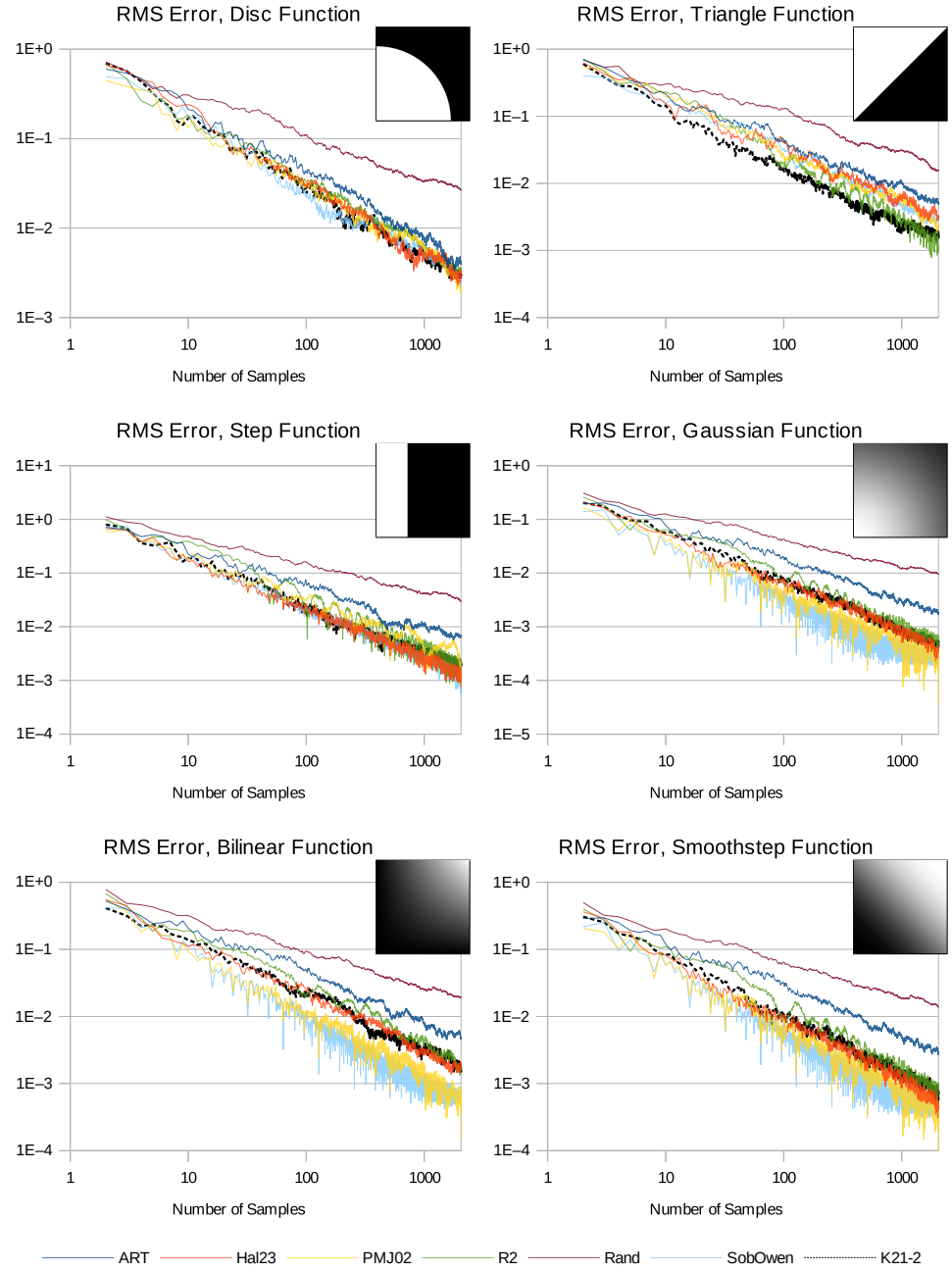


Figure 8. The RMS error using sample generators to integrate various continuous and discontinuous functions. The axes are logarithmic, base 10.

	Samples	Disc	Tri.	Step	Gaussian	Bilinear	Smoothstep
K21-2 better than R2	$[2, 256]$	90%	97%	85%	87%	83%	90%
	$[257, 512]$	88%	89%	80%	83%	97%	90%
	$[513, 1024]$	100%	92%	73%	84%	92%	83%
	$[1025, 2048]$	95%	30%	60%	78%	55%	67%
K21-2 best	$[2, 256]$	13%	95%	23%	0%	1%	1%
	$[257, 512]$	33%	89%	16%	0%	0%	0%
	$[513, 1024]$	20%	92%	2%	0%	0%	0%
	$[1025, 2048]$	40%	30%	0%	0%	0%	0%

The K21-2 sequence frequently outperformed the R2 sequence. When compared with other generators, it performed respectably for discrete signals; but it underperformed the Sobol–Owen and PMJ02 generators when sampling continuous signals.

5.2. Pixel Sampling

To test our generators for pixel sampling applications, we sampled signals into 256×256 images and measured the resulting RMS error. We began by rendering a reference image with 512Ki samples per pixel. For each test, 11 trials were run with the same sample generator and the same sample count. Each generator was used to produce 25 variations. Each variation of the ART, Halton23, R2, and K21-2 generators used a different random Cranley–Patterson rotation. Variations of the PMJ02, Sobol–Owen, and uniformly random generators used different random seeds. A variation of the generator being tested was selected randomly to be used for each pixel of each trial. The RMS error for the resulting image was recorded, and the median of the results of the trials is reported in Figure 9. An analysis of these data is summarized in the table that follows.

	Samples	Binary Zone Plate	Smooth Zone Plate	Checker-board	Tiled Bells
K21-2 better than R2	$[2, 256]$	91%	93%	91%	93%
	$[257, 512]$	99%	100%	100%	100%
	$[513, 1024]$	74%	94%	72%	96%
	$[1025, 2048]$	48%	67%	40%	75%
K21-2 best	$[2, 256]$	47%	0%	9%	6%
	$[257, 512]$	67%	0%	48%	25%
	$[513, 1024]$	84%	0%	12%	30%
	$[1025, 2048]$	45%	0%	18%	75%

As before, the K21-2 generator frequently showed better results than the R2 generator. Compared against the others, the K21-2 underperformed the Sobol–Owen and PMJ02 generators for the smooth zone plate; but it performed well for the binary zone

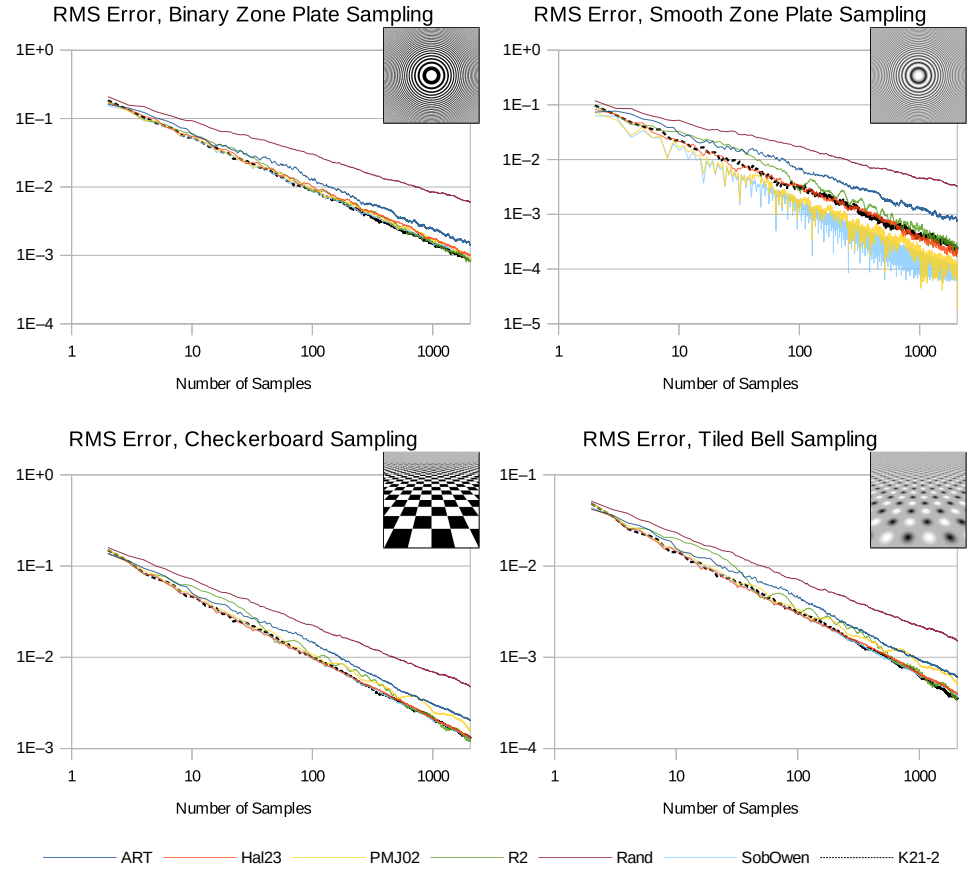
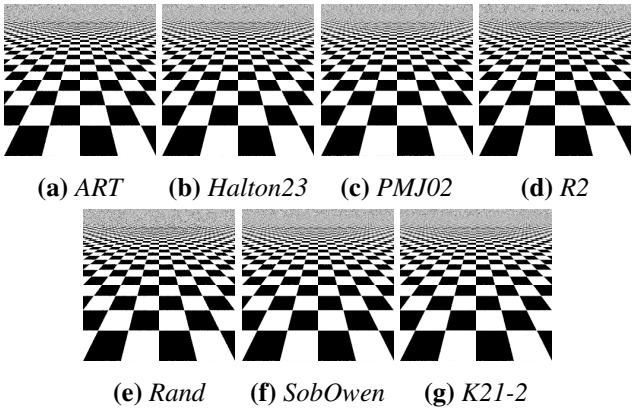


Figure 9. The RMS error using sample generators to pixel-sample various continuous and discontinuous functions.

plate. For the other signals, its performance was respectable. Figure 10 shows some renders that resulted from this test.



(a) *ART* **(b)** *Halton23* **(c)** *PMJ02* **(d)** *R2*

(e) *Rand* **(f)** *SobOwen* **(g)** *K21-2*

Figure 10. Renders of the binary checkerboard using different sample generators. Each render used 8 samples per pixel.

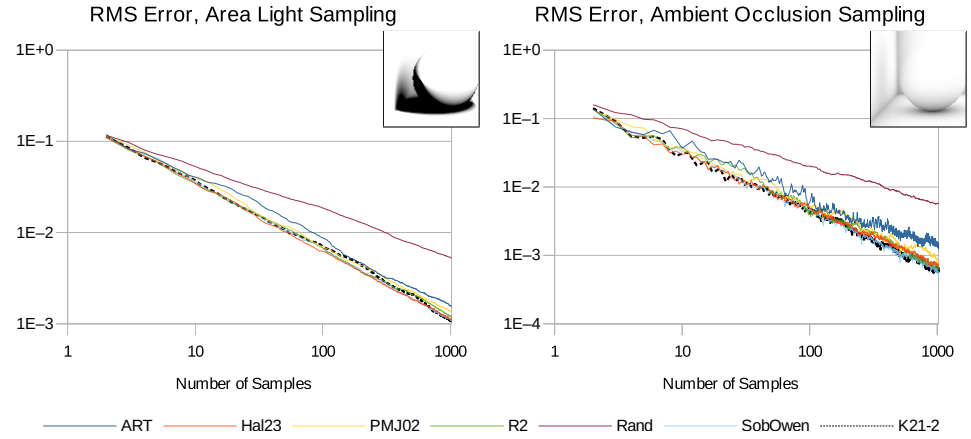


Figure 11. The RMS error using various sample generators to calculate shadows.

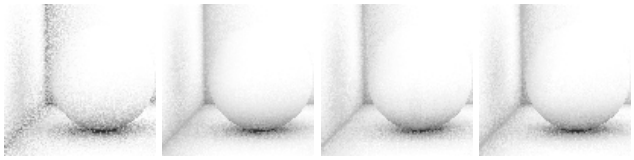
5.3. Shadow Sampling

We used Beason's path tracing code base [Beason 2014] to implement a simple framework for testing shadow sampling. We conducted two tests, one for sampling the shadow from a large spherical area light and another for calculating the ambient occlusion of the same scene. Images were rendered at a resolution of 100×100 pixels using one camera ray per pixel. Upon intersection with geometry, a number of rays were spawned to conduct the shadow calculation using sample points to determine the direction of each ray. Each instance of a shadow calculation used a randomly selected variation of the sample generator being tested. Each sample generator had 25 variations, produced in the same manner described in the last section.

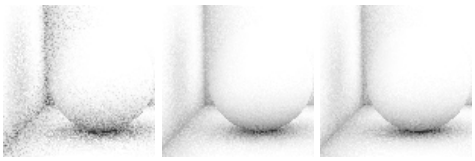
We measured the resulting RMS error against a reference rendered with 64Ki samples per shadow calculation. The results are graphed in Figure 11. An analysis is summarized in the table that follows.

	Samples	Area Light	Ambient Occlusion
K21-2 better than R2	$[2, 128]$	43%	82%
	$[129, 256]$	0%	74%
	$[257, 512]$	51%	100%
	$[513, 1024]$	96%	93%
K21-2 best	$[2, 128]$	2%	35%
	$[129, 256]$	0%	27%
	$[257, 512]$	0%	39%
	$[513, 1024]$	59%	58%

With these tests, the K21-2 generator outperformed the R2 generator more modestly. When compared with the other generators, it was infrequently the best performer, but its results were respectable. In Figure 12, we show images produced during this test.



(a) *ART* **(b)** *Halton23* **(c)** *PMJ02* **(d)** *R2*



(e) *Rand* **(f)** *SobOwen* **(g)** *K21-2*

Figure 12. Ambient occlusion renders using various sample generators. Each render used 8 samples per shading point for the ambient occlusion calculation.

5.4. Path Tracing

We tested our collection of sample generators by rendering a simple scene with path tracing. A variation of each generator was selected randomly to cast camera rays into the scene for each pixel. The variations were generated in the same manner described in the previous sections. Diffuse reflection rays also shared a randomly selected variation of a generator if they originated from the same pixel, had the same ray depth, and were reflecting from the same surface. An explicit light sampling calculation occurred for diffuse surfaces; area light samples shared a randomly selected variation of a generator if they originated from the same pixel, had the same ray depth, and were passing from the same surface to the same light.

Rays were cast into the scene at a variety of sample counts for each generator to produce 256×128 resolution images. The RMS error for each image was calculated using a reference rendered with 512Ki rays cast per pixel.

Figure 13 presents graphs of the results. This was perhaps the least dynamic result of all the testing; the various sample generators produced very similar results. Nonetheless, an analysis of the modest differences between them found that the K21-2 generator outperformed the R2 generator 75% of the time. It performed best of all the generators 59% of the time over the full range of samples. It did particularly well at lower sample counts; in the range $[2, 256]$, it was best 84% of the time.

5.5. Performance Testing

We tested the performance of our implementation of 2D Kronecker sequences against various other fast implementations of sample generators. We used the following:

- A uniformly random 2D generator using the fast pseudo-random number generator presented by Kensler for use in his stratified sample generator [Kensler

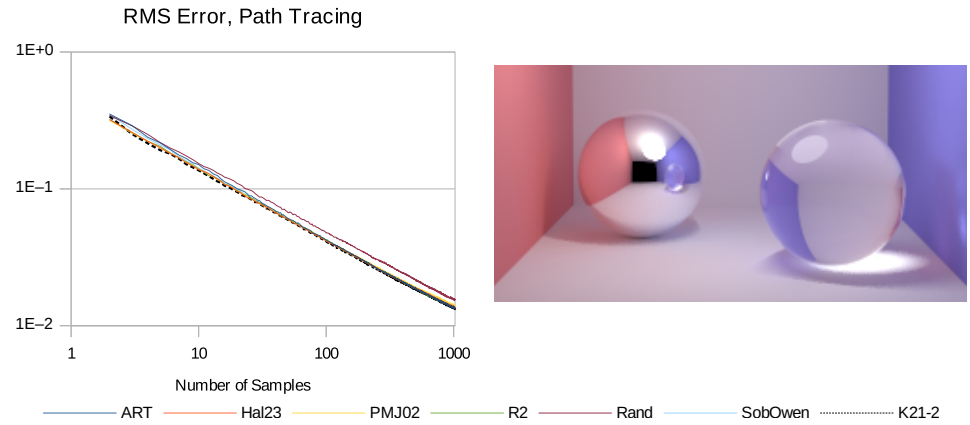


Figure 13. The RMS error using various sample generators to path-trace a simple scene.

2013]. In our tests, it produced floating-point values significantly faster than the glibc drand48() generator.

- Grünschloß's implementation of the Halton23 sequence.
- Burley's implementation of the Sobol sequence with Owen scrambling.
- The Sobol generator of Heitz et al. [2019]. This generator does not satisfy all the criteria set forth in Section 1; for example, its lookup tables depend upon resolution and sample count. Nonetheless, it is considered to be particularly fast, so it was included in the test.
- The stochastic Sobol sequence, with Owen scrambling, of Helmer et al. [2021]. We used the implementation from Listing 2 of their paper. We replaced its use of the drand48() generator with Kensler's generator to improve performance and to reduce the calculation to single-precision floating-point values.

All the implementations tested produce 32-bit floating-point samples. We tested each generator using a single thread on an Intel Xeon W3670 CPU running at a clock rate of 3.20 GHz. This is an older-generation processor; however, we expect that the relative performance between generators will be approximately the same when these algorithms are executed on other kinds of hardware.

	Million Samples per Second
2D Kronecker	515
Uniformly Random	154
Heitz Sobol–Owen	139
Helmer Stochastic Sobol–Owen	100
Grünschloß Halton23	73
Burley Sobol–Owen	10

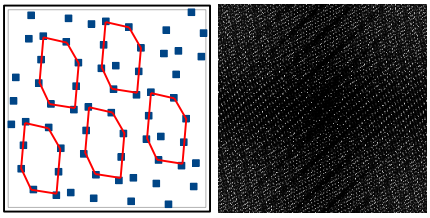


Figure 14. Left: The samples of a Kronecker sequence naturally create recurring shapes of points. Right: As a result, the power spectral density is similar to that of a lattice signal, where energy is not favorably distributed.

Clearly, the 2D Kronecker sequence is a significantly faster generator than the others. Our implementation uses, for each dimension, one integer addition, one integer-to-floating-point conversion, and one floating-point multiply operation. Details for acquiring our implementation are presented in the supplementary materials section.

6. Blue Noise Properties

Kronecker sequences produce groups of samples in shapes that recur. Indeed, we exploited this property in an earlier section to create a critical optimization for the algorithms that search for useful sets of irrational values.

Unfortunately, this property of Kronecker sequences means that the power spectral density (PSD) does not show very good quality. We visualize this in Figure 14. The energy is concentrated in impulses instead of being well-distributed.

Two-dimensional sequences with favorable blue noise properties can be created from one-dimensional low-discrepancy sequences by permuting them in a series of specific ways [Ahmed et al. 2016]. Kronecker sequences seem well suited as inputs to this approach, but a full investigation is beyond the scope of this paper. The method is a unique sample generator unto itself, whereas the subject of this section is to refine Kronecker sequences as conservatively as possible to improve their blue noise properties.

A method for improving the blue noise properties of Kronecker sequences has been presented [Roberts 2018a]. The approach involves jittering samples with a diminishing amplitude as the number of samples generated increases. This technique applies to any dimension.

We revise that method here in two ways. First, we focus on fast jittering using only square areas, introducing a conservative scaling factor to reduce the probability that square jitter areas will overlap between nearby samples. Second, we demonstrate that a second Kronecker sequence can be used as a source for jitter values. Previously, a multiplicative recurrence was used, which requires special care to calculate due to the limits of floating-point precision. In the previous section, we saw that Kronecker sequences can be generated faster than pseudo-random numbers, so using them for jittering improves performance.

First, we need to define a volume within which each sample may be jittered. We begin by dividing the space of the unit hypercube into smaller hypercubes, each associated with a sample. If we divide the space equally among the samples produced thus far, then the edge length of the hypercube associated with the nth sample in s dimensions is given by

$$L_n = \frac{1}{\sqrt[s]{n}}. \tag{5}$$

A corrective factor of $\gamma = 0.78$ has been suggested to account for the expected closest distance between points of the R2 Kronecker sequence [Roberts 2018a]. We preserve its use. Furthermore, we introduce a corrective factor of $\frac{1}{\sqrt{s}}$ to account for the length of the diagonal of the s-dimensional hypercube. This prevents jitter areas from overlapping when the vector between the two closest points is not axis-aligned. Last, we apply a user-control factor, j, in the range $[0, 1]$ to allow the jittering effect to be modulated. Applying these factors to the edge length gives us the window within which we may displace each sample. If $\overline{S}_0[n]$ and $\overline{S}_1[n]$ are the nth vectors produced by two Kronecker sequences of good quality, then the jittered sample in dimension s is given by

$$\overline{S}[n] = \left\{ \overline{S}_0[n] + \left(\frac{L_n \cdot \gamma \cdot j}{\sqrt{s}} \right) \cdot \overline{S}_1[n] \right\}. \tag{6}$$

In our testing, we used K21-2 as $\overline{S}_0[n]$ and K21b-2 as $\overline{S}_1[n]$. The irrationals for these Kronecker sequences are given in the appendix. Visualizations of the sample sets resulting from jittering the K21-2 sequence are given in Figure 15. The effect of jittering on the PSD is given in Figure 16. Even with the most modest example, $j = 6.25\%$, the PSD shows significant improvements.

It is worth noting that, though jittered versions of the K21-2 sequence demonstrate better blue noise properties, the performance of the jittered K21-2 sequence deteriorates definitively as the amplitude of the jitter increases. Figure 17 demonstrates this. Therefore, we can only recommend jittering when the PSD must show an improved distribution of energy for a particular application, and we recommend that the amplitude of the jitter be kept as small as the application would allow.

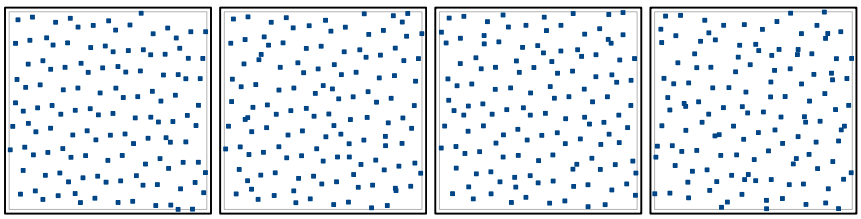


Figure 15. From left to right, 128 samples of the K21-2 sequence jittered with $j = 6.25\%$, 12.5%, 25%, and 50%.

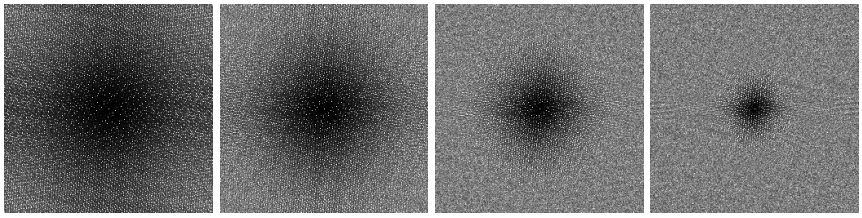


Figure 16. From left to right, the PSD of the K21-2 sequence jittered with $j = 6.25\%$, 12.5%, 25%, and 50%.

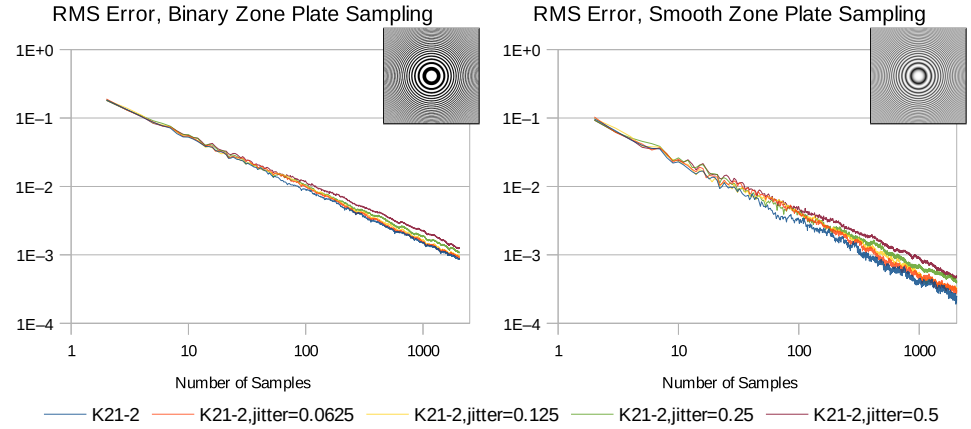


Figure 17. Comparison of the RMS error of zone plate sampling using the K21-2 sequence and various jittered versions of it.

7. Conclusions

We presented an original method for finding sets of irrationals to drive Kronecker sequences, which may be used to generate multidimensional samples with extreme speed for computer graphics applications. We discussed a method for improving the blue noise properties of such sequences. The irrational values found by our methods compare favorably against those currently used to drive Kronecker sequences.

Though Kronecker sequences found by our method did not always outperform other kinds of sample generators, the results were competitive. We anticipate that these sequences will be useful in adaptive sampling applications where the speed and compactness of the sample generator is important.

Acknowledgements

We would like to thank the reviewers and John Carey for useful comments in the development of this paper.

Appendix

This section presents the results of our application of the algorithms described in Section 3. We begin the name of each irrational set with a "K" to indicate that it is intended to produce a Kronecker sequence. Then, we use "21" for the year of its discovery. Last, we append the dimension of the sequence.

As described, we tested a large number of irrational sets. This is given by the column "Num. Sets Tested." During the execution of the test, we iteratively increased the number of samples produced by the Kronecker sequence. The maximum number of samples tested is given by the column "Num. Samples Tested."

The best results are given by the following:

Name	Num. Samples Tested	Num. Sets Tested	Irrational Values
K21-2	514 K	4 Mi	$\frac{\sqrt{506598872547596}}{29147227}$, $\frac{\sqrt{107882942223468}}{28993644}$
K21-3	832 K	2 Mi	$\frac{\sqrt{136155583282554}}{19015340}$, $\frac{\sqrt{263438703080803}}{17181595}$, $\frac{\sqrt{352662070147437}}{22118332}$
K21-4	1.34 M	1 Mi	$\frac{\sqrt{1062447381118571}}{33084971}$, $\frac{\sqrt{147063651917932}}{30639341}$, $\frac{\sqrt{711707016062345}}{29661368}$, $\frac{\sqrt{328399936443598}}{27256281}$

Next, we present secondary results, which are useful for the methods that improve the power spectral density of a Kronecker sequence, described in this paper. We use a "b" in these names to indicate that they are secondary results. Though these irrationals did not outperform the ones listed above, they are useful nonetheless:

Name	Num. Samples Tested	Num. Sets Tested	Irrational Values
K21b-2	514 K	4 Mi	$\frac{\sqrt{415745956465435}}{32662800}$, $\frac{\sqrt{16340581432791}}{25338159}$
K21b-3	832 K	2 Mi	$\frac{\sqrt{6742281674969}}{20126138}$, $\frac{\sqrt{42845384312863}}{18315113}$, $\frac{\sqrt{1044922263929}}{25238999}$
K21b-4	1.34 M	1 Mi	$\frac{\sqrt{79054014721081}}{17204034}$, $\frac{\sqrt{7916082904289}}{18894472}$, $\frac{\sqrt{859650028021546}}{29772799}$, $\frac{\sqrt{623200003618550}}{27601088}$

Index of Supplemental Materials

Our reference implementation of Kronecker sequences is part of the Uni{form|corn} tool kit, available at github.com/utk-team/utk.

A publication snapshot of the Kronecker sequence code is also provided in jcgt.org/published/0011/01/04/sample-code.zip.

References

AHMED, A. G. M., PERRIER, H., COEURJOLLY, D., OSTROMOUKHOV, V., GUO, J., YAN, D.-M., HUANG, H., AND DEUSSEN, O. 2016. Low-discrepancy blue noise sampling. *ACM Transactions on Graphics 35*, 6 (November), 247:1–247:13. URL: https://doi.org/10.1145/2980179.2980218. 74

AHMED, A. G. M., NIESE, T., HUANG, H., AND DEUSSEN, O. 2017. An adaptive point sampler on a regular lattice. *ACM Transactions on Graphics 36*, 4 (July). URL: https://doi.org/10.1145/3072959.3073588. 56, 66

BEASON, K., 2014. smallpt: Global illumination in 99 lines of C++. Personal website, November. URL: https://www.kevinbeason.com/smallpt/. 71

BRIDSON, R. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, Association for Computing Machinery. URL: https://doi.org/10.1145/1278780.1278807. 56

BURKARDT, J., 2012. Diaphony: The "Diaphony" (dispersion) of an M-dimensional pointset. Personal website, January. URL: https://people.math.sc.edu/Burkardt/cpp_src/diaphony/diaphony.html. 64

BURLEY, B. 2020. Practical hash-based Owen scrambling. *Journal of Computer Graphics Techniques (JCGT) 10*, 4 (December), 1–20. URL: http://jcgt.org/published/0009/04/01/. 66

CHEN, C.-M., BHATIA, R., AND SINHA, R. K. 2003. Multidimensional declustering schemes using golden ratio and Kronecker sequences. *IEEE Transactions on Knowledge and Data Engineering 15*, 3, 659–670. URL: https://doi.org/10.1109/TKDE.2003.1198397. 61

CHI, H. 2013. Generation of parallel modified Kronecker sequences. *Monte Carlo Methods and Applications 19*, 4, 261–271. URL: https://doi.org/10.1515/mcma-2013-0008. 58

CHRISTENSEN, P., KENSLER, A., AND KILPATRICK, C. 2018. Progressive multi-jittered sample sequences. *Computer Graphics Forum 37*, 4, 21–33. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13472. 56, 66

CRANLEY, R., AND PATTERSON, T. N. L. 1976. Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis 13*, 6, 904–914. URL: https://doi.org/10.1137/0713071. 57

DICK, J., KUO, F. Y., AND SLOAN, I. H. 2013. High-dimensional integration: The quasi-Monte Carlo way. *Acta Numerica 22*, 133–288. URL: https://doi.org/10.1017/S0962492913000044. 57

DOBKIN, D., AND MITCHELL, D. 1993. Random-edge discrepancy of supersampling patterns. In *Proceedings of Graphics Interface '93*, Canadian Information Processing Society, 62–69. URL: https://graphicsinterface.org/proceedings/gi1993/gi1993-9/. 64

DOERR, C., GNEWUCH, M., AND WAHLSTRÖM, M. 2014. Calculation of discrepancy measures and applications. In *A Panorama of Discrepancy Theory*, W. Chen, A. Srivastav, and G. Travaglini, Eds. Springer International Publishing, 621–678. URL: https://doi.org/10.1007/978-3-319-04696-9_10. 64

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. URL: https://doi.org/10.1145/1141911.1141915. 56

GRÜNSCHLOSS, L., 2021. Leonhard Grünschloß. Personal website, June. URL: http://gruenschloss.org/. 66

HEITZ, E., BELCOUR, L., OSTROMOUKHOV, V., COEURJOLLY, D., AND IEHL, J.-C. 2019. A low-discrepancy sampler that distributes Monte Carlo errors as a blue noise in screen space. In *SIGGRAPH'19 Talks*, Association for Computing Machinery. URL: https://hal.archives-ouvertes.fr/hal-02150657. 73

HELLEKALEK, P., AND NIEDERREITER, H. 1998. The weighted spectral test: Diaphony. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (Jan.), 43—-60. URL: https://doi.org/10.1145/272991.273008. 64

HELMER, A., CHRISTENSEN, P., AND KENSLER, A. 2021. Stochastic generation of (t, s) sample sequences. In *Eurographics Symposium on Rendering—DL-Only Track*, The Eurographics Association, 21–33. URL: https://diglib.eg.org/handle/10.2312/sr20211287. 73

KELLER, A., PREMOZE, S., AND RAAB, M. 2012. Advanced (quasi) Monte Carlo methods for image synthesis. In *ACM SIGGRAPH 2012 Courses*, Association for Computing Machinery, 21:1–21:46. URL: https://doi.org/10.1145/2343483.2343502. 57

KELLER, A., GEORGIEV, I., AHMED, A., CHRISTENSEN, P., AND PHARR, M. 2019. My favorite samples. In *ACM SIGGRAPH 2019 Courses*, Association for Computing Machinery, 15:1–15:271. URL: https://doi.org/10.1145/3305366.3329901. 57, 58

KELLER, A. 2006. Myths of computer graphics. In *Monte Carlo and Quasi-Monte Carlo Methods 2004*, H. Niederreiter and D. Talay, Eds., Springer, 217–243. URL: https://doi.org/10.1007/3-540-31186-6. 58

KELLER, A. 2013. Quasi-Monte Carlo image synthesis in a nutshell. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, Eds., Springer, 213–249. URL: https://doi.org/10.1007/978-3-642-41095-6_8. 57

KENSLER, A., 2013. Correlated multi-jittered sampling. Technical Memo 13-01, Pixar, March. URL: https://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf. 56, 73

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics 25*, 3 (July), 509–518. URL: https://doi.org/10.1145/1141911.1141916. 56

KUIPERS, L., AND NIEDERREITER, H. 1974. *Uniform Distribution of Sequences*. A Wiley Interscience Publication. Wiley. URL: https://books.google.ca/books?id=XMvYRwAACAAJ. 57, 58

LIU, H., HAN, H., AND JIANG, M. 2021. Rank-1 lattices for efficient path integral estimation. *Computer Graphics Forum 40*, 2, 91–102. URL: https://doi.org/10.1111/cgf.142617. 56

NIEDERREITER, H., AND WINTERHOF, A. 2015. *Applied Number Theory*. Springer International Publishing. URL: https://books.google.ca/books?id=5H6BCgAAQBAJ. 58, 61

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. In *ACM SIGGRAPH 2007 Papers*, Association for Computing Machinery, 78:1–78:6. URL: https://doi.org/10.1145/1275808.1276475. 56

ROBERTS, M., 2018. A simple method to construct isotropic quasirandom blue noise point sequences. *Extreme Learning*, November 13. URL: http://extremelearning.com.au/a-simple-method-to-construct-isotropic-quasirandom-blue-noise-point-sequences/. 74, 75

ROBERTS, M., 2018. The unreasonable effectiveness of quasirandom sequences. *Extreme Learning*, April 25. URL: http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/. 58, 64

SCHLÖMER, T., HECK, D., AND DEUSSEN, O. 2011. Farthest-point optimized point sets with maximized minimum distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, Association for Computing Machinery, 135–142. URL: https://doi.org/10.1145/2018323.2018345. 58

SCHRETTER, C., KOBBELT, L., AND DEHAYE, P.-O. 2012. Golden ratio sequences for low-discrepancy sampling. *Journal of Graphics Tools 16*, 2, 95–104. URL: https://doi.org/10.1080/2165347X.2012.679555. 58

SCHRETTER, C., HE, Z., GERBER, M., CHOPIN, N., AND NIEDERREITER, H. 2016. Van der Corput and golden ratio sequences along the Hilbert space-filling curve. In *Monte Carlo and Quasi-Monte Carlo Methods*, R. Cools and D. Nuyens, Eds., Springer International Publishing, 531–544. URL: https://doi.org/10.1007/978-3-319-33507-0_28. 58

SHIRLEY, P. 1991. Discrepancy as a quality measure for sample distributions. In *EG 1991—Technical Papers*, Eurographics Association, 183–194. URL: https://doi.org/10.2312/egtp.19911013. 56

WEISSTEIN, E., 2021. Hypersphere packing. *MathWorld—A Wolfram Web Resource*, March. URL: https://mathworld.wolfram.com/HyperspherePacking.html. 59

ZHU, Y. C. 2007. Discrepancy of certain Kronecker sequences concerning transcendental numbers. *Acta Mathematica Sinica, English Series 23*, 10, 1897–1902. URL: https://doi.org/10.1007/s10114-005-0939-0. 58

ZINTERHOF, P. 1996. Parallel generation and evaluation of Weyl sequences. Tech. rep., RIST++, March. URL: https://www.researchgate.net/publication/2706744_Parallel_Generation_and_Evaluation_of_Weyl_Sequences. 58

Author Contact Information

Mayur Patel
Prodigy Education
226 Wyecroft Road
Oakville ON Canada, L6K 3X7
patelm@acm.org

Mayur Patel, Optimizing Kronecker Sequences for Multidimensional Sampling, *Journal of Computer Graphics Techniques (JCGT)*, vol. 11, no. 1, 55–81, 2022
http://jcgt.org/published/0011/01/04/

Received:	2021-07-10		
Recommended:	2021-11-27	Corresponding Editor:	Matt Pharr
Published:	2022-02-02	Editor-in-Chief:	Marc Olano