Orthogonal Array Sampling for Monte Carlo Based Rendering

Undergraduate Thesis

Dartmouth College

Afnan Enayet
Dr. Wojciech Jarosz
Advisor

Dartmouth Computer Science Technical Report TR2019-872

June 14, 2019

Abstract

In computer graphics (especially in offline rendering), the current state of the art rendering techniques utilize Monte Carlo integration to simulate light and calculate the value of each pixel in order to generate a realistic-looking image.

Monte Carlo integration is a highly efficient method to estimate an integral that scales extremely well to a high number of dimensions, making it well suited for graphics, because generating images creates a high-dimensional integrand. The efficiency of these Monte Carlo integrations depends on the sampling techniques used, and using a more efficient sampling technique can make a Monte Carlo simulation converge to the right answer quicker than using more naive sampling techniques.

In this thesis, we present an efficient sampling method that demonstrates much higher performance than many other sampling techniques. This novel sampling method, based on orthogonal arrays, offers guaranteed stratification in arbitrary projections, leading to better theoretical performance with integrands that have cross-correlated variance compared to sampling methods that do not offer these same guarantees.

Acknowledgements

I would like to thank my thesis advisor, Wojciech Jarosz, whose help was crucial in the development of this thesis. I would also like to acknowledge Andrew Kensler, Per Christensen, and Charlie Kilpatrick from Pixar for their help and collaborative efforts in the development of this thesis and the related research paper. I would also like to thank Rebecca Luo for her support, patience, and critical eye.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Background

This section will explore background knowledge that pertains to the findings in this thesis. In order to understand the significance of a sampling method, we must first explore the fundamentals of Monte Carlo integration and how different sampling strategies can affect the effectiveness of Monte Carlo integration. The background section will discuss different conventional sampling techniques for Monte Carlo integration.

1.1 Monte Carlo Integration

Suppose we have a square dartboard that is one meter by one meter. We will call the length of a side l. We know that the area of a square is simply l^2, so the area of the dartboard is 1. Now suppose we have a perfect circle inside of the square, and we do not know the area of this circle. We have an unlimited supply of darts, and we know whether each dart we throw falls inside the circle, or outside of the circle (assuming that every dart hits the dartboard). If we want to find the area of the circle, we can throw a bunch of random darts, record the number of darts that hit the circle, and use the ratio of the number of darts that hit the circle to the total number of darts thrown to find the area of the circle. For example, if about a quarter of our darts hit the circle, then we can reasonably estimate that the size of our circle is $0.25 \cdot 1 = 0.25$.

The principle of stochastically estimating some arbitrary area or shape is the same principle that underlies Monte Carlo integration. Instead of a perfectly square dartboard, imagine an

arbitrary integration domain. Now replace the circle with some arbitrary function. We can define the integral of the function as the area under the function. Now imagine throwing darts at random, within some domain, except now we are trying to find the integral of a function rather than the area of a circle.

In practice, Monte Carlo integration involves randomly selecting some point in the function's domain ($\mathbb{R}^d$) a certain number of times (N) and averaging the calculated values.

$$\int_0^1 f(x) \approx \frac{1}{N} \sum_{i=1}^{i=N} f(\overline{x}_i) \quad \text{where } \overline{x}_i \text{ is a uniform random number} \tag{1.1}$$

It is important to note that it is not strictly necessary to use uniform random sampling for Monte Carlo integration, but doing so makes the explanation of Monte Carlo simpler. An important property of Monte Carlo integration is that the estimation is guaranteed to converge towards the correct answer as N approaches infinity. As a result, N is a variable that allows one to choose between speed and accuracy.

$$\lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{i=N} f(\overline{x}_i) = \int_a^b f(x) \tag{1.2}$$

1.1.1 Variance

Generally, we want to know how reliable a numerical method is. If we have some method of approximating a value, it seems useful to know whether the answer we get will vary wildly across different runs, or whether it tends to be fairly reliable. In most cases, we want more reliability, something that becomes difficult to attain with the element of randomness. The measure of how unreliable these approximations are is called **variance**, also denoted as σ^2.

$$\sigma^2 = E[X - \mu^2] \tag{1.3}$$

where E denotes the **expected value**, μ is the average value of many runs, and X is simply a random variable (Loeve, 1977). In this case, X is the result of some run of a Monte Carlo integration.

If we get the same value every single time, then the difference between the result of a particular and the average of all the runs will be 0. Clearly, if the answers are more reliable, they will vary less. Variance gives us a way to quantify the reliability of random variables, or variables that are the result of some sort of random process. It is important to note that variance is not the same as accuracy - while having low variance is desirable, it does not necessarily imply that the answers that we get are correct.

1.1.2 Convergence Rate

Even if we know that Monte Carlo integration will eventually converge to the correct answer, it is useful to know how quickly it will converge. We call this property the **convergence rate**. One thing that makes Monte Carlo integration so appealing for many use cases is that its convergence rate is independent of the dimensionality of the integrand. Whether our integrand is a 2D or a 90D integrand, the accuracy of the approximate integral will increase at the same rate. While Monte Carlo with uncorrelated sampling has the advantage of having a convergence rate that does not scale with dimensionality, there are methods that make it possible to yield a steeper convergence rate. The convergence rate of Monte Carlo integration with a particular sampling method can be calculated theoretically given knowledge about the sampling methods and integrand, which yields an asymptotic bound as with random sampling, or empirically, by measuring the variance of a sampling method at various sample counts with a high number of trials. As part of our research, we ascertained the theoretical convergence bounds for various integrands and samplers, and verified them empirically.

1.2 Sampling

One heavily researched method to speed up Monte Carlo integration entails changing the sampling strategy. Recall the example with the dartboard. What if instead of randomly throwing darts at the board, we were to break the dartboard up into small grids? Doing so would ensure that we cover all parts of the dartboard, alleviating the worry that we might have missed an area of the board.

Many sampling strategies were designed to address exactly this concern. Naive random sampling can fail by oversampling or undersampling critical parts of a function that will heavily influence the average (usually parts of a function that are really large or really small relative to the average value of the function). In this section, we will explore notable sampling methods that are used in Monte Carlo integration that we considered when developing the new sampling methods that will be introduced in this paper.

While this section will describe and explore many sampling methods, this is not necessarily a comprehensive survey of all Monte Carlo sampling methods. These are important sampling methods that we looked at as prior work as we worked on orthogonal array based sampling.

1.2.1 Discrepancy

Some common properties of sampling methods that are often used to describe a point set include something called the **discrepancy** of a point set. The discrepancy is a numerical measure of how spaced out a point set is (Shirley, 1991). We do not want our points to be clumped together, and ensuring that all the points are well spaced out tends to be a property that is very important for Monte Carlo sampling. In the literature, many points sets are referred to in a favorable manner by being described as a low-discrepancy sequence. There are several ways to measure discrepancy, but in this thesis, we do not actually calculate the discrepancy values of different point sets, so it is not important to know the mathematical definition so much as the general implication of the term.

1.2.2 Padding

There is a common technique used specifically for high-dimensional integrands called **padding** which combines different instances or shuffles of a low-dimensional sequence or point set to create a higher dimensional point set. Suppose that we have some point set that has a favorable 2D distribution. We want to use this point set, which is well stratified in two dimensions, on a four dimensional integrand. Suppose that we have N samples, which consist of two-tuples

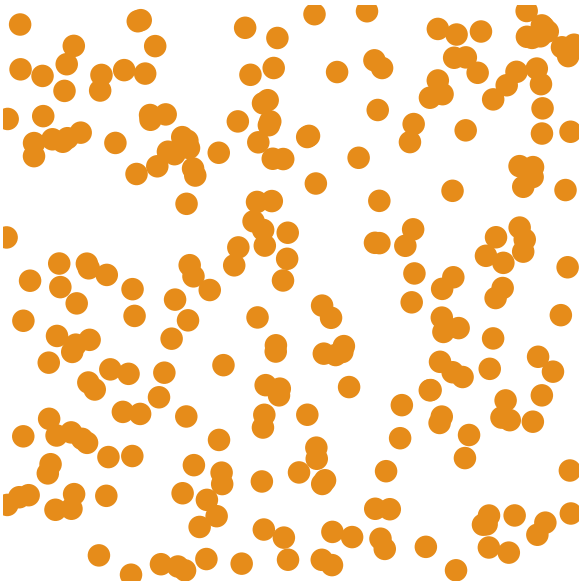

Figure 1.1: Random sampler, 256 points

(the first element representing the first dimension, and the second element representing the value in the second dimension). Now shuffle the indices of the point set, and retain that list of numbers, calling it L, so that the first element of the original point set corresponds to the first element in L. We can use these shuffled indices to add two more dimensions to the point set, by selecting some index, i, from the original point set, and "padding" it with some other index l which is imply the i^{th} element of L, yielding four dimensions. This method is called uncorrelated jittering (Cook u. a., 1984). There have been other methods proposed for padding, such as Owen Scrambling (Owen, 1997) and random digit XOR scrambling (Kollig und Keller, 2002).

Padding provides an obvious advantage with respect to efficiency. Consider that we can effectively "re-use" a point set to add more dimensions to it without actually computing or storing more samples. If we only need a good distribution in a lower set of dimensions than the dimensionality of the integrand, then padding is an easy way to add the dimensions we need.

1.2.3 Random Sampling

We have already discussed one method for Monte Carlo sampling: uniform random sampling. There are several advantages to using this method, namely that it is relatively easy to implement

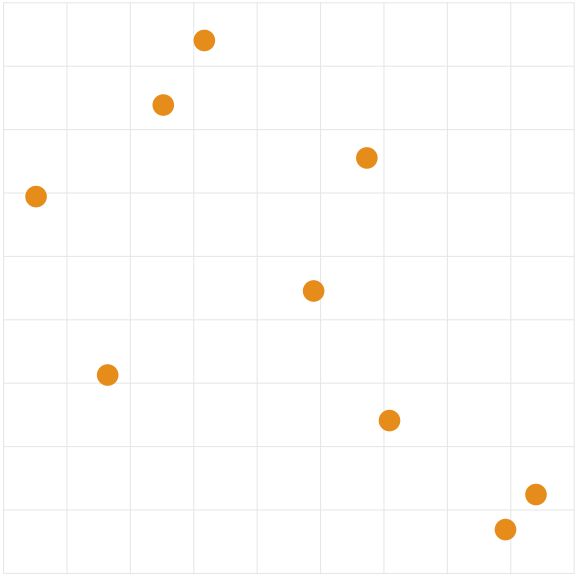

Figure 1.2: N-rooks sampler with strata lines, 9 points

correctly and simple to understand. We can see what such a point set looks like in Figure 1.1.

There is an intuitive pitfall that results from using naive random sampling. Consider a scenario where we are trying to sample a step function in which $f(x) = 0$ for $x \leq 0.5$ and $f(x) = 1$ for $x > 0.5$. In this case, if it so happens that every sample falls in the region where $f(x) = 0$, then the estimated integral will be 0, which is clearly incorrect. Random sampling presents the danger of oversampling or undersampling regions of functions. This phenomenon becomes even more dangerous with functions that vary more across the sampled domain.

1.2.4 N-Rooks Sampling

N-Rooks sampling is a method which yields a perfect 1D distribution of points (Shirley, 1991). Imagine a finite 1D line. Now suppose that we want to place N points on the line, ensuring that none of the points "overlap." An easy method to do this is to divide the line into N sections, or **strata**, and place a point within each stratum. We can also randomly jitter a point within a strata to provide a "random" looking set of points while maintaining one-dimensional stratification. Refer to Figure 1.2 to see what this distribution looks like in two dimensions.

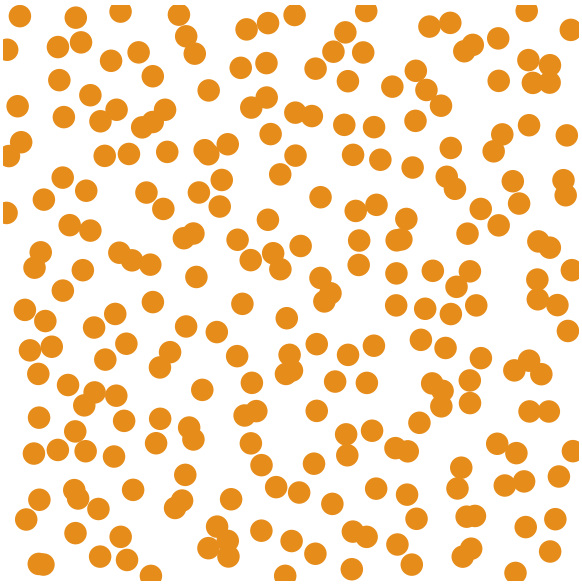

Figure 1.3: Jittered sampler, 256 points

N-Rooks sampling can be extended to multiple dimensions by generating an N-Rooks point set for each dimension and arbitrarily padding them together. This method tends to be suboptimal but has the advantage of being trivial to implement and not very computationally intensive. As we can see in Figure 1.2, the points are well-stratified in 1D projections, but are not stratified in two dimensions. These properties lead to lower discrepancy and are not desirable for Monte Carlo integration.

1.2.5 Jittered Sampling

In the jittered sampling approach, we divide the domain into different strata and place one sample randomly inside each stratum. We can see what this looks like by referring to Figure 1.3. Stratifying our sample domain ensures that each stratum has a sample inside of it, which alleviates some of the issues with uniform random sampling. For example, if we define even just two strata in the case discussed for uniform random sampling, with the step function, we can avoid the pitfall of only sampling the side where $f(x) = 0$.

This method has the advantage of being relatively simple to implement, and it can be extended in order to work in multiple dimensions. We can jitter in 2D by creating a grid of squares

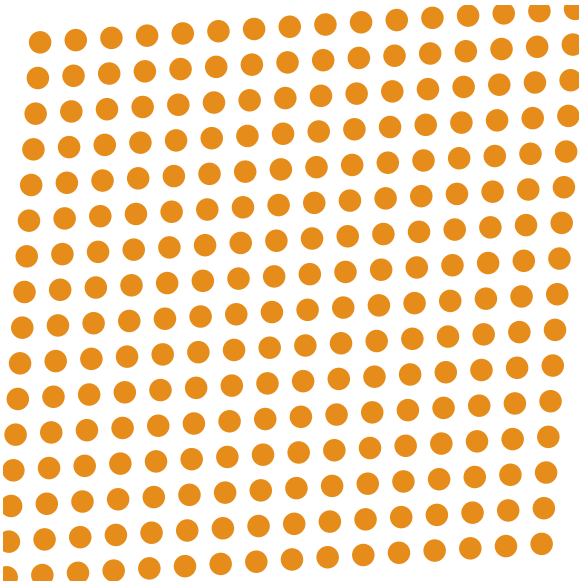

Figure 1.4: The “canonical” multi-jittered arrangement, 256 points

within the domain, jittering in 3D is the same except we select cubes, etc. Jittering falls prey to the “curse of dimensionality,” meaning it does not scale to higher dimensions efficiently, and scaling to higher and higher dimensions is progressively more expensive. Jittered sampling does not provide the most effective manner of sampling a function, as it has the unintended effect of clumping points near the boundaries of strata, as seen in Figure 1.3.

1.2.6 Multi-Jittered Sampling

This sampling method was introduced by Pete Shirley to address functions that exhibit variance in two dimensions (Shirley, 1991). Shirley wanted to address the potential clumping of points that tends to happen with jittered sampling. His solution was to combine the constraints of jittered sampling with the constraints of N-rooks. This yields a perfect distribution in each 1D projection as well as a nice distribution in 2D.

The technique to generate a multi-jittered point set is to start out by first dividing the domain (2D) into a set of strata and corresponding substrata, giving us effectively a two-tier set of grids. Start by placing a point in each stratum, forming the “canonical” arrangement, which we can see in Figure 1.4. Note that the canonical arrangement maintains the jittered and N-Rooks

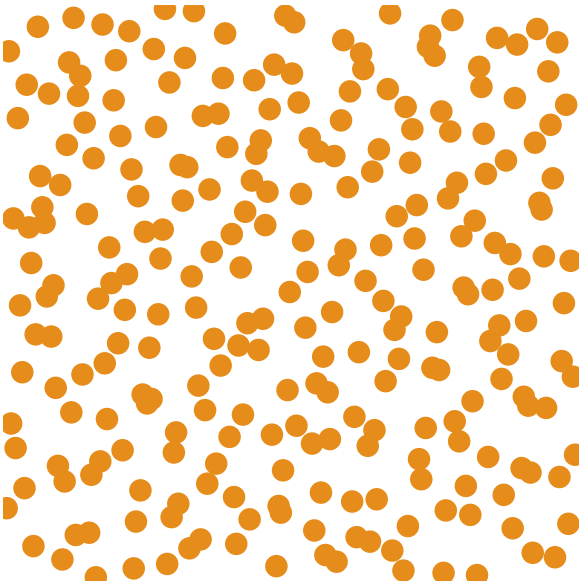

Figure 1.5: Randomized multi-jittered sampler, 256 points

constraints. Then shuffle the samples within each stratum in each dimension, while combining the jittered and N-Rooks constraints, which we can see in Figure 1.5. The jittering constraint yields stratification in a $\sqrt{N} \times \sqrt{N}$ grid of strata, supposing that there are a total of N samples in a point set. The stratification constraint guarantees that there will be at least one point in every part of this grid, ensuring that there is coverage over the entire domain. Within each large strata, there are $\sqrt{N}$ substrata, which is where the N-rooks constraint is applied. No point in one of the smaller strata can overlap vertically or horizontally with a point in another stratum.

1.2.7 Correlated Multi-Jittered Sampling

Introduced by Andrew Kensler, correlated multi-jittered (CMJ) sampling modifies Shirley's multi-jittered sampling design to provide point sets that have a better shuffling arrangement and consequently yield steeper convergence rates in empirical testing (Kensler, 2013).

The key idea behind CMJ is changing the shuffling step from multi-jittered sampling. In Shirley's method, the strata in each dimension shuffle points independently of shuffles in other strata. Kensler's modification entails using the same shuffle over each stratum in each dimension (Kensler, 2013). Qualitatively, these points look more aesthetically pleasing given the stronger

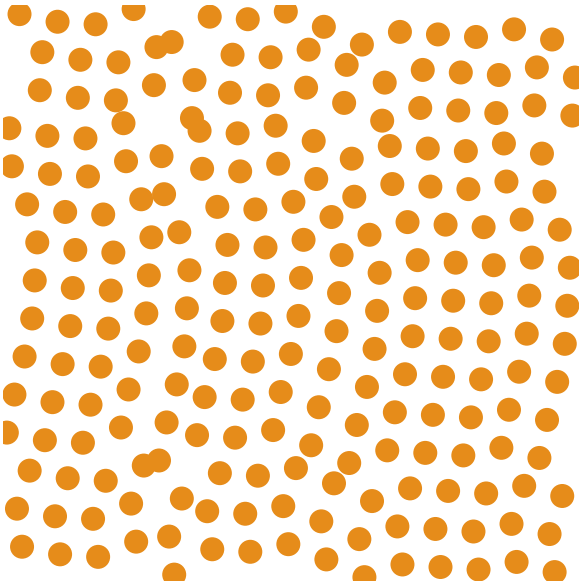

Figure 1.6: Correlated multi-jittered sampler, 225 points

constraints that CMJ has for its points as we can see in Figure 1.6.

1.2.8 Quasi Monte Carlo Sampling

All of the sampling methods discussed so far rely on some element of randomness. Each produced point has some sort of input that stems from pseudo-random number generation. There are methods of sampling that have been developed that do not rely on any sort of randomness. We call such sampling methods "Quasi-Monte Carlo" (QMC) sampling.

QMC sampling methods tend to provide much higher performance in Monte Carlo integration than other sampling methods and have become quite popular due to their superior performance, low discrepancy, and high efficiency.

There are some common point sets and sequences in QMC that are commonly referred to in the literature, such as **(t, s) sequences** and **(t, m, s) nets**. Such terms describe stratification guarantees for samplers in a convenient and consistent manner. To explain these terms, we must first define the **elementary interval**, which we describe as the elementary s-interval in

base b. Suppose we have some interval with the following properties:

$$\prod_{j=1}^{s} \left[\frac{a_j}{b^{d_j}}, \frac{a_j + 1}{b^{d_j}} \right] \tag{1.4}$$

Note that the invariants $s \geq 1$ and $b \geq 2$ must hold for this definition (Niederreiter, 1988). We can see that the elementary interval provides a notation that describes some sort of even partitioning within an arbitrary domain. This domain, consisting of evenly spaced partitions, naturally falls within the scope of sampling. This talk of elementary intervals is very similar to the idea of stratification, but intervals provide a far more formal and explicit bound.

Now let us define a (t, m, s)-net in base b. We can define one as a set of b^m points in a domain $[0, 1)^s$ such that if we define some sequence x_i, the cardinality of some elementary interval $P \cup \{x_1, \cdots, x_{b^m}\} = b^t$ for every elementary interval P in base b such that the hypervolume of $P = b^{t-m}$ (Niederreiter, 1988). While the formal mathematical definition seems daunting, the more general implication of the (t, m, s)-net constraints is that some set of points is distributed in a uniform manner which is more acutely described by the properties of the net, t, m, s.

A (t, s)-sequence is somewhat based on (t, m, s)-nets. A (t, s)-net is an infinite sequence which requires that for every $m > t$ and every $k \geq 0$, the set N is defined by Equation (1.5) (Niederreiter, 1988), where N must be a valid (t, m, s) net in base b.

$$N = \{[x_i] : kb^m \leq i < (k + 1)b^n\} \tag{1.5}$$

1.2.9 Halton Sequence

The Halton sequence, named after its author, was introduced in 1960 as a means of deterministically generating points that have low discrepancy, but at the same time, appear to look somewhat random (Halton, 1964) (refer to Figure 1.7). The Halton sequence has the additional advantage of being a progressive sequence - we do not need knowledge of the previous points in order to generate the next point. This property allows for extremely high efficiency in generating samples.

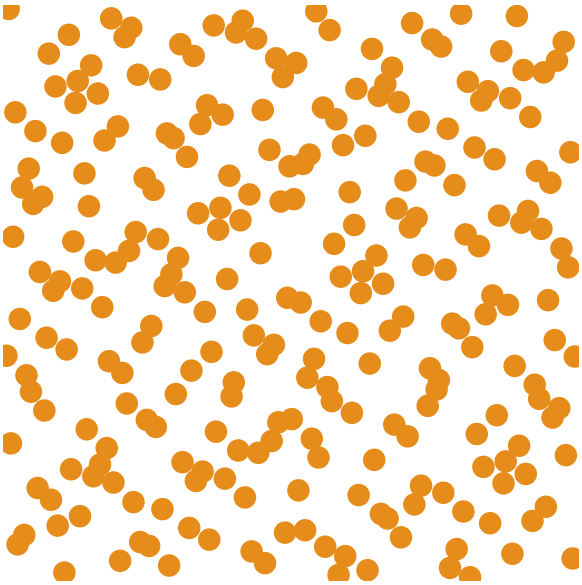

Figure 1.7: Halton sampler, 256 points

The Halton sequence can be constructed as follows: given some index in the point set, get the representation of the number in binary. Take the number, invert it by reading the number from right to left, and then place that number after a decimal point. Interpreting that number in base 10 yields the point in the $[0, 1)^d$ domain.

As an example, let us look at the point at the fifth index of the Halton sequence. $5 = 101.\ 101$ inverted is 101. Placing the number after a decimal point yields 0.101, which we interpret in base 10. It is easy to generate points in this manner, as we can trivially convert an index into binary and perform these steps to calculate where the point lies in space with high computational efficiency and very low computational overhead.

1.2.10 Sobol Sequence

Perhaps one of the most widely used sequences in Monte Carlo rendering and other Monte Carlo applications, the Sobol sequence provides a high-efficiency, low-discrepancy point set that performs extremely well in Monte Carlo integration. We can think of Sobol as the current gold standard of Monte Carlo sampling. Not only does Sobol perform well, it is also a progressive

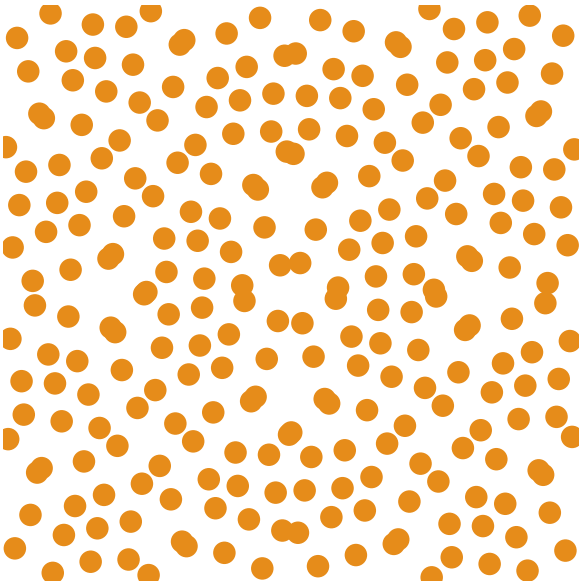

Figure 1.8: Sobol sampler, 256 points

sequence, so we do not need to know the number of samples we will use beforehand, for example, which is a limitation present in many other sampling methods. The Sobol sequence exploits the properties of numbers in their binary representations in order to progressively refine the intervals that points are distributed across (Sobol, 1967) (refer to Figure 1.8).

The Sobol sequence has the property of being well-distributed in the unit domain and can progressively partition the domain as more samples are generated to maintain a good distribution. The progressive nature of Sobol sequencing makes it very attractive to use in any scenario where we would want to have incremental sampling, such as computer graphics.

1.3 Monte Carlo in Computer Graphics

While we have extensively discussed the usage of Monte Carlo sampling and how it pertains to integration in general, this does not explain the relevance of Monte Carlo sampling methods in the context of computer graphics, and specifically pertaining to rendering.

We can generally think of the value of a pixel in a rendered image as the result of the integral of light multiplied by the color value. In 1986, James Kajiya introduced the rendering equation

(1.6), perhaps one of the most well-known fundamental equations in graphics (Kajiya, 1986).

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t)(\omega_i \cdot n)\omega_i' \tag{1.6}$$

Integration is a necessary operation when calculating the radiance value of some point. Monte Carlo integration offers an way to approximate this equation, which is extremely difficult to calculate analytically. Not only that, the rendering equation tends to be extremely high dimensional, and it becomes even more practical to use Monte Carlo integration due to the fact that its convergence rate is independent of dimensionality. Generally in rendering, taking a sample is equivalent to tracing a ray, which is a computationally expensive operation.

The practical benefit to speeding up Monte Carlo integration equates to time and money saved when actually rendering a picture or a movie. It also equates to increased accuracy if we want to use the same amount of time when rendering. In either case, optimizing Monte Carlo integration carries obvious and significant benefits for rendering.

1.4 High-Dimensional Integration

Many of the samplers we presented focus on stratifying and addressing variance in two dimensions. Rendering is extremely high dimensional and potentially infinitely dimensional. The functions that Monte Carlo integration attempts to approximate often exhibit variance in more than two dimensions. Having sampling methods that explicitly address this high-dimensional variance would offer a significant improvement in efficiency.

One of the techniques explained earlier, padding, attempts to rectify this to some degree. It generates high dimensional samples, but these samples are not stratified in high dimensions; they are only well-stratified in pairs of dimensions (if we have a base sampler that is two dimensional and padded together).

Another issue that arises is that samplers that are well-stratified in higher dimensions still

exhibit poor stratification in certain projections. Consider a four-dimensional sampler, with the dimensions $\{0, 1, 2, 3\}$. A sampler that is well-stratified in four dimensions and two dimensions can still suffer if we take some arbitrary cross-dimensional projections, such as dimension 0 and dimension 3.

The fact that arbitrary projections can exhibit poor stratification, or even the fact that many samplers do not exhibit stratification in high dimensions, can cause serious issues with Monte Carlo integration. If there is a mismatch between stratification and the dimensions in which a function exhibits variance, the performance of Monte Carlo will degrade to $O(N^{-1})$ (Singh und Jarosz, 2017).

Chapter 2

Orthogonal Arrays

In this chapter we will explore the definition of an orthogonal array (OA), as well as the statistical significance of OAs. We will then go on to demonstrate practical construction techniques for OAs.

2.1 Experiment Design

Orthogonal arrays are closely tied to statistical experiment design. Many of the terms used to describe and define orthogonal arrays in the literature are conducive to a description of OAs in the context of scientific experiments.

Suppose we want to conduct an experiment to figure out the optimal method to take care of a plant that sits by our window. The only variables we can control are how much water we give the plant and how much sunlight it receives. We would like to know the optimal levels of water and sunlight to give our plant in order to keep it alive.

Let us define the variables here: levels of sunlight and levels of water. For simplicity, let us say it is a binary choice and we can either give water/sunlight or not. We can also call these variables **factors**, and state that there are two factors we are considering in this experiment. For each of these factors, we define two **levels**, which is the binary choice we referred to earlier. For the factor water, the levels are: administering water and not administering water. For the fac-

tor sunlight, the levels are: allowing the plant to see sunlight and not allowing the plant to see sunlight.

Once we have our factors and levels figured out, we need to carry out a number of trials of our experiment. We can call these trials **runs**, so we "run" our experiment a number of times with different configurations of factors and levels.

Now suppose that we want to figure out the most efficient way to test for the optimal combination of sunlight and water levels with respect to the survival of the plant. We could test every combination of factors and levels, which would definitively tell us the effect of each factor on the dependent variable (survival of the plant). This method also has a name: **full factorial design**. With an experiment utilizing full factorial design, we get a robust route for experimentation at the expense of time. It is not a very efficient method, since we have to try every combination of factors and levels to get our results.

Orthogonal arrays attempt to provide a better way of tackling this issue by testing different levels and factors in a more efficient manner. They do so by creating arrays that can be used for experiment design, which aim to provide the most optimal method of testing the effect of a factor on the outcome of an experiment without needing to resort to the full factorial design. Often, it will provide a way for us to design an experiment without needing to try every combination of factors and levels. In fact, the full factorial design is referred to as a degenerate case of an OA (Geyer, 2014).

2.2 Defining an Orthogonal Array

Now that we have an intuition for the effective purpose and some of the terminology used with orthogonal arrays, we can go on to formally define orthogonal arrays. First, we need to define a very important term in the OA literature: **strength**. The strength of an orthogonal array must be less than or equal to the number of defined levels. If an orthogonal array has a strength of t, it means that every t-tuple from the set of levels appears exactly the same number of times

across any t factors.

In the example with the plant experiment, we had defined two levels. Let us refer to them as 0 and 1, so the set that defines the levels is as follows: $\{0, 1\}$. If we have an orthogonal array with strength two, it means that we will see the following tuples appear in the OA with the same frequency: $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$. The number of times we see the tuples appear in the OA is referred to as the **index**. For the purposes of our research, we are most interested in cases where the index is 1, also referred to as the **unit index**.

Now that we have some an understanding of the defining terms of an orthogonal array, we can provide a formal mathematical description. We denote an orthogonal array as follows:

$$OA(N, d, s, t, \lambda) \tag{2.1}$$

- N: number of runs
- d: number of factors
- s: number of levels
- t: strength
- λ: index

Note that the λ term is often omitted because it can be inferred from the other properties. If λ is known, we can easily calculate N as $N = \lambda s^t$. It is also important to note that the symbols used to define these terms are not entirely consistent across the literature, but the properties themselves are fairly consistent.

The orthogonal array for the plant experiment we described earlier would be described as $OA(4, 2, 2, 2, 1)$. We can also see what the OA actually looks like on Figure 2.1.

Table 2.1: An example of an $OA(4, 2, 2, 2, 1)$

0	0
0	1
1	0
1	1

Table 2.2: An example of an $OA(9, 4, 3, 2, 1)$

0	0	0	0
0	1	1	1
0	2	2	2
1	0	1	2
1	1	2	0
1	2	0	1
2	0	2	1
2	1	0	2
2	2	1	0

Because the number of factors is so small, the resultant OA is a full factorial design, which does not adequately show the power and advantage of using an OA. Let us change the experiment slightly so we can understand why OAs are so useful. Suppose, in this case, everything about the experiment remains the same, except now we have three levels: no sunlight/water, some sunlight/water, and a lot of sunlight/water. We will represent this set of levels as $\{0, 1, 2\}$, so now $s = 3$. Let us add two more factors as well: fertilizer and quality of soil, so now we have a total of 4 factors ($d = 4$). We can see the resulting OA in Table 2.2.

We have four factors and three levels, yet our experiment design only requires nine runs. If we considered a full factorial design, we would require $4^3 = 64$ runs. Clearly, using this design would be far more efficient. We can also better understand what a strength of two ($t = 2$) really

means - take any two columns, and in those columns, we will see that every two-tuple from the set $\{0, 1, 2\}$ appears exactly once.

Orthogonal arrays offer an efficient way to design experiments, optimizing for the number of runs required to determine the significance of some number of factors. This efficiency is been recognized and OAs are used for software testing (Pressman, 2010), experiment design, and cryptography.

2.3 Orthogonal Array Construction Techniques

A lot of the literature surrounding orthogonal arrays focuses more on existence proofs of OAs with certain properties than practical (or even impractical) ways to construct OAs. As useful as it is to know that these OAs exist and have these great properties, we may also want to know how to actually produce one. We will explore some construction techniques which provide an efficient algorithmic method to create an OA. Understanding these construction techniques will also provide an understanding of the current limitations of orthogonal arrays with respect to the constraints associated with these techniques.

2.3.1 The Bose Construction Technique

First, we will explore the Bose construction technique, which can produce the following orthogonal array

$$OA(p^2, p + 1, p, 2, 1) \tag{2.2}$$

where p is a prime number (Owen, 2013). Admittedly, this construction is very limited in the regard that the orthogonal array must have a prime squared number of runs and can only produce OAs of strength two.

Let us suppose that the OA is denoted as A, and A_{ij} is the number at the i^{th} row and j^{th} column

of A. We define the construction of column 0 as

$$A_{i0} = \left\lfloor \frac{(i-1)}{p} \right\rfloor \tag{2.3}$$

the construction of column 1 as

$$A_{i1} = (i-1) \bmod p \tag{2.4}$$

and the construction of the other columns as

$$A_{ij} = A_{i1} + (j-2)A_{i2} \quad \text{where } j > 1 \tag{2.5}$$

noting that the construction of the other columns depends on the values from the first two columns.

2.3.2 The Bush Construction Technique

The Bush construction technique is a generalization of the Bose construction technique that relaxes the constraint on the strength (t) and the constraint on the number of runs associated with an OA. With the Bush construction, we can construct the following OA (Owen, 2013):

$$OA(p^t, p+1, p, t, 1) \tag{2.6}$$

While we still have the constraint of needing a prime "base", we can now have an arbitrary strength, t, such that $1 \leq t < p$ (Owen, 2013). The number of runs, as a consequence, can be a somewhat arbitrary power of a prime. It is still a big constraint, but less constricting than the Bose construction.

The Bush construction relies on polynomials of the form (Owen, 2013):

$$\phi_i(x) = a_{i,t}x^{t-1} + \cdots a_{i,1}x + a_{i,0} \tag{2.7}$$

We construct these polynomials for every i such that $0 \leq i < p^t$, picking the coefficients so

$i = \sum_{l=0}^{t-1} a_{i,l} p^l$ (Owen, 2013). This is equivalent to taking a number, x, interpreting it in base p, and recording the coefficients, (including leading zeroes) up to degree t. Owen notes that we do not actually need to take this particular polynomial, we simply need p^t distinct polynomials, and interpreting the index in base p presents an easy way to do so. There are other ways of creating distinct polynomials, which would allow for a base that is any power of a prime, rather than a base that is just a prime. In order to do so, we must utilize Galois field arithmetic to generate polynomials and perform the other operations (Owen, 2013).

Once we define the polynomials, we can construct the columns $j = 0 \cdots p - 1$ of the OA as such:

$$A_{ij} = \phi_{i-1}(j-1) \bmod p \tag{2.8}$$

and construct the last column, p,

$$A_{ip} = (i-1) \bmod p \tag{2.9}$$

which is a special case (Owen, 2013).

Chapter 3

Orthogonal Arrays in Monte Carlo Integration

Now that we understand how to construct orthogonal arrays, we can apply some transformations to them so they can be used effectively in Monte Carlo integration as a sampler. Veach, in his thesis, briefly mentioned OAs for use in rendering (Veach, 1998). As far as we know, OAs have not been explored in Monte Carlo integration in the context of computer graphics.

We can normalize the points generated by an orthogonal array so that they fit within the domain for Monte Carlo sampling. We can also translate some of the terms used to describe OAs so that they make sense in the context of Monte Carlo integration. The number of runs in an orthogonal array is analogous to the total number of points in a point set, the factors are equivalent to the dimensionality of the point set, and the levels are analogous to the number of large strata used when sampling.

3.1 Owen's Transformations

Owen demonstrates a simple transformation that can be applied to points of an orthogonal array to make them suitable for Monte Carlo integration. This mostly entails randomizing them, normalizing them, and jittering them within strata.

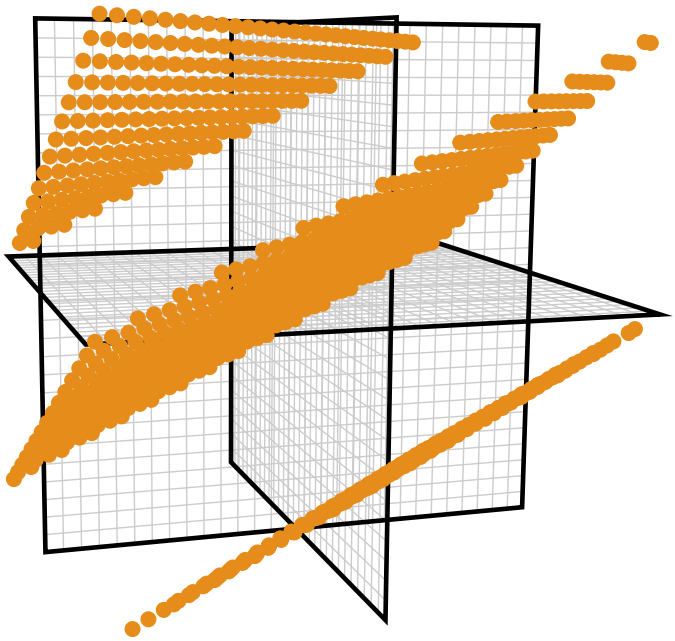

Figure 3.1: An OA with no randomization, highlighting the “planar flaw” (Jarosz u. a., 2019).

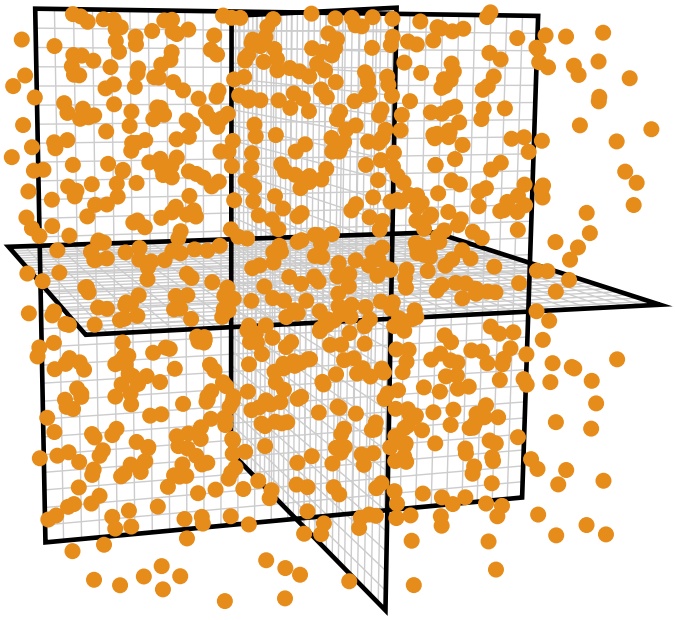

Figure 3.2: A randomized orthogonal array (Jarosz u. a., 2019).

It is crucial to randomize the orthogonal arrays, otherwise the points align along several planes, something that is referred to as the "planar flaw" (Owen, 2013). Refer to Figure 3.1 to see what the canonical arrangement of the Bose construction looks like, and compare with Figure 3.2 to see what the OA looks like once randomized.

Owen defines the randomized, jittered, and normalized arrangement of an OA as such:

$$X_{ij} = \frac{\pi_j(A_{ij}) + U_{ij}}{b} \tag{3.1}$$

where $1 \leq i \leq n$ and $1 \leq j \leq d$ (Owen, 2013). $\pi_1 \cdots \pi_d$ are defined as random permutations of the indices of the OA. In essence, this equation shuffles the points of an orthogonal array, jitters the points within their strata, and then normalizes them so that they fall within the $[0, 1)$ unit domain (per each dimension).

There are still improvements that can be made. For example, none of the shuffling in this transformation of the OA is correlated, and we do not meet the Latin hypercube/N-rooks constraint, which would improve the sampler's performance in Monte Carlo integration.

3.2 Practical Construction Algorithms

In this section, we aim to provide practical construction techniques for generating point sets that use the transformation provided by Owen. With these algorithms, we aim to be efficient, which entails low memory and computational requirements. These implementations are in-place, which means that any index and dimension of a sampler can be generated without requiring knowledge of other points, which lends itself to low memory overhead. Of course, this introduces some added complexity - how do we know where to place a point so that it meets strict stratification requirements and maintain the N-rooks guarantee if we don't know where the other points are positioned? In this section, we will explain how we can achieve that while presenting examples of code that we used to generate orthogonal array based point sets.

```
/*!
 * \brief Permute a number
 *
 * \param i The number to permute/the index of the permutation vector
 * \param l The desired size of the permutation vector
 * \param p The seed of the shuffle
 */
unsigned permute(unsigned i, unsigned l, unsigned p) {
  unsigned w = l - 1;
  w |= w >> 1;
  w |= w >> 2;
  w |= w >> 4;
  w |= w >> 8;
  w |= w >> 16;

  do {
    i ^= p;
    i *= 0xe170893d;
    i ^= p >> 16;
    i ^= (i & w) >> 4;
    i ^= p >> 8;
    i *= 0x0929eb3f;
    i ^= p >> 23;
    i ^= (i & w) >> 1;
    i *= 1 | p >> 27;
    i *= 0x6935fa69;
    i ^= (i & w) >> 11;
    i *= 0x74dcb303;
    i ^= (i & w) >> 2;
    i *= 0x9e501cc3;
    i ^= (i & w) >> 2;
    i *= 0xc860a3df;
    i &= w;
    i ^= i >> 5;
  } while (i >= l);
  return (i + p) % l;
}
```

Listing 3.1: The hashed permutation function as presented by Kensler, which allows for in-place permutation of an arbitrarily sized array with no memory overhead.

3.2.1 Background Listings

In this section, we will introduce some code snippets that will be used throughout this paper. These are useful snippets of code that we have either constructed or derived from other works.

The `permute` function produces an in-place or hashed permutation (Listing 3.1). It is equivalent to creating a list of l elements, and shuffling each element around, and retrieving the i^{th} element in that list. This function is taken from Kensler's technical report (Kensler, 2013).

```
 1  float randfloat(unsigned i, unsigned p) {
 2      i ^= p;
 3      i ^= i >> 17;
 4      i ^= i >> 10;       i *= 0xb36534e5;
 5      i ^= i >> 12;
 6      i ^= i >> 21;       i *= 0x93fc4795;
 7      i ^= 0xdf6e307f;
 8      i ^= i >> 17;       i *= 1 | p >> 18;
 9      return i * (1.0f / 4294967808.0f);
10  }
```

Listing 3.2: An efficient function to generate a pseudo-random number.

Libraries

For the purpose of this research, I wrote additional libraries to help with the verification of point sets and construction methods. One library, called `oars`, is an orthogonal array construction and verification library written in the Rust language. It contains functions to generate orthogonal arrays using the Bose and Bush construction techniques, it also has verification methods to check whether a given point set is valid with respect to the OA parameters that it should have. Additionally, `oars` has extensive benchmarking tools to check the performance of each constructor.

Another more general purpose library, `mcsampler`, is a Monte Carlo sampling library written in Rust. It contains a general interface for sampling and implements a number of sampling techniques, not just orthogonal array based sampling. The `mcsampler` library also contains utilities for checking the validity of point sets, whether points are properly stratified, and have the N-rooks guarantee (see Listing 3.3).

3.2.2 (Correlated) Multi-Jittering in 2D Projections

We have already discussed the Bose construction techniques. We propose a slight modification of the technique in order to allow for correlated multi-jittering, as was described by Kensler (2013), but in a manner that extends these properties to a higher-dimensional point set.

When constructing the first two columns of the point set, the Bose construction, as described

```rust
pub fn verify<T: OAInteger>(oa: &OA<T>) -> bool {
    if oa.points.ndim() != 2 {
        return false;
    }

    if oa.points.shape()[1] != oa.factors.to_usize().unwrap() {
        return false;
    }

    let col_combos =
        (0..oa.factors.to_u64().unwrap()).combinations(oa.strength.to_usize()
        .unwrap());

    // this iterator gives us every possible combination of columns
    for selection in col_combos {
        // tuple count holds the count for how many times each possible tuple is
        // seen
        let mut tuple_count: HashMap<u64, u64> = HashMap::new();

        // loop through the points and count up how many times we encounter
        // the tuple
        for i in 0..oa.points.shape()[0] {
            let mut tuple_index = 0;

            for (power, column) in selection.iter().enumerate() {
                tuple_index += (oa.points[[i, column.to_usize().unwrap()]] *
                    pow(oa.levels, power))
                    .to_u64()
                    .unwrap();
            }
            // set count to 1 if it doesn't exist, otherwise update the count
            *tuple_count.entry(tuple_index).or_insert(0) += 1;
        }

        // now verify that the hashmap has every possible combination, `index`
        // times
        for i in 0..oa
            .levels
            .to_u64()
            .unwrap()
            .pow(oa.strength.to_u32().unwrap())
        {
            // if the entry is not present in the array, set the count to 0
            if *tuple_count.entry(i).or_insert(0) != oa.index.to_u64()
                                                        .unwrap() {
                return false;
            }
        }
    }
    true
}
```

Listing 3.3: A method that verifies if a point set is a valid orthogonal array given a point set and the parameters for the orthogonal array, written in Rust.

by Equation (2.3) and Equation (2.4), essentially takes the index and breaks it down into its base p representation. The implication of using a base representation means that every index will yield unique strata. This is the key to being able to select strata in-place while ensuring that every point lands in different strata. We know that any two numbers a and b, such that $a \neq b$, will yield representations in any base such that a and b have different representations. In short, this means that different indices yield different combinations of strata.

This is a good start, because we have well-stratified points with respect to the coarse strata in the domain (there are p strata in each dimension), but we also want to maintain the N-rooks constraint, as well as multi-jittered, and correlated multi-jittered offsets. Within each of the p strata, there are p additional substrata in each dimension. The key to achieving the proper offsets is to select the correct substrata once a coarse stratum has been selected. Our strategy for selecting offsets entails using the stratum from another dimension to inform what the stratum ought to be for the current dimension.

In order to implement simple jittering, we can choose a substratum at random by permuting the index through all of the available substrata. This ensures that we pick a unique substratum for each point to meet the N-rooks constraint, but does not provide any correlation. If we want multi-jittered (MJ) sampling, we can be a bit more clever when choosing the offset function. We can permute through the substrata based on the coarse strata selected by some other dimension. For the first two dimensions, we base the substrata of the x dimension on the coarse stratum of the y dimension. For higher dimensions, we simply need a consistent method of selecting a stratum from another dimension.

We introduce a further complication in order to implement correlated-multi jittered (CMJ) sampling with respect to selecting the other dimension. We only enforce the CMJ constraint for "primary" pairs of dimensions, such as $(0, 1)$, $(2, 3)$, etc. This means that all of the cross-dimensional projections of the point set do not get the CMJ stratification. Instead, we perform the MJ offset for the cross-dimensional projections. For the CMJ offset, we select the same substrata based off of the coarse strata for every dimension, so that the shuffles within CMJ-enabled

```
float boseOA(int i, int j, int s, int p, Offset ot) {
    int Aij, Aik;
    i               = permute(i % (s*s), N, p * 0x51633e2d);
    int Ai0         = i % s;
    int Ai1         = i / s;
    if (j == 0) {
        Aij         = Ai0;
        Aik         = Ai1;
    } else if (j == 1) {
        Aij         = Ai1;
        Aik         = Ai0;
    } else {
        int k       = (j % 2) ? j-1 : j+1;
        Aij         = (Ai0 + (j-1) * Ai1) % s;
        Aik         = (Ai0 + (k-1) * Ai1) % s;
    }
    int stratum     = permute(Aij, s, p);
    int subStratum = offset(Aij, Aik, s, p * 0x68bc21eb, ot);
    float jitter    = randfloat(i, p * 0x02e5be93);
    return (stratum + (subStratum + jitter) / s) / s;
}

// Compute substrata offsets
int offset(int sx, int sy, int s, int p, OffsetType ot) {
    if (ot == J)   return permute(0, s, (sy * s + sx + 1) * p);
    if (ot == MJ)  return permute(sy, s, (sx + 1) * p);
    return permute(sy, s, p);   // Defaults to CMJ
}
```

Listing 3.4: An implementation of the Bose construction technique with various types of offsets. The parameter i represents the index of the point set, j is the "other" dimension to use to calculate offsets within strata, s is the number of coarse strata in each dimension, p is the effective seed to use for the shuffling and randomization, and ot is the offset type to apply when shuffling within coarse strata.

dimensions are identical, as implemented in the original 2D variant of CMJ (Kensler, 2013).

Listing 3.4 demonstrates an implementation of the Bose in-place sampler with selectable jittered, multi-jittered, and correlated-multi jittered offsets.

3.2.3 (Multi-)Jittering in t-D Projections

The Bush construction allows us to select an arbitrary strength (t), which equates to an arbitrary stratification guarantee. Where Bose was restricted to strength 2, and thus can only be stratified in 2D projections, the Bush construction allows us to expand beyond two dimensions and stratify in any t-dimensional projection.

The basic strategy is similar to what we did for the Bose construction. We first select a coarse

```
float bushOA(int i, int j, int s, int t, int p, Offset ot) {
    int N          = pow(s, t);
    i              = permute(i, N, p * 0x51633e2d);
    auto iDigits   = toBaseS(i, s, t);
    int stm        = N / s;     // s^(t-1)
    int k          = (j % 2) ? j - 1 : j + 1;
    int phi        = evalPoly(iDigits, j);
    int stratum    = permute(phi % s, s, p);
    int subStratum = offset(i, s, stm, p * 0x68bc21eb, ot);
    float jitter   = randfloat(i, p * 0x02e5be93);
    return (stratum + (subStratum + jitter) / stm) / s;
}

// Compute the digits of decimal value `i` expressed in base `b`
vector<int> toBaseS(int i, int b, int t) {
    vector<int> digits(t);
    for (int ii = 0; ii < t; i /= b, ++ii)
    digits[ii] = i % b;
    return digits;
}

// Evaluate polynomial with coefficients a at location arg
int evalPoly(const vector<int> & a, int arg) {
    int ans = 0;
    for (int l = a.size()-1; l >= 0; --l)
    ans = (ans * arg) + a[l];        // Horner's rule
    return ans;
}

// Compute substrata offsets
int offset(int i, int s, int numSS, int p, OffsetType ot) {
    if (ot == J) return permute(0, numSS, (i + 1) * p);
    return permute((i / s) % numSS, numSS, p);  // Defaults to MJ
}
```

Listing 3.5: An implementation of the Bush in-place construction technique with various offsets. The parameters are identical as described in Listing 3.4.

stratum based on the index of the sample re-interpreted in base p. We then select a substratum by assigning a new unique index from the original index. In this case, we elected to make the new index $i/p \bmod p^{t-1}$. Another transformation can be used, as long as every sample that falls into the same strata receives a unique index so we can ensure they all fall in different substrata. The standard jittering approach remains the same as before - select a random substratum for each sample.

In Listing 3.5, we have the in-place implementation of Bush with jittered and multi-jittered offsets. The functions `evalPoly` and `toBaseS` correspond to Equation (2.7), which we established is equivalent to transforming the index to base p, and reinterpreting it in base 10.

```
float cmjdND(int i, int s, int t, int p) {
    int N          = pow(s, t);
    i              = permute(i, N, p * 0x51633e2d);
    auto iDigits   = toBaseS(i, s, t);
    int stm1       = N / s;     // s^(t-1)
    int stratum    = permute(iDigits[j], s, p);
    auto pDigits   = allButJ(iDigits, j);
    int subStratum = evalPoly(pDigits, s);
    subStratum     = permute(sstrata, stm1, p * 0x68bc21eb);
    float jitter   = randfloat(i, p * 0x02e5be93);
    return (stratum + (subStratum + jitter) / stm1) / s;
}

// Copy all but the j-th element of vector in
vector<int> allButJ(const vector<int> & in, int omit) {
    vector<int> out(in.size()-1);
    copy(in.begin(), in.begin() + omit, out.begin());
    copy(in.begin() + omit + 1, in.end(), out.begin() + omit);
    return out;
}
```

Listing 3.6: An implementation of the CMJND construction technique. i represents the index of the point within the point set, s is the number of coarse strata in each dimension, t is the strength of the resulting point set (and also the cross-stratification guarantee), and p is the seed for the hashed pRNG methods used by the method (such as the hashed permutation and random number generator functions).

3.2.4 CMJND

We constructed a generalization of Kensler's CMJ technique that relaxes the constraint on the base that is present for the Bush and Bose construction techniques and allows an arbitrary base that is not necessarily a prime number. The downside to this method is that it constructs a full factorial design, an orthogonal array where $t = d$, so the stratification is equal to the dimensionality of the points (while still holding the N-rooks constraint).

We achieve this design by converting the index to base p, which essentially gives us the selection of each stratum. We can permute these factors in order to yield a random shuffling of the points. The trick here is to use the coefficient for the current dimension (use the first coefficient to calculate a point in the first dimension, and so on) to calculate the strata, and use the rest of the coefficients to calculate the substrata. We have already established that this strategy will create a unique selection of strata because the uniqueness of the indices holds regardless of which base they are represented in.

3.2.5 PBRT Implementation

In order to test the performance of these sampling techniques on rendered scenes, we opted to use PBRT, an educational renderer that is commonly used in research. In order to do this, we had to integrate the sampling code we had into PBRT's sampling interface.

PBRT provides two types of sampling interfaces: the per-pixel sampler and the global sampler. The per-pixel sampler provides a convenient interface for generating sample sets for each pixel. Every single pixel in the image will receive its own set of sampling points, all of which are unaware of the sampling points in other pixels. Sometimes, this is not sufficient, and a sampling point set should be well distributed over the entire image. For this, PBRT provides a global sampler. In the global sampler, one set of sampling points are used over a whole image, so samples that land in different pixels are "aware" of each other.

We implemented the CMJND, Bush, and Bose sampling strategies in PBRT with the per-pixel sampler due to performance and ease of implementation. The global sampler in PBRT requires an instant reverse lookup of a pixel location to the index of a point. For example, if we have a pixel at location (x, y), given our set of samples, we need to know which index i results in a sample point that will land in the pixel (x, y). The only feasible way to implement this is to know how to create a mapping of pixel locations to indices in the sampling point set. Because we shuffle points using Kensler's permutation function, it would seem that we could compute the reverse of the hash and get a mapping that yields the pixel locations and indices in an efficient manner. Unfortunately, because the function relies on a destructive modulo operator, the method is not reversible. In this case, the only remaining option is to calculate all of the points and store a mapping of the points and pixel locations in memory, which is too inefficient for rendering.

Chapter 4

Results

We have explored the theory behind OA sampling and different efficient construction techniques for different variations of OA sampling. It is also important to test for and establish both the theoretical bounds and empirical performance of the samplers. In this section we will present the theoretical bounds of OA sampling as well as real world performance and results from empirical analyses.

4.1 Theoretical Bounds

Functions can exhibit variation in a number of dimensions, and we ideally want our sampler to be well-stratified in the dimensions that a function exhibits such variance. For example, if we have an additive one-dimensional function, any sampler that meets the N-rooks constraint will perform similarly, even identically when looking at the asymptotic convergence rates.

We can take this idea and get finer bounds on convergence rates if we know the dimensions in which the integrand varies and in which our sampler is well-stratified. Analytic variance analyses have been performed in several works which show the theoretical bounds on Monte Carlo integration in particular scenarios (Jarosz u. a., 2019).

We know that naive random sampling always nets an asymptotic bound of $O(N^{-1})$, regardless of the function, its variance properties, or the dimensionality. Samplers that are well-stratified

Table 4.1: The convergence rate improvement (b in $O(N^{-1-b})$) as a function of the dimensionality and smoothness of the integrand for various samplers. The 1- and t-additive integrands are d-dimensional, where $t < d$. Best case for each integrand is bold. (Jarosz u. a., 2019)

Sampler	**Convergence rate improvement** b							
Integrand:	d-dim.		t-dim.		t-additive		1-additive	
Discontinuity:	C^1	C^0	C^1	C^0	C^1	C^0	C^1	C^0
Random	0	0	0	0	0	0	0	0
d-stratified	$\mathbf{2/d}$	$\mathbf{1/d}$	$2/d$	$1/d$	$2/d$	$1/d$	$2/d$	$1/d$
Padded t-stratified	0	0	$\mathbf{2/t}$	$\mathbf{1/t}$	0	0	0	0
Padded t-stratified+LH	0	0	$\mathbf{2/t}$	$\mathbf{1/t}$	0	0	**2**	**1**
OA strength-t	0	0	$\mathbf{2/t}$	$\mathbf{1/t}$	$\mathbf{2/t}$	$\mathbf{1/t}$	$2/t$	$1/d$
OA strength-t+LH	0	0	$\mathbf{2/t}$	$\mathbf{1/t}$	$\mathbf{2/t}$	$\mathbf{1/t}$	**2**	**1**

in d dimensions with d dimensional integrands that are C1 discontinuous have a convergence rate of $O(N^{-1-2/d})$ (Jarosz u. a., 2019; Owen, 2013). The same scenario with a C0 discontinuous integrand yields an asymptotic convergence rate of $O(N^{-1-1/d})$ (Jarosz u. a., 2019). Figure 4.1 presents a selection of scenarios and the best case convergence bounds, taken from a preprint of an EGSR paper (Jarosz u. a., 2019).

We can see that orthogonal array sampling presents some distinct advantages. One big advantage is that orthogonal array sampling can handle variation in any d-projection. If we know a function has variation in d dimensions, we do not need to do any clever mapping to make sure the variation lines up with the dimensions of the sampler that are well-stratified, because every $d = t$ dimensional projection is well-stratified with OA sampling. This provides an even bigger advantage for functions that have cross-dimensional variation. Suppose we have some function that exhibits variance in multiple arbitrary d dimensional projections. OA sampling can handle that as well, since we already have well stratified-projections.

The downside is that the variation needs to match the strength of the OA. Suppose we have an OA with $t = 3$, but the integrand is four-dimensional. This will lead to the OA having a performance of $O(N^{-1})$, which is the same as random sampling, which is not ideal. Sobol sampling does not have this issue, as it maintains good stratification in multiple dimensions. OA sampling

does not perform optimally when the dimensionality of the integrand is less than the strength of the OA. While the performance does not degrade as much as when the dimensionality is higher, it still does not offer as dramatic of an improvement as when the dimensionality and the strength match up. OA sampling uniquely poised to perform well when functions exhibit variance in all combinations of dimensions, for example:

$$f(x, y, z) = f(x, y) + f(y, z) + f(x, z) \tag{4.1}$$

While this is a unique case, there is no other sampler better equipped to handle variation in this fashion.

4.2 Empirical Results with Analytic Integrands

We modified the Empirical Error Analysis (EEA) (Subr u. a., 2016) tool to support multi-dimensional integrands and added a number of samplers to it in order to test the variance of various functions and samplers. This tool allowed us to see if our samplers matched the theoretical expectations that we laid out, and the results seem to match up.

Prior work has mostly focused on variation in two dimensions, and the variance analyses presented in this paper offer results that focus on variance in multiple dimensions. We crafted several functions that used one-dimensional functions as the basis, which were then combined to form higher dimensional functions in a way that allowed us to control the dimensionality (and the dimensionality of the variance) as well as the discontinuity of the resulting functions. Our one-dimensional basis functions consisted of a Gaussian,

$$g^{\infty}(r) = \exp(-r^2/(2\sigma^2)) \tag{4.2}$$

a C1 discontinuity,

$$g^{1}(r) = 1 - \text{linearStep}(r, r_{\text{start}}, r_{\text{end}}) \tag{4.3}$$

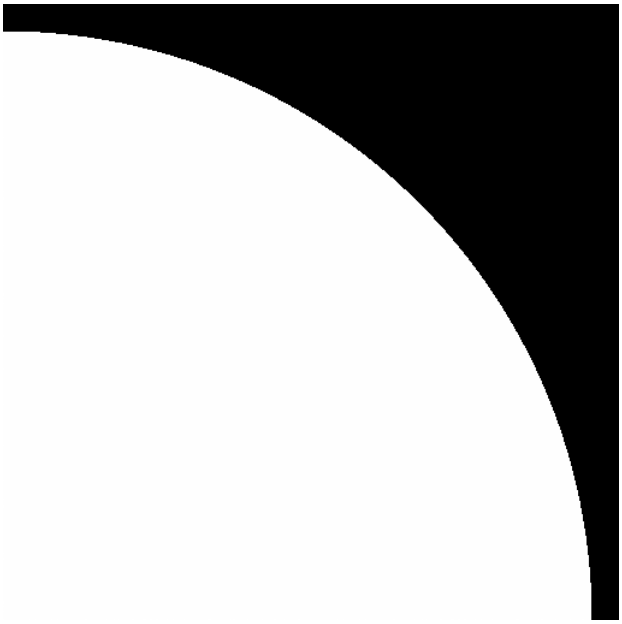

Figure 4.1: A visual representation of the g^0 function

and a C0 discontinuity,

$$g^0(r) = 1 - \text{binaryStep}(r, r_{\text{end}}) \tag{4.4}$$

all of which were clipped so that the center of the functions began in the origin of the domain. The clipping made the function asymmetrical, which is a desired property for these tests. They all share the parameters $r_{\text{end}} = {}^3/\pi$, $r_{\text{start}} = r_{\text{end}} - 0.2$, and $\sigma = {}^1/3$.

We built several integrands, both additive and multiplicative, by constructing variance in various projections. The way we achieved this was by constructing d dimensional functions that were additive or multiplicative in t dimensions, so an additive function with $t = 2$ and $d = 3$ would resemble Equation (4.1).

We can define a more generalized formula to describe the functions that we constructed in a dimensionally generic way. The additive and multiplicative variants are defined in Equations

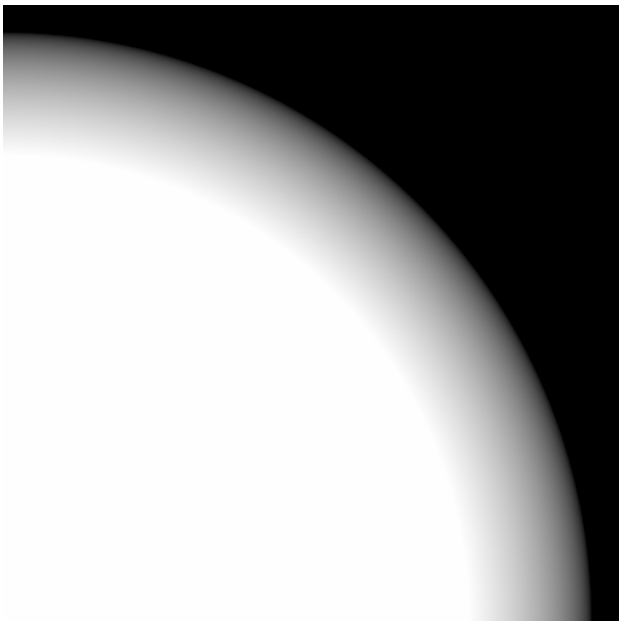

Figure 4.2: A visual representation of the g^1 function

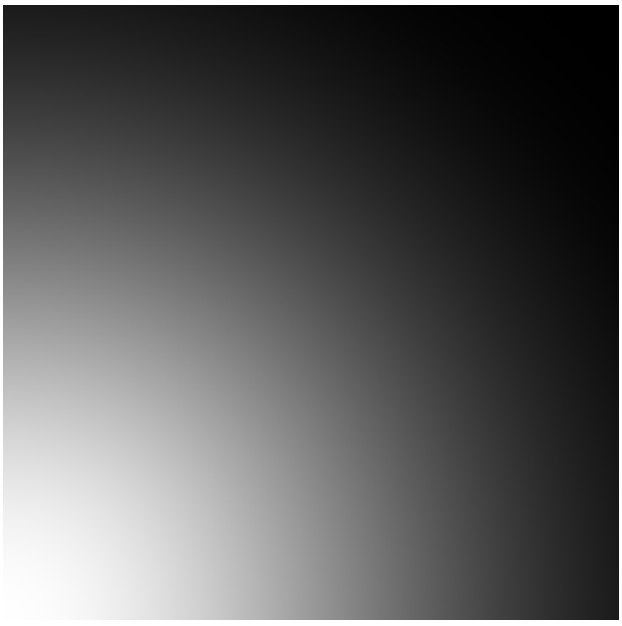

Figure 4.3: A visual representation of the g^∞ function

(4.5) and (4.6), respectively. Equation (4.1) would be considered a function of the type $f^D_{3,2+}$.

$$f^D_{d,t+}(p_1, \cdots, p_d) = \sum_{i_1=1}^{d} \cdots \sum_{i_t=i_{t-1}+1}^{d} f^D_t(p_{i_1}, \cdots, p_{i_t}) \tag{4.5}$$

$$f^D_{d,t\times}(p_1, \cdots, p_d) = \prod_{i_1=1}^{d} \cdots \prod_{i_t=i_{t-1}+1}^{d} f^D_t(p_{i_1}, \cdots, p_{i_t}) \tag{4.6}$$

We tested our samplers on these integrals. In addition to the samplers that have already been discussed, we tested several variations of padded samplers, such as CMJ padded, and the padded $(0, 2)$-sequence sampler that is present in PBRT (Pharr u. a., 2016).

In Figure 4.4, we present empirical convergence graphs for an extensive set of variance samplers with various functions. The legend of these graphs contain a sampler name, along with the convergence rate of the sampler, such that the number provided, x, corresponds to the slope $O(N^x)$.

We can see that the most significant performance improvements occur when the dimensionality of the integrand, d, matches the strength of the OA, t. The 1-additive integrand provides a scenario where the OA samplers can achieve a steeper convergence rate than even the QMC samplers, a significant result in Monte Carlo integration. Unfortunately, when $t \neq d$ for non-additive integrands, the performance of the OA samplers degrades to roughly the performance that we expect of naive random sampling. Sobol and Halton, which offer higher dimensional stratification than OA sampling, perform well in these circumstances.

4.3 Empirical Results with Rendered Scenes

On top of performing empirical analyses with analytic integrands, we tested our samplers in real-world rendering scenarios. We implemented the Bose and Bush samplers in PBRT as per-pixel samplers. In PBRT, samplers can either be implemented as "global" samplers, where the samples are distributed across the whole image, or as "per-pixel" samplers, where each pixel gets its own set of samples, which are unaware of the samples in other pixels. Global sampling tends to accentuate structural artifacts, such as the ones that are common with the Sobol sampler.

To measure the performance of our samplers, we computed a ground truth render for each scene, using the PBRT-implemented uniform random sampler with four orders of magnitude more samples than the rest of the samplers. This gave us a reasonably confident estimate of what image the samplers should be converging to. We computed the MSE on each image to see how much error each sampler had at comparable sample counts.

We created three scenes. The first, dubbed CornellBox, is the classic Cornell box scene and uses anti-aliasing, soft shadows, and depth of field with a direct lighting integrator to create a 7D integrand (Pharr u. a., 2016). The second, dubbed BlueSpheres, is a scene that utilizes motion blur, depth of field, antialiasing, and inter-reflections with the path tracing integrator to create a 9D integrand. The third and most complex scene, dubbed Barcelona, is a scene of an apartment with a swimming pool featuring global illumination with the path tracing integrator to yield a 43D integrand.

We found that the benefits of using OA sampling had diminishing returns as the scenes got more complex. Visually, it is difficult to see the difference between samplers as more complex scenes introduce more elements and noise. We computed the MSE for each scene with each sampler and cropped specific zoom-ins for regions where we felt that there was a notable difference in performance between the samplers. These comparisons and MSE figures are in Figure 4.5. Generally, we found our OA sampling methods to be the highest performing non-QMC

methods, except in the CornellBox scene, where OA sampling outperforms QMC sampling. Notice the artifacts from the Sobol sampler in the soft shadows of the CornellBox scene. These artifacts are distracting, very noticeable, and likely more pronounced because the Sobol sampler in PBRT is a global sampler, as mentioned earlier. It is likely that some of this artifacting could be rectified by using a per-pixel Sobol sampler.

We also conducted variance analyses using different sample sizes on a single pixel for the CornellBox and BlueSpheres scenes. The pixels were from the regions where the zoom-ins and crops were placed, highlighting areas in each scene where we thought the samplers had significant differences. In Figure 4.6, we can see that Bose outperforms all of the other samplers in CornellBox, and in the BlueSpheres scene, we get the expected results - the Bose sampler outperforms all of the non-QMC samplers.

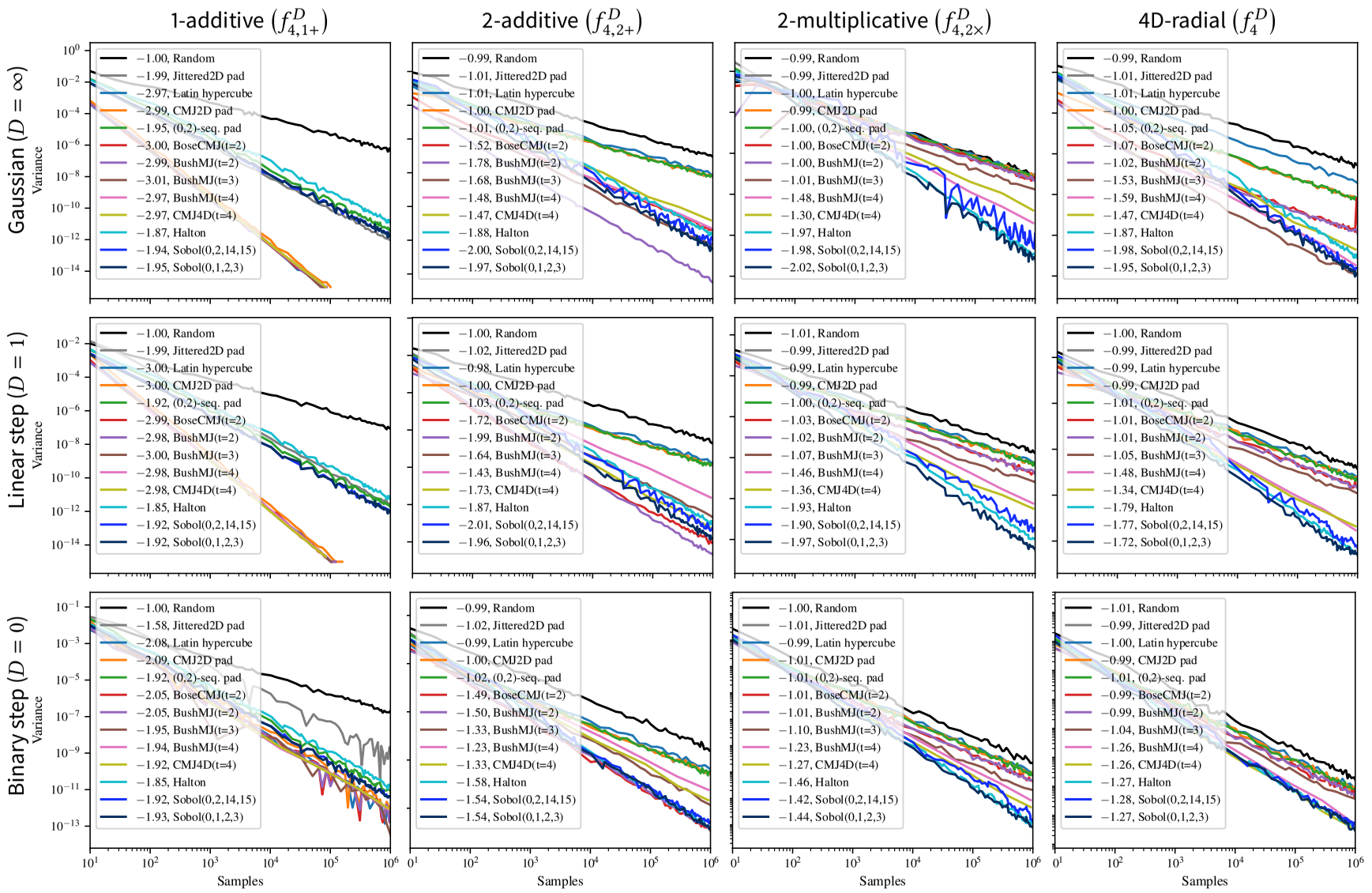

Figure 4.4: Variance behavior of 13 samplers on 4D analytic integrands of different complexity (columns) and continuity (rows). We list the best-fit slope of each technique, which generally matches the theoretically predicted convergences rates (4.1). Our samplers always perform better than traditional padding approaches, but are asymptotically inferior to high-dimensional QMC sequences for general high-dimensional integrands. When strength $t < d$ (right two columns), convergence degrades to -1, but higher strengths attain lower constant factors. (Jarosz u. a., 2019)

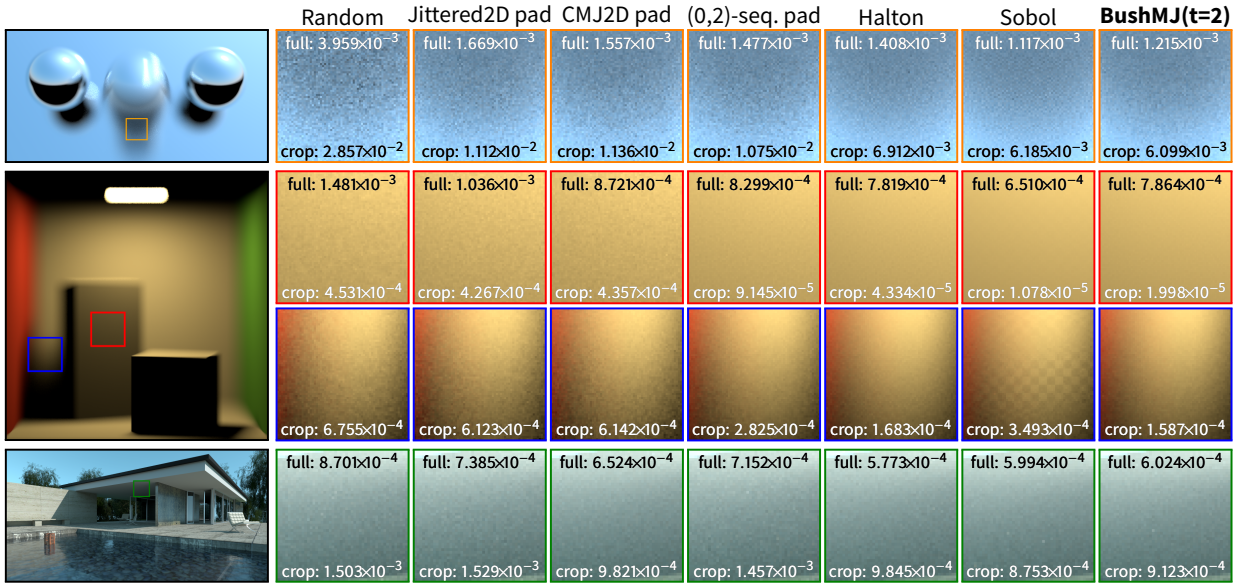

Figure 4.5: The BLUESPHERES, CORNELLBOX, and BARCELONA scenes feature different combinations of pixel antialiasing, DoF, motion blur, and several bounces of indirect illumination for combined integrands of 9D, 7D, and 43D respectively. The relative MSE numbers for the entire image (top) and each inset (bottom) show that our OA-based sampling technique is able to outperform 2D padded samplers (first 4 columns), and is close to the quality of multi-dimensionally stratified global samplers like Halton and Sobol (Jarosz u. a., 2019).

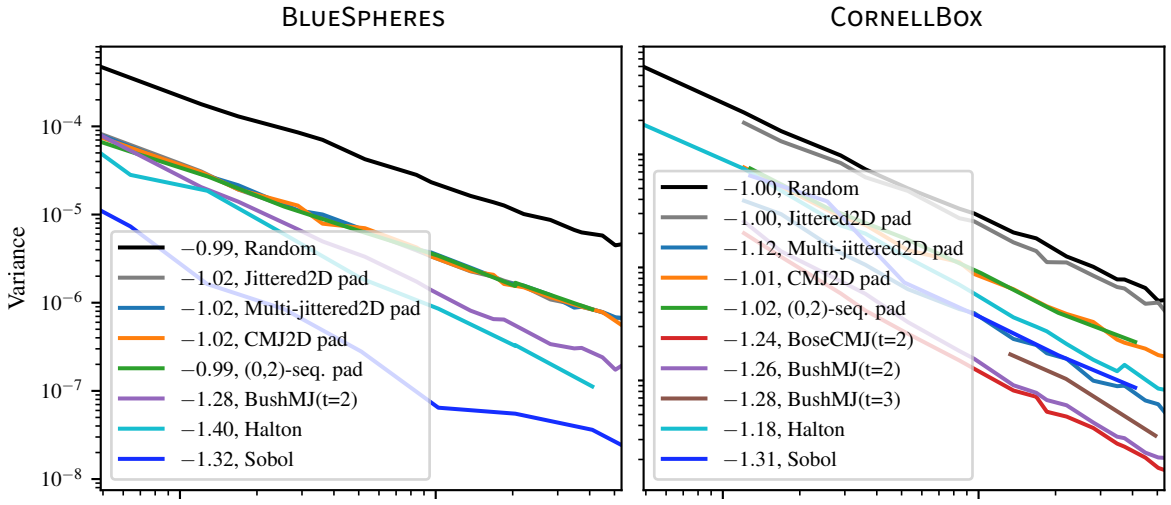

Figure 4.6: Variance behavior and best-fit slope of various samplers for a pixel in the yellow inset in BLUESPHERES and the blue inset of CORNELLBOX in 4.5. Our samplers always perform better than traditional padding approaches and even outperform the global Halton and Sobol samplers in CORNELLBOX (Jarosz u. a., 2019).

Chapter 5

Conclusion

The introduction of orthogonal arrays along with practical construction techniques gives us a way to efficiently generate a sampling method for Monte Carlo integration that performs well with high-dimensional integrands. It addresses a problem that has been relatively unexplored in computer graphics. The usage of orthogonal arrays is relatively new to the field, and it seems that OA sampling could offer many solutions for Monte Carlo rendering. Though OA sampling has shown promising results, it is not without its flaws, and there are many things that can be done to improve on the existing research.

5.1 Limitations

The Bose and Bush construction have a rather daunting constraint: they require a prime base (Owen, 2013; Bose und Bush, 1952; Bush, 1952). With larger and larger sample sizes, finding a prime base can be extremely inconvenient. At lower sample sizes, it can cause an issue because the dimensionality of the point set is limited by the prime base, so for extremely complex scenes, for example, we may not be able to use small sample sizes in our point sets.

We have also already discussed the fact that OAs are optimized for integrands with a dimensionality d that match the strength of an OA t. One advantage that the Sobol and Halton samplers have is that they are well-stratified in many dimensions and do not require this kind of fine-tuning for particular integrands. Sobol and Halton also have the advantage of being progressive

sequences (Christensen u. a., 2018; Halton, 1964; Sobol, 1967). As it stands with the construction techniques we have introduced, we need to know the number of samples required in advance and cannot progressively distribute samples in the manner achieved by Halton and Sobol.

5.2 Future and Related Work

There is some active research in the areas that we have pointed out as good areas of improvement for Monte Carlo integration. On the point of the dimensionality mismatch between OA strength and integrand variance, there is a promising area of research called **Strong Orthogonal Arrays** (SOAs) that could solve this issue by providing better stratification guarantees for multiple levels of projections (He und Tang, 2012). Ideally, we could generate an SOA with $t = 4$, for example, and that would yield good stratification for $1, 2, 3, 4$-d projections. He and Tang have already proven that it is possible to generate an SOA from an OA with a larger dimensionality (He und Tang, 2012).

The stratification guarantees from SOAs are much denser than the guarantees of OAs. Consider a strength t. Now consider every combination of numbers that adds to t. We will say this set of numbers is called E. We can take this set and use these numbers as exponents for the base of the OA (which denotes the number of coarse strata). Suppose that there are $|E|$ elements in E, so the stratification guarantees for an SOA of strength t are $s^{E_0} \times s^{E_1} \times \cdots \times s^{E_{|E|}}$. As a more practical example, consider $t = 3$. We have the combinations: $0 + 0 + 3$, $0 + 1 + 2$, $1 + 1 + 1$, etc. Those sets correspond to the stratification guarantees: $1 \times 1 \times s^3$, $1 \times s \times s^2$, $s \times s \times s$, etc (He und Tang, 2012). This is not a comprehensive list, as the other combinations of these numbers shuffled around at different dimensions are included in the list of stratification guarantees that the SOA provides. As we can see, these stratification guarantees are quite dense and seem similar to the concept of elementary intervals (Niederreiter, 1988; Sobol, 1967). The `oars` library discussed earlier also contains utilities for verifying whether a point set is a valid SOA.

There currently seem to be no efficient or easily implementable methods to construct strong orthogonal arrays, and existing methods require one to generate a larger orthogonal array and

"collapse" it into an SOA (He und Tang, 2012), or use generalized orthogonal arrays (Lawrence, 1996). A working, efficient, SOA construction method could yield a lot of gains for Monte Carlo integration performance, providing the advantages we get from OAs for high dimensional functions but relaxing constraints while possibly increasing the benefits that we get from high-dimensional stratification.

Bibliography

[Bose und Bush 1952] Bose, Raj C. ; Bush, K. A.: Orthogonal Arrays of Strength Two and Three. In: *The Annals of Mathematical Statistics* 23 (1952), Dezember, Nr. 4, S. 508–524. – ISSN 0003-4851

[Bush 1952] Bush, K. A.: Orthogonal Arrays of Index Unity. In: *The Annals of Mathematical Statistics* 23 (1952), Nr. 3, S. 426–434. – ISSN 00034851

[Christensen u. a. 2018] Christensen, Per ; Kensler, Andrew ; Kilpatrick, Charlie: Progressive Multi-Jittered Sample Sequences. In: *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 37 (2018), Juli, Nr. 4, S. 21–33. – ISSN 0167-7055

[Cook u. a. 1984] Cook, Robert L. ; Porter, Thomas ; Carpenter, Loren: Distributed Ray Tracing. In: *Computer Graphics (Proceedings of SIGGRAPH)* 18 (1984), Juli, Nr. 3, S. 137–145. – ISSN 00978930

[Geyer 2014] Geyer, J. A.: *Different Formulations of the Orthogonal Array Problem and Their Symmetries*, Air Force Institute of Technology, Dissertation, 2014

[Halton 1964] Halton, J. H.: Algorithm 247: Radical-inverse Quasi-random Point Sequence. In: *Commun. ACM* 7 (1964), Dezember, Nr. 12, S. 701–702. – URL `http://doi.acm.org/10.1145/355588.365104`. – ISSN 0001-0782

[He und Tang 2012] He, Yuanzhen ; Tang, Boxin: Strong Orthogonal Arrays and Associated Latin Hypercubes for Computer Experiments. In: *Biometrika* 100 (2012), Dezember, Nr. 1, S. 254–260. – ISSN 1464-3510

[Jarosz u. a. 2019] Jarosz, Wojciech ; Enayet, Afnan ; Kensler, Andrew ; Kilpatrick, Charlie ; Christensen, Per: Orthogonal Array Sampling for Monte Carlo Rendering. In: *Computer Graphics Forum* (2019), S. 12

[Kajiya 1986] Kajiya, James T.: The Rendering Equation. In: *SIGGRAPH Comput. Graph.* 20 (1986), August, Nr. 4, S. 143–150. – ISSN 0097-8930

[Kensler 2013] Kensler, Andrew: Correlated Multi-Jittered Sampling / Pixar Animation Studios. März 2013 (13-01). – Forschungsbericht

[Kollig und Keller 2002] Kollig, Thomas ; Keller, Alexander: Efficient Multidimensional Sampling. In: *Computer Graphics Forum (Proceedings of Eurographics)* 21 (2002), September, Nr. 3, S. 557–563. – ISSN 0167-7055

[Lawrence 1996] Lawrence, K. M.: A Combinatorial Characterization of (t,m,s)-Nets in Base b. In: *Journal of Combinatorial Designs* 4 (1996), Nr. 4, S. 275–293

[Loeve 1977] Loeve, Michel: *Probability Theory 1*. Springer-Verlag New York, 1977. – ISBN 978-1-4684-9464-8

[Niederreiter 1988] Niederreiter, Harald: Low-discrepancy and low-dispersion sequences. In: *Journal of Number Theory* 30 (1988), Nr. 1, S. 51 – 70. – URL `http://www.sciencedirect.com/science/article/pii/0022314X8890025X`. – ISSN 0022-314X

[Owen 1997] Owen, Art B.: Monte Carlo Variance of Scrambled Net Quadrature. In: *SIAM Journal on Numerical Analysis* 34 (1997), Oktober, Nr. 5, S. 1884–1910. – ISSN 0036-1429

[Owen 2013] Owen, Art B.: *Monte Carlo Theory, Methods and Examples*. To be published, 2013

[Pharr u. a. 2016] Pharr, Matt ; Jakob, Wenzel ; Humphreys, Greg: *Physically Based Rendering: From Theory To Implementation*. 3. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2016. – ISBN 978-0-12-800645-0

[Pressman 2010] Pressman, Roger: *Software Engineering: A Practitioner's Approach*. 7. New York, NY, USA : McGraw-Hill, 2010. – ISBN 0-07-337597-7

[Shirley 1991] Shirley, Peter: Discrepancy as a Quality Measure for Sample Distributions. In: *Proceedings of Eurographics*. Amsterdam, North-Holland : Eurographics Association, 1991, S. 183–194. – ISSN 1017-4656

[Singh und Jarosz 2017] Singh, Gurprit ; Jarosz, Wojciech: Convergence Analysis for Anisotropic Monte Carlo Sampling Spectra. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 36 (2017), Juli, Nr. 4, S. 137:1–137:14. – ISSN 0730-0301

[Sobol 1967] Sobol, I.M: On the distribution of points in a cube and the approximate evaluation of integrals. In: *USSR Computational Mathematics and Mathematical Physics* 7 (1967), Nr. 4, S. 86 – 112. – URL http://www.sciencedirect.com/science/article/pii/0041555367901449. – ISSN 0041-5553

[Subr u. a. 2016] Subr, Kartic ; Singh, Gurprit ; Jarosz, Wojciech: Fourier Analysis of Numerical Integration in Monte Carlo Rendering: Theory and Practice: Understanding Estimation Error in Monte Carlo Image Synthesis. In: *ACM SIGGRAPH Course Notes*, ACM Press, Juli 2016, S. 10. – ISBN 978-1-4503-4289-6

[Veach 1998] Veach, Eric: *Robust Monte Carlo Methods for Light Transport Simulation*, Stanford University, Dissertation, 1998