Single-pass stratified importance resampling

by

Ege Ciklabakkal

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

© Ege Ciklabakkal 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Simulating the behavior of light to produce realistic images involves evaluation of difficult integrals. Monte Carlo integration is a general method to estimate such integrals and is therefore widely used in physically based rendering. Being a stochastic method, it requires generation of random samples and evaluation of the integrand at those samples. This process can be computationally expensive, especially when samples need to be generated from arbitrary distributions and the scene is complex. When the number of samples used in the estimation is inadequate, the resulting renderings are not converged and contain (white) noise. Much effort has gone into investigating ways to efficiently reduce this noise. In this thesis, we focus on a particular method to reduce the noise called resampling, point out its deficiencies, and propose novel procedures to efficiently address them.

Resampling is the process of selecting from a set of candidate samples to achieve a distribution (approximately) proportional to a desired target. Recent work has revisited its application to Monte Carlo integration, yielding powerful and practical importance resampling methods. One drawback of these methods is that they cannot generate stratified (more evenly distributed) samples efficiently. There are two main reasons for the lack of stratification: first, the discrete sampling method used for resampling fails to preserve the stratification; second, resampling doesn't consider the placement of high-dimensional candidates in their domain. For the first issue, although the common inverse cumulative distribution (CDF) sampling is successful in maintaining the stratification, the method requires a precomputation step where a data structure is built before the sampling step. We instead propose a novel algorithm which yields the same result as conventional inverse CDF sampling, in a single pass over the candidates. The algorithm traverses the candidate list adaptively from both ends, without needing to store them. For the second issue, we introduce an ordering among the high-dimensional candidates by generating them along a space-filling curve. Aside from stratification, we also show how blue-noise error distribution can be achieved for better perceptual quality. The resulting method is showcased on various resampling-based rendering problems.

Acknowledgements

Firstly, I would like to thank my supervisor, Toshiya Hachisuka for giving me the opportunity to work in his group and providing much valuable advice from day one. I would also like to thank Adrien Gruson, Iliyan Georgiev and Derek Nowrouzezahrai for their discussions and help during our research project which forms the basis of this thesis. They lead me through our work and I am lucky to learn from such distinguished researchers.

I thank my parents for their constant support throughout my studies.

Finally, the scenes in Figs. 1.1, 5.2 and 5.3 are from Benedikt Bitterli's rendering resources [3]. The scene in Fig. 5.4 comes from PBRT's resources. The models used in Fig. 5.5 were obtained from Artec 3D.

Table of Contents

List of Figures viii

1 Introduction 1

1.1 Contributions . . . 4

1.2 Organization . . . 5

2 Monte Carlo Integration 6

2.1 Basic Monte Carlo Integration . . . 6

2.1.1 Random Variable . . . 7

2.1.2 Expected Value . . . 8

2.1.3 Variance . . . 9

2.1.4 The Monte Carlo Estimator . . . 10

2.1.5 Monte Carlo Rendering . . . 11

2.2 Variance Reduction . . . 12

2.2.1 Importance Sampling . . . 13

2.2.2 Multiple Importance Sampling . . . 14

2.2.3 Stratified Sampling . . . 15

2.3 Quasi-Monte Carlo . . . 18

2.3.1 Discrepancy . . . 19

2.3.2 Radical inversion and Halton Sequence . . . 20

3 Sampling **22**

3.1 Inverse CDF Sampling . . . 22
3.2 Rejection Sampling . . . 24
3.3 Weighted Reservoir Sampling . . . 25
3.4 Resampled Importance Sampling . . . 28
3.5 Blue-Noise Dithered Sampling . . . 30
3.6 Stratified Sampling Using the Hilbert Curve . . . 32

4 Single-Pass Stratified Importance Resampling **34**

4.1 Lack of Stratification in Resampling . . . 34
4.1.1 Resampling in Monte Carlo Integration . . . 34
4.1.2 Stratification of Sampled Indices . . . 35
4.1.3 Impact of Candidate Order . . . 36
4.1.4 Problem Statement and Overview . . . 37
4.2 Single-pass Bidirectional CDF Sampling . . . 38
4.2.1 Taking Multiple Samples . . . 39
4.3 Ordering Candidates on a Space-Filling Curve . . . 42
4.4 Stratified Resampling Algorithm . . . 42
4.4.1 Blue-noise Error Distribution . . . 43

5 Results **44**

5.1 Single Scattering in Media . . . 44
5.2 Direct Illumination . . . 45
5.3 Resampled Multiple Importance Sampling . . . 46
5.4 Higher-dimensional Integration . . . 46
5.5 Convergence . . . 47

6 Conclusions **53**

References **55**

Appendix **59**

List of Figures

1.1 Stochastic nature of Monte Carlo estimation results in noise in output images when the number of rays / samples per pixel is insufficient. Rendering this kitchen scene by casting 16384 primary rays per pixel (rpp) produces a far more converged output than using 64 rpp. 2

2.1 Local geometry for scattering at $\mathbf{x}$. 12

2.2 16 random samples generated by uniform random sampling and stratified sampling with 16 uniform strata. Note the gaps and clumps present in purely random case. 16

2.3 First 256 points of the Halton sequence and the Hammersley point set in 2D. 21

3.1 Comparison of a 1D random mask with decorrelated pixel values and a 1D blue-noise dither mask with negatively correlated pixels. The random mask displays white noise properties: It has a flat frequency spectrum meaning that the energy is distributed uniformly over all frequency bands. In contrast, blue-noise has low energy in low frequencies. 31

3.2 Generation of the Hilbert curve. The curve has a resolution of 2^m in each dimension. 33

4.1 Histogram comparison between reservoir sampling with rescaled 1D Sobol (i.e., van der Corput) samples (a) and M-dimensional Sobol samples (b), and inverse CDF sampling with 1D independent samples (c) and 1D Sobol samples (d). The candidates are regularly-spaced and sorted $(0.5, \ldots, 49.5)$. Sampling is done proportionally to Gaussian weights ($\mu = 25$, $\sigma = 10$). Neither reservoir sampling variant retains the input's stratification; inverse CDF sampling does, producing a substantially lower L_2 histogram error. . 36

4.2 The ordering of resampling candidates can substantially affect the stratification of the output samples. (a) We generate $M = 8,192$ Halton samples on the unit square and weight them by a 2D Gaussian target function. We then select $N = 256$ of them via stratified inverse CDF sampling. (b) Processing the candidates in their Halton-sequence generation order produces no stratification in the output. (c) Ordering the candidates using our proposed method (Section 4.3) yields stratified unit-square samples. 37

4.3 Overview of our algorithm. Sorted 1D stratified samples are mapped onto a space-filling curve to produce stratified candidates y_k on the unit hypercube. We resample these candidates along the curve proportionally to their associated weights using our on-the-fly bidirectional CDF sampling. Stratifying the resampling input u yields well-distributed output samples x in the integration domain. 38

4.4 Variance analysis of 1D importance resampling using our bidirectional CDF method (Alg. 6). (a) Candidates are generated with density p and resampled according to $\hat{q}$ which better matches the integrand f. (b) Plots of integral-estimation variance as a function of sample count N, with $M = 4N$ candidates. When the candidates and input samples are independent, subset-based resampling (solid orange) yields lower variance than always resampling from the entire candidate set (solid blue), in line with the theory in Section 4.2.1. Stratifying the candidates and the input samples reduces variance further (dashed curves), though slightly less with subset-based resampling. 41

5.1 Importance resampled single scattering in a medium with $N = 1$ (top row) and $N = 8$ (bottom row) samples. Our method yields lower pixel and perceptual error than reservoir-based resampling, thanks to its effective sample stratification and blue-noise dithering. prelMSE values are scaled by $100\times$. 45

5.2 Comparison of our bidirectional CDF resampling against reservoir baselines on area-light illumination with $N = 1$ sample from $M = 32$ Halton-sequence candidates. Dithered reservoir resampling fails to produce a pleasing error distribution, whether using the candidate generation order (a) or along a Hilbert curve (b). In contrast, our dithered bidirectional CDF resampling with Hilbert-curve ordering (c) produces a high-quality blue-noise distribution. 48

5.3 Our candidate generation along a Hilbert curve, combined with our bidirectional CDF resampling, yields blue-noise error distribution (a). Using unsorted Halton-sequence candidates (b), or reservoir sampling (c), fails to attain such distribution. 49

5.4 Resampling an MIS mixture of direct-illumination candidates. 16 unit-square candidates produce $M = 32$ hemispherical candidates via environment-map and BSDF warping. For both $N = 1$ and $N = 4$ samples, our technique produces a blue-noise error distribution and reduces the overall error over reservoir-based resampling. prelMSE values are scaled by $100\times$. 50

5.5 Single scattering from multiple area lights in a medium. We compare our bidirectional CDF resampling of Hilbert-curve candidates to reservoir resampling of 3D Halton-sequence candidates ($M = 32$, $N = 8$). We include false-color error zoom-ins. 51

5.6 Error plots as functions of the sample count N for different scenes, with $M = 8N$ candidates. Our technique always outperforms reservoir sampling variants, achieving a better convergence rate. All of the methods shown are single pass. 52

Chapter 1

Introduction

Physically based rendering refers to the process of synthesizing realistic images by following the principles of physics that govern how light particles interact with the environment. Simulating such realism often comes at a great computational cost. The theory requires us to solve complex, high-dimensional integrals of functions with various sources of discontinuities.

Monte Carlo integration is a natural choice to handle difficult integration problems and therefore lends itself well to rendering. In Monte Carlo method, we generate random points in the integration domain, evaluate the integrand at those locations and average the values to obtain an estimate for our integral; note that the integrand can be discontinuous and the random points can be high dimensional. Particularly, in rendering, light transport can be simulated by generating random rays from a virtual camera, recursively tracing them, and computing light contribution at each scattering event that occurs when the ray intersects the scene geometry (objects or medium particles). At each scattering event, the ray continues its path in a newly sampled random direction until it reaches a light source. This method of solving the light transport problem is called *path tracing* and it requires the evaluation of a high dimensional (light is carried over the whole path) and non-smooth (discontinuities in the scene such as light being occluded by objects) integrand. Therefore, Monte Carlo integration is a simple yet powerful tool to utilize in physically based rendering algorithms.

However, Monte Carlo method is infamous for its slow convergence rate compared to other numerical integration methods. In rendering, we see the effects of slow convergence as noise in the output images. Figure 1.1 shows an example of how the noise appears and how it can be reduced by increasing the number of samples used in Monte Carlo estimation.

However, evaluating more samples increases the amount of computation substantially. Depending on the scene complexity, it may take hours or even days to render a noise-free image using a state-of-the-art path tracing method. We do not always have such high computation budget and thus, there exist a multitude of work focusing on reducing the amount of noise at relatively the same degree of computation. Some of this work concentrates on finding efficient evaluation of light transport to handle particularly challenging lighting conditions such as the scene being lit by mostly indirect lighting or light being focused by a glass object to form caustics. Other work investigates how sampling distributions and placement of samples in the integration domain can effect both the amount and the distribution of noise.

Figure 1.1: Stochastic nature of Monte Carlo estimation results in noise in output images when the number of rays / samples per pixel is insufficient. Rendering this kitchen scene by casting 16384 primary rays per pixel (rpp) produces a far more converged output than using 64 rpp.

In this thesis, we are mostly interested in superior uniformity of sample locations and (loosely) refer to it as *stratification*. Section 2.2.3 presents a more rigorous definition of stratification. Stratified samples lead to faster convergence of the output rendering. They are especially useful for low-dimensional problems as it is easier to generate a moderate

amount of samples that are well distributed over the domain. The thesis builds on the idea of resampling and shows how to enhance it by efficiently introducing stratification. Resampling is a powerful method for approximately sampling from distributions that are difficult to handle directly; it may be too expensive to generate samples exactly following certain distributions. The basic approach in resampling is to stochastically select from a list of candidate samples (coming from another, known distribution that is cheap to sample), where the selection probability of each candidate is proportional to an associated discrete weight. Judicious setting of these weights makes the resulting samples drawn approximately proportionally to a chosen target distribution. Talbot et al. [34] showed how resampling can generate scattered ray directions with distribution that follows the light contribution (except for visibility) to obtain possibly high contribution light rays efficiently. More recently, Bitterli et al. [4] applied such resampling to spatio-temporal reuse of path samples.

The key operation in resampling is to select one or more candidates with probability proportional to their weights, which is performed by a discrete sampling algorithm. Then, the selected samples can be used in Monte Carlo integration with a small modification to the estimator. Two common discrete sampling algorithms for resampling are inverse cumulative distribution function (CDF) sampling and reservoir sampling [5, 8]. The former precomputes a CDF from the candidate weights and inverts it to select a sample. The latter is a precomputation-free alternative that can select a sample according to the desired distribution in a single pass over all candidates.

To date, the key difference between the aforementioned two algorithms is understood as the need or absence of a precomputed data structure. We show that another significant difference lies in their *stratification* abilities. Put briefly, inverse CDF sampling can potentially transfer the stratification of the input canonical samples (e.g., coming from a low-discrepancy sequence) to the output samples, whereas reservoir sampling cannot. We also show that neither approach retains stratification when the sampling domain has more than one dimension. This latter limitation is due to the fact that resampling is always performed on the 1D space of candidate indices, thus the ordering of candidates impacts the output samples. As such, no efficient approaches exist to produce stratified samples with resampling.

We propose a solution to this problem. Our *bidirectional CDF sampling* algorithm combines the strengths of inverse CDF sampling and reservoir sampling, without their limitations. Specifically, it yields the same output as inverse CDF sampling (i.e., can retain input stratification), but in a single pass over the candidates, like reservoir sampling. This is achieved via an adaptive traversal of the candidate list from both ends. Avoiding precomputation is desirable when substantial memory consumption is restrictive to the

performance. For instance, a rendering application running on a *graphics processing unit* (GPU) takes advantage of its highly parallel architecture to parallelize the computation over all pixels. With each pixel requiring storage of (possibly) multiple CDFs, the memory consumption quickly becomes an issue.

Bidirectional CDF sampling is a general discrete sampling method that can directly replace inverse CDF sampling or reservoir sampling in certain applications, even outside the context of rendering. Within a single-pass, constant memory problem setting, bidirectional CDF sampling (largely) preserves stratification of the input to the output. We augment this algorithm with *candidate ordering* along a space-filling curve to form a bijective map that preserves the locality of high-dimensional samples in their 1D index space. A similar idea has been used successfully in the context of importance sampling [33], and we show how it can be used for resampling to effectively connect the stratification in the index space to the high-dimensional candidate space.

Lastly, unless the Monte Carlo integration is run until convergence or the scenes we render are too simplistic, the images will contain some amount of noise. Interestingly, it is not just the numerical error of the image that determines how we perceive the visual quality of the image but also the distribution of the error over the image pixels. The ordinary *white noise* is the result of decorrelating random samples among pixels. Decorrelation and thus white noise is generally preferred over correlated samples as correlation may lead to regular patterns that can be more distracting than white noise. However, Georgiev and Fajardo [9] show that not all correlation is harmful. In particular, *negative correlation*, which indicates that neighboring pixels should receive very different samples, can improve the perceptual quality of the output image. The resulting error distribution is *blue noise* which stands for noise with minimal energy in low frequencies. This thesis explains how to introduce blue-noise to resampling for improved visual fidelity.

1.1 Contributions

Combining bidirectional CDF sampling with candidate ordering enables the first single-pass resampling method that achieves output stratification. An additional benefit of our method is that it can be easily combined with techniques that impose a blue-noise error distribution in image space [9], further improving the visual result. We demonstrate the benefits of our method across various light-transport simulation settings. Our novel contributions include:

- A resampling algorithm that yields identical results to classical inverse CDF sampling but in a single pass over the candidates;

- Ordering of candidates along a space-filling curve; and
- A practical stratified resampling-based rendering algorithm that exhibits blue-noise distribution of image error.

1.2 Organization

Chapter 2 introduces relevant background information on Monte Carlo integration. We discuss the fundamentals behind Monte Carlo estimation, variance reduction and quasi-Monte Carlo method which replaces the independent pseudo-random samples in the standard Monte Carlo method by fully deterministic, low-discrepancy sequences.

Chapter 3 discusses relevant methods and ideas in the subject of sampling. Inverse CDF sampling and weighted reservoir sampling are two methods commonly used for resampling, both of which appear frequently in the discussion of our method. Furthermore, this section talks about blue-noise dithered sampling for improved perceptual quality as well as an application of the Hilbert space-filling curve to stratified sampling.

Chapter 4 presents the core ideas in this work. We first show the reasons for why current methods fail to achieve stratification in resampling. Then, we introduce our bidirectional CDF sampling algorithm that preserves the stratification of the input canonical samples to the output, in a single-pass over the candidates. We also explain our candidate reordering scheme on the Hilbert Curve for stratified resampling. Lastly, we combine these ideas to form our single-pass resampling algorithm and show how to introduce negative correlation to obtain blue-noise error distribution.

Chapter 5 demonstrates application of our method on various rendering problems such as single scattering in media, direct illumination, combination with multiple importance sampling and higher-dimensional integration. We also provide error plots to illustrate the benefit of stratification compared to reservoir sampling alternatives.

Parts of the material presented in this thesis are based on:

Ege Ciklabakkal, Adrien Gruson, Iliyan Georgiev, Derek Nowrouzezahrai and Toshiya Hachisuka. “Single-pass stratified importance resampling”, Computer Graphics Forum, 2022.

Chapter 2

Monte Carlo Integration

Monte Carlo integration is a general method for stochastically estimating complex integrals. Estimating integrals involving high dimensional, discontinuous functions is of utmost importance to the study of physically-based rendering as the light transport problem can be formulated as an integration problem by the rendering equation [16]. Thanks to its generality, Monte Carlo integration is the method of choice for many light transport algorithms. Robustness to scene geometry, lighting conditions, and high-dimensional integrands makes Monte Carlo integration an attractive method to estimate the radiance of light paths inside the scene. Using Monte Carlo integration in solving the rendering equation is computationally expensive due to its low convergence rate. The error of the Monte Carlo method manifests itself as random noise in output images. A large body of work in Monte Carlo rendering is concerned with reducing this noise with minimal added computational complexity such as *variance reduction* techniques [36, 27, 20] and *Quasi-Monte Carlo integration* [17, 19].

This chapter reviews Monte Carlo integration, followed by a brief introduction to variance reduction techniques that are relevant to the material in the following chapters. Lastly, certain concepts regarding the Quasi-Monte Carlo method, which replaces the pseudo-random samples in Monte Carlo integration by fully deterministic, *low discrepancy* [24] sequences to obtain a faster convergence rate are discussed.

2.1 Basic Monte Carlo Integration

Monte Carlo integration is a numerical integration method for estimating integrals. It is a probabilistic method that evaluates the integrand at random points to estimate arbitrary

integrals. Before discussing the Monte Carlo estimator, we review some basic probability definitions.

2.1.1 Random Variable

In statistics, a random variable is a mapping of outcomes of a random process to numerical values. A random variable admits a *probability distribution* that relates probabilities to possible outcomes. There are two types of random variables, discrete and continuous.

A discrete random variable X takes on a finite (or countably infinite) number of values. Thus, we can enumerate all possible outcomes x_i of X and assign a particular probability p_i to each of them. This gives us the probability distribution $p(x)$ of X which is called the *probability mass function* (PMF). Often, it is required to find the probability that X is less than some value x, which is given by the *cumulative distribution function* (CDF):

$$F(x) = P(X \leq x) = \sum_{x_i < x} p_i. \tag{2.1}$$

A continuous random variable X defined over a range of a continuous domain takes on infinitely many number of possible values. We consider X defined on the real line R for brevity. The probability that X takes on a value in the interval $[a, b]$, $a < b \in \mathbb{R}$ is given by

$$P(a \leq X \leq b) = \int_a^b p(x)\, dx \tag{2.2}$$

where $p(x)$ is called the *probability density function* (PDF). The corresponding CDF is then given by

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} p(t)\, dt. \tag{2.3}$$

Note that

$$p(x) = \frac{dF(x)}{dx}. \tag{2.4}$$

In general, for a PDF $p(x)$ defined over the domain Ω to be valid,

$$p(x) \geq 0,\ \forall x \in \Omega \tag{2.5}$$

$$\int_\Omega p(x)\, dx = 1 \tag{2.6}$$

must hold. Similarly, for the discrete case, we would have a sum instead of the integral for the second condition. If the integral (or the sum) over the domain is equal to 1, then we say that the PDF (or PMF) is *normalized*.

In the following chapters, we frequently deal with probability distributions which do not integrate to 1 but instead some $C \in \mathbb{R}$, i.e. the PDF is *unnormalized*. Then, we can normalize the distribution by dividing it by C. Following the notation by Talbot [35], we denote the unnormalized PDF by $\hat{p}$, where $\hat{p} = Cp$.

2.1.2 Expected Value

The *expected value* of a random variable X is the weighted average of the values X may assume where the weights are defined by its PDF (or PMF). If $X \in \Omega$ is a continuous random variable with PDF p(x), the expected value of X is given by

$$E[X] = \int_{\Omega} x\, p(x)\, dx. \tag{2.7}$$

Expectation may also be conditional on a set of random variables with known values. For instance, given a function $f(X, Y)$ of random variables $(X, Y) \in \Omega_X \times \Omega_Y$, and the value of $Y = y$, the conditional expectation is:

$$E[f \mid Y = y] = \int_{\Omega_X} f(x, y)\, p(x \mid y)\, dx \tag{2.8}$$

where $p(x \mid y)$ is the *conditional density function* representing the probability distribution of X ($x \in X$), given $Y = y$. Some important properties of the expected value are as follows:

(1) For any constant $c \in \mathbb{R}$, $E[c] = c$.

(2) Expected value is a linear operator: $E[\sum_i c_i X_i] = \sum_i c_i E[X_i]$, for constants c_i and random variables X_i.

(3) *Law of total expectation* states that the expectation of a random variable X is equal to the iterated expectation of X conditioned on a random variable Y on the same sample space as X, i.e., $E[X] = E[E[X \mid Y]]$.

2.1.3 Variance

The *variance* of a random variable X is the expected value of the squared deviation between a random variable and its expectation. Variance is defined by

$$V[X] = E\left[(X - E[X])^2\right] \tag{2.9}$$

Variance can also be conditional:

$$V[X \mid Y] = E\left[(X - E[X \mid Y])^2 \mid Y\right]. \tag{2.10}$$

Some important properties of the variance are as follows:

(1) For any constant $c \in \mathbb{R}$, $V[c] = 0$.

(2) $V[cX] = c^2 V[X]$.

(3) An alternative expression for variance is $V[X] = E[X^2] - E[X]^2$.

(4) If X_i are independent random variables, $V\left[\sum_i X_i\right] = \sum_i V[X_i]$.

(5) Square root of the variance $\sigma = \sqrt{V[X]}$ is called *standard deviation*.

(6) *Law of total variance* states that $V[X] = E[V[X \mid Y]] + V[E[X \mid Y]]$ where X and Y are random variables on the same probability space.

Law of total variance is derived by using law of total expectation as follows:

$$\begin{aligned}
V[X] &= E\left[X^2\right] - E[X]^2 \\
&= E\left[E\left[X^2 \mid Y\right]\right] - (E[E[X \mid Y]])^2 \\
&= E\left[E\left[X^2 \mid Y\right]\right] - (E[E[X \mid Y]])^2 + E\left[(E[X \mid Y])^2\right] - E\left[(E[X \mid Y])^2\right] \\
&= \left(E\left[E\left[X^2 \mid Y\right]\right] - E\left[(E[X \mid Y])^2\right]\right) + \left(E\left[(E[X \mid Y])^2\right] - (E[E[X \mid Y]])^2\right) \\
&= \left(E\left[E\left[X^2 \mid Y\right] - E[X \mid Y]^2\right]\right) + (V[E[X \mid Y]]) \\
&= E[V[X \mid Y]] + V[E[X \mid Y]]
\end{aligned} \tag{2.11}$$

2.1.4 The Monte Carlo Estimator

The objective of Monte Carlo integration is to estimate the integral

$$I = \int_{\Omega} f(x)\, dx. \tag{2.12}$$

The idea is to have a random variable $\hat{I}$ such that its expected value is equal to the integral to be estimated. Letting $\hat{I} = {}^{f(X)}/_{p(X)}$, where X has distribution $p(X)$, we have

$$E\left[\hat{I}\right] = E\left[\frac{f(X)}{p(X)}\right] = \int_{\Omega} \frac{f(x)}{p(x)}\, p(x)\, dx = \int_{\Omega} f(x)\, dx = I \tag{2.13}$$

with the condition that $p(x)$ is nonzero everywhere $f(x)$ is nonzero. $\hat{I}$ is called the *estimator* of the integral I. In general, letting $\hat{I} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$, we have

$$E\left[\hat{I}\right] = E\left[\frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N}\sum_{i=1}^{N} E\left[\frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N}\sum_{i=1}^{N} I = I. \tag{2.14}$$

Therefore, the standard Monte Carlo estimator is defined by

$$\hat{I}_{MC} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \;\approx\; \int_{\Omega} f(x)\, dx \tag{2.15}$$

where X_i are from the PDF p.

To assess the accuracy of Monte Carlo method, we examine the variance of the estimator $\hat{I}_{MC}$:

$$\begin{aligned} V\left[\hat{I}_{MC}\right] &= V\left[\frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}\right] \\ &= \frac{1}{N^2}\sum_{i=1}^{N} V\left[\frac{f(X_i)}{p(X_i)}\right] \\ &= \frac{1}{N} V\left[\frac{f(X)}{p(X)}\right] \end{aligned} \tag{2.16}$$

where X_i are independent and X represents an evaluation of some X_i.

Eq. (2.16) suggests that the variance decreases as $1/N$. The standard error of the estimator is defined as the standard deviation of the estimator:

$$\sigma_{\hat{I}_{MC}} = \sqrt{V\left[\hat{I}_{MC}\right]} = \sqrt{\frac{1}{N}V\left[\frac{f(X)}{p(X)}\right]}. \tag{2.17}$$

Hence the error converges at a rate of $O(1/\sqrt{N})$. The number of samples N need to be quadrupled to halve the error. In contrast, numerical quadrature methods such as the midpoint rule, the trapezoidal rule, and Simpson's rule have error bounds of $O(N^{-2})$, $O(N^{-2})$, and $O(N^{-4})$ respectively. However, the number of function evaluations for these quadrature methods scale exponentially with the number of dimensions d, resulting in rates of $O(N^{-2/d})$, $O(N^{-2/d})$, and $O(N^{-4/d})$. Moreover, if the integrand is discontinuous, the error bounds can become much worse. For instance, Simpson's rule requires a continuous fourth order derivative for the integrand to obtain the $O(N^{-4})$ bound. On the other hand, the error for Monte Carlo method is independent of the number of dimensions and the smoothness of the integrand.

2.1.5 Monte Carlo Rendering

In the context of physically-based rendering, color of each pixel can be computed by solving the rendering equation [16] to compute the radiance (lighting) for all points (in the scene) visible through the pixels. The rendering equation

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_s(\mathbf{x}, \omega_i \to \omega_o)\, L(t(\mathbf{x}, \omega_i), -\omega_i)\, cos\,\theta_i\, d\omega_i \tag{2.18}$$

suggests that the (outgoing) radiance at a point $\mathbf{x}$ in the direction ω_o is equal to the sum of emitted (L_e) and reflected radiance (the integral over all incoming directions). The reflected radiance is computed by evaluating an integral that is a product of multiple terms: $f_s(\mathbf{x}, \omega_i \to \omega_o)$ is the *bidirectional scattering distribution function* (BSDF) and describes how much light from incoming direction ω_i is scattered in the outgoing direction ω_i, $L(t(\mathbf{x}, \omega_i), -\omega_i)$ is the radiance term, and $cos\,\theta_i$ where θ_i is the angle between the incoming direction and the surface normal n at point $\mathbf{x}$. The integration is over all directions on the hemisphere Ω centered at $\mathbf{x}$, facing the surface normal. The radiance term contains the ray casting function $t(\mathbf{x}, \omega_i)$, which traces a ray with origin at $\mathbf{x}$ towards direction ω_i. Since radiance is constant along a ray (the scene is assumed to be inside a vacuum and there exist no particles to interact with the ray), we conclude that $L(t(\mathbf{x}, \omega_i), -\omega_i) = L_i(\mathbf{x}, \omega_i)$,

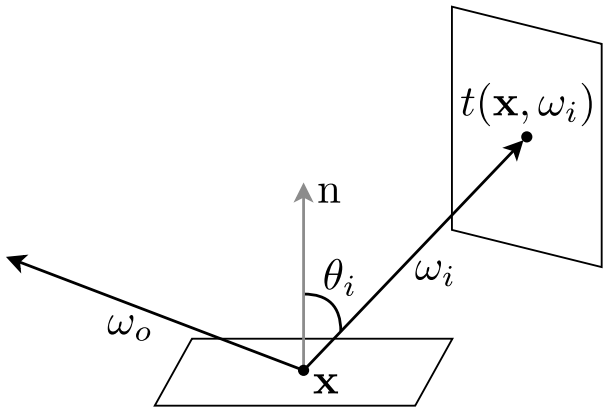


Figure 2.1: Local geometry for scattering at $\mathbf{x}$.

i.e., the radiance term is equal to the incoming radiance in the direction ω_i. Figure 2.1 provides a diagram of the local geometry at $\mathbf{x}$.

Note the recursive nature of the integral equation; the radiance term L appears on both sides. To evaluate the outgoing radiance at $\mathbf{x}$, we need the radiance at $t(\mathbf{x}, \omega_i)$ in the direction $-\omega_i$. We need to keep tracing rays and evaluate the radiance at those hitpoints. Hence, the integral is high dimensional. Additionally, discontinuities exist in the integral such as objects blocking the light in certain directions.

As established in Section 2.1.4, Monte Carlo method is apt to solve such integrals. One application of the Monte Carlo integration to solve the rendering equation is path tracing, which generates *primary rays* from the camera, traces them into the scene, and at each intersection with objects, samples new rays with random directions to be further cast into the scene. In the end, we are left with paths that connect light sources to the camera. This process is repeated many times and the results are averaged at each pixel.

2.2 Variance Reduction

Designing efficient Monte Carlo estimators is linked closely to the variance of the estimator $V\left[\hat{I}_{MC}\right]$.

As indicated by Eq. (2.17), a simple way to reduce the overall error of a Monte Carlo estimation is to increase the number of samples N. However, the convergence is slow. On the other hand, we can also reduce the variance $V\left[\frac{f(X)}{p(X)}\right]$, while keeping the number

of samples the same and thus reducing the overall error without a significant increase in computational effort.

A substantial amount of work in rendering research have focused on reducing the overall variance of Monte Carlo estimators, known as *variance reduction* techniques. We review some of them that are relevant for the work presented in this thesis. For a more extensive discussion, refer to the thesis by Veach [36].

2.2.1 Importance Sampling

In Monte Carlo integration, we have the freedom to choose an arbitrary PDF p for our estimator. Moreover, the ratio $\frac{f(X)}{p(X)}$ has direct influence on the variance of the estimator. The main idea of *importance sampling* is to choose the PDF such that the variance $V\left[\frac{f(X)}{p(X)}\right]$ is minimized, which leads to reduced overall variance of the estimator. If p is chosen to be proportional to f, i.e., $p(x) = cf(x)$ for some constant c, then we have

$$V\left[\hat{I}_{MC}\right] = \frac{1}{N}V\left[\frac{f(X)}{p(X)}\right] = \frac{1}{N}V\left[\frac{1}{c}\right] = 0. \tag{2.19}$$

Since $p(x)$ must be a normalized PDF, the constant c can be computed:

$$c = \frac{1}{\int_\Omega f(x)\,dx}. \tag{2.20}$$

Note that finding such a $p(x)$ and in turn c is impractical: The purpose of Monte Carlo method is to solve the integral $\int_\Omega f(x)\,dx$ in the first place. Yet, if $p(x)$ is similar to $f(x)$, the variance will be reduced. Intuitively, importance sampling seeks to place more samples in regions where the value of the integrand is high.

Care must be taken in choosing $p(x)$ as poor choices will result in increased variance. If $p(x)$ is small where $f(x)$ is large, and similarly $p(x)$ is large where $f(x)$ is small, the estimates will vary greatly. Any prior knowledge we have about the integrand can help us design PDFs that mimic f as much as possible.

It is common in rendering that we deal with complex integrands that are products of multiple functions, e.g., $f(x) = g(x)h(x)$. For instance, the rendering equation (Eq. (2.18)) contains the product of BSDF, radiance, and a cosine factor in the integral. It may be challenging to construct PDFs that closely follow the product and are also cheap to sample directly. However, it is plausible that we are able to devise a PDF that follows one of the terms in $f(x)$ such as $p \propto g$, or approximates some of the terms so that sampling is inexpensive.

2.2.2 Multiple Importance Sampling

Following the discussion of Section 2.2.1, a question that arises naturally is: Which PDF should be sampled when there are more than one suitable alternative? It turns out, we can construct a low-variance estimator by utilizing multiple sampling techniques. *Multiple importance sampling* (MIS) [37, 36] shows how to combine estimates from multiple sampling strategies with provable reduction of variance. The main idea of MIS is to use a set of weighting functions such that the combination of sample estimates results in lower variance.

Given a set of sampling distributions $\{p_1, \ldots, p_n\}$, all of which are potentially good strategies for importance sampling, and the number of samples n_i to draw from the distribution p_i we define the MIS estimator as follows:

$$\hat{I}_{MIS} = \sum_{i=1}^{n} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})} \tag{2.21}$$

where w_i are the corresponding weighting functions and $X_{i,j}$ is the j'th sample from the i'th distribution. The estimate is unbiased if the conditions

(1) $\sum_{i=1}^{n} w_i(x) = 1$ whenever $f(x) \neq 0$

(2) $w_i(x) = 0$ whenever $p_i(x) = 0$

hold. Therefore, there is some freedom in devising the weighting functions. However, not all sets of weighting functions will result in significant variance reduction. Veach and Guibas [37] propose certain weighting functions that are provably close to optimal, one of which is called the *balance heuristic* that defines the weights as

$$w_i(x) = \frac{n_i p_i(x)}{\sum_{k=1}^{n} n_k p_k(x)}. \tag{2.22}$$

It should be noted that in addition to generating samples from all $p_i(x)$ and evaluating them at $f(x)$, balance heuristic requires evaluating $p_i(x)$ at any point $x \in \Omega$ as well. In other words, samples generated from a particular density should be evaluated at all other densities.

The estimator discussed so far is called the *multi-sample estimator* which takes multiple samples from multiple strategies. The alternative is to randomly select one of the sampling strategies and generate a single sample from it to be evaluated with the *one-sample estimator*:

$$w_I(X_I) \frac{f(X_I)}{c_I p_I(X_I)} \tag{2.23}$$

where $I \in \{1, \ldots, n\}$ is a random variable sampled according to the probabilities $\{c_1, \ldots, c_n\}$, $\sum_{i=1}^{n} c_i = 1$, and X_I is sampled from the PDF p_I. One-sample estimator is also unbiased subject to the conditions stated above regarding the weighting functions. Furthermore, the balance heuristic is provably optimal for one-sample estimator.

Multiple importance sampling is a powerful variance reduction method as the MIS estimator is *robust*, meaning that it will achieve low variance under various configurations of integrands. For rendering problems, even if the integrand has varying behaviour over the scene and over the pixels in the image, we generally have an idea about its structure and we are able to devise multiple sampling techniques, each of which mimic the integrand to some extent. The weighting heuristics are the key to properly combine multiple sampling techniques so that the advantages of each technique are preserved.

2.2.3 Stratified Sampling

Importance sampling and multiple importance sampling concern themselves with the probability distributions that the samples are generated from. However, it is also noteworthy to consider how a finite amount of samples are placed over the domain of integration. In particular, taking a finite number of samples from a random sampler may result in clumps, i.e., multiple samples that are very close to each other. These clumps are undesirable because the nearby samples do not add much new information about the behaviour of the integrand. Ideally, we would like to have the samples more evenly spaced.

Stratified sampling subdivides the integration domain Ω into n non-overlapping regions $\Omega_1, \ldots, \Omega_n$ called *strata* and generates n_i many samples from each strata according to PDF p_i. Thanks to this stratification, the samples are distributed more uniformly.

The stratified Monte Carlo estimator is:

$$\hat{I}_{st} = \sum_{i=1}^{n} v_i \frac{1}{n_i} \sum_{j=1}^{n_i} \frac{f(X_{i,j})}{p_i(X_{i,j})} \tag{2.24}$$

where v_i is the volume of the *stratum* Ω_i, and $X_{i,j}$ denotes the j'th random sample from stratum i.

Figure 2.2 shows 2D samples on a unit square generated by uniform random sampling and stratified sampling. We can observe the clumping behavior of samples in the random case. Stratified samples on the other hand are distributed more evenly.

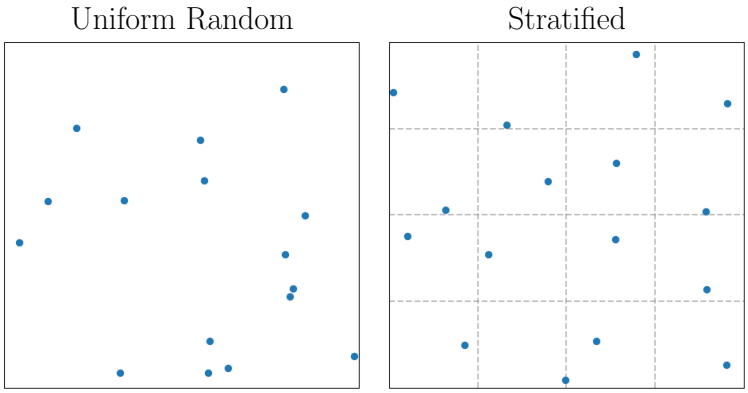


Figure 2.2: 16 random samples generated by uniform random sampling and stratified sampling with 16 uniform strata. Note the gaps and clumps present in purely random case.

The variance of the stratified Monte Carlo estimator is:

$$\begin{aligned} V\left[\hat{I}_{st}\right] &= V\left[\sum_{i=1}^{n} v_i \frac{1}{n_i} \sum_{j=1}^{n_i} \frac{f(X_{i,j})}{p_i(X_{i,j})}\right] \\ &= \sum_{i=1}^{n} \frac{v_i^2}{n_i^2} V\left[\sum_{j=1}^{n_i} \frac{f(X_{i,j})}{p_i(X_{i,j})}\right] \\ &= \sum_{i=1}^{n} \frac{v_i^2}{n_i^2} \cdot n_i V\left[\frac{f(X_i)}{p_i(X_i)}\right] \\ &= \sum_{i=1}^{n} \frac{v_i^2}{n_i} V\left[\frac{f(X_i)}{p_i(X_i)}\right] \end{aligned} \tag{2.25}$$

where X_i is a sample from p_i. To compare the variance of the stratified estimator to the unstratified one, first suppose that n_i's are chosen according to volumes v_i; that is $n_i = v_i N$, with N many samples in total. Then, the variance is:

$$\frac{1}{N} \sum_{i=1}^{n} v_i V\left[\frac{f(X_i)}{p_i(X_i)}\right]. \tag{2.26}$$

Consider that, first selecting a stratum Ω_i randomly with probability v_i and then generating the random sample $X_{i,j}$ inside Ω_i is equivalent to unstratified sampling in Ω. With this in

mind, we can say $X_{i,j}$ is generated conditionally on choosing Ω_i. Now, recall the variance for the standard Monte Carlo estimator:

$$V\left[\hat{I}_{MC}\right] = \frac{1}{N} V\left[\frac{f(X)}{p(X)}\right]. \tag{2.27}$$

Let A be a discrete random variable such that $A \in \{1, \ldots, n\}$ with probabilities $v_1, \ldots, v_n$, representing the random choice of a stratum. Let $E_Z[Y]$ denote the expectation over the domain of random variable Z. Thus, $E_A[Y] = \sum_{i=1}^{n} v_i Y$. Note that conditional expectation (and also variance) of $\frac{f(X)}{p(X)}$ given $A = a$ is:

$$E\left[\frac{f(X)}{p(X)} \mid A = a\right] = E\left[\frac{f(X_a)}{p_a(X_a)}\right]. \tag{2.28}$$

Then, we write the variance term $V\left[\frac{f(X)}{p(X)}\right]$ in terms of conditional expectations using law of total variance:

$$\begin{aligned}
V\left[\frac{f(X)}{p(X)}\right] &= E_A\left[V\left[\frac{f(X)}{p(X)} \mid A\right]\right] + V_A\left[E\left[\frac{f(X)}{p(X)} \mid A\right]\right] \\
&= \sum_{i=1}^{n} v_i\, V\left[\frac{f(X_i)}{p_i(X_i)}\right] + E_A\left[\left(E\left[\frac{f(X)}{p(X)} \mid A\right] - E_A\left[E\left[\frac{f(X)}{p(X)} \mid A\right]\right]\right)^2\right] \\
&= \sum_{i=1}^{n} v_i\, V\left[\frac{f(X_i)}{p_i(X_i)}\right] + E_A\left[\left(\underbrace{E\left[\frac{f(X)}{p(X)} \mid A\right]}_{\mu_i} - \underbrace{\sum_{i=1}^{n} v_i\, E\left[\frac{f(X_i)}{p_i(X_i)}\right]}_{I}\right)^2\right] \\
&= \sum_{i=1}^{n} v_i\, V\left[\frac{f(X_i)}{p_i(X_i)}\right] + E_A\left[(\mu_i - I)^2\right] \\
&= \sum_{i=1}^{n} v_i\, V\left[\frac{f(X_i)}{p_i(X_i)}\right] + \sum_{i=1}^{n} v_i\ (\mu_i - I)^2
\end{aligned} \tag{2.29}$$

where μ_i and I are the expected values of $\frac{f(X)}{p(X)}$ over the regions Ω_i and Ω respectively. Hence, the variance of unstratified estimator is:

$$V\left[\hat{I}_{MC}\right] = \frac{1}{N}\left(\sum_{i=1}^{n} v_i\, V\left[\frac{f(X_i)}{p_i(X_i)}\right] + \sum_{i=1}^{n} v_i\ (\mu_i - I)^2\right). \tag{2.30}$$

Comparing the above with the variance of the stratified estimator with $n_i = v_i N$ (Eq. (2.26)), we see that the only difference is the right hand side of the sum, which is a square and

always non-negative. The conclusion is that stratified Monte Carlo is never worse than plain Monte Carlo.

For stratified sampling, we typically consider the d-dimensional unit hypercube $[0,1]^d$ as the domain. Once the samples are generated in the hypercube, they can be transformed to other domains by $T : [0,1]^d \to \Omega$ where T is some appropriate transformation [36]. Note that this transformation can be designed to introduce arbitrary probability densities to the transformed samples.

The variance of unstratified Monte Carlo estimator converges at the rate of $O(N^{-1})$ under uniformly distributed random sampling (Eq. (2.16)). On the other hand, variance when using stratified samples converges at a rate between $O(N^{-1})$ and $O(N^{-2})$ depending on the smoothness of the integrand in the image space [23]. In higher dimensions (d), convergence rate of stratified Monte Carlo for a smooth integral is $O(N^{-1-2/s})$ [36].

Stratified sampling is favorable in low-dimensional settings because the number of stratas (and thus the number of samples) grows exponentially with the number of dimensions: For a d dimensional problem with n stratas in each dimension, there will be n^d stratas in total. Another disadvantage of stratified sampling is that the number of samples needs to be known in advance.

2.3 Quasi-Monte Carlo

As discussed in the previous section, superior uniformity of the samples in Monte Carlo integration can lead to lower variance. Much like the standard Monte Carlo method, *quasi-Monte Carlo* (QMC) method is used in numerical integration. The difference is that instead of using (pseudo) random samples, fully deterministic *quasi-random* samples are used. For simplicity, assume that the integration domain Ω is d-dimensional unit hypercube $[0,1]^d$ and the probability distribution $p(x)$ is constant. The QMC estimator is then written as

$$\frac{1}{N}\sum_{i=1}^{N} f(x_i) \approx \int_{[0,1]^d} f(x)\,dx \tag{2.31}$$

where $x_1, \ldots, x_N \in [0,1]^d$ are deterministic quasi-random points. They share some statistical properties with random numbers that allow them to be used in numerical integration. In addition, quasi-random sequences are distributed more uniformly and thus result in faster convergence of the integration problem.

This section presents a discussion of the term *discrepancy* as a measure for uniformity of sequences and point sets and look at a well-known *low-discrepancy* sequence in *Halton* sequence.

2.3.1 Discrepancy

Discrepancy is a measure of *equidistribution* for point sample sets. In other words, it measures how uniformly the points are distributed. A sequence with low-discrepancy will be distributed more evenly and be devoid of clumps. Discrepancy has been introduced to computer graphics by Shirley [31], who highlighted the connection between discrepancy and image error.

Intuitively, for a sequence to be as uniformly distributed as possible, every subregion of the domain $[0,1)^d$ should contain a number of sample points proportional to the subregion's volume. For instance, assume that we have N sample points in the hypercube $[0,1)^d$ and a subregion of volume V_{sub} contains n sample points, then n/N should roughly be equal to V_{sub} if the points are equidistributed.

Formally, let B denote the family of all subsets of $[0,1)^d$ Then, the discrepancy of the point set $P = x_1, \ldots, x_N$ is defined by:

$$D_N(B,P) = \sup_{b \in B} \left| \frac{\#\{P \cap b\ \}}{N} - v(b) \right| \tag{2.32}$$

where $v(b)$ denotes the Lebesgue measure (volume) of the subset b. A common choice for B is the family of all axis-aligned boxes in $[0,1)^d$ with a corner at the origin, denoted B^*:

$$B^* = \{[0,a_1] \times [0,a_2] \times \cdots \times [0,a_d] \ \mid 0 \le a_i < 1 \text{ for all } i\}. \tag{2.33}$$

In particular, Eq. (2.32) is called the *star discrepancy* and denoted by $D_N^*(P)$ when B^* is used. Another common alternative for B is to use arbitrary axis aligned boxes.

Error bounds for quasi-Monte Carlo estimation are closely related to discrepancy. The *Koksma-Hlawka inequality* states that

$$\left| \frac{1}{N} \sum_{i=1}^{N} f(x_i) - \int_{[0,1]^d} f(x)\, dx \right| \le V(f)\, D_N^*(P) \tag{2.34}$$

where f has bounded *variation* $V(f)$ *in the sense of Hardy and Krause* [25]. If f has bounded variation, than $V(f)$ is finite and the integration error is proportional to discrepancy. Therefore, by using low-discrepancy sequences, a smaller error of the integration can

be guaranteed. However, $V(f)$ is generally not finite when f is discontinuous, which is typical for rendering problems and makes the error bound less useful [36].

Sequences with discrepancy of $O((log\, N)^d/N)$ are often referred to as low-discrepancy sequences. Sequences by definition are of infinite length. Finite point sets on the other hand can have a slightly lower discrepancy: $O((log\, N)^{d-1}/N)$.

In conclusion, the error bound for quasi-Monte Carlo integration is of $O((log\, N)^d/N)$ for a low-discrepancy sequence and $O((log\, N)^{d-1}/N)$ for a low-discrepancy point set, which is a large improvement over the error bound of $O(\sqrt{N})$ of plain Monte Carlo.

2.3.2 Radical inversion and Halton Sequence

This section presents the construction of the low-discrepancy *Halton sequence*. The construction is based on the concept of *radical inversion*.

Radical Inverse. A non-negative integer a can be expanded in any base $b > 1$ as the sum of values d_i such that $0 \leq d_i < b$, $i = 1, \ldots, m$:

$$a = d_m b^{m-1} + d_{m-1} b^{m-2} + \cdots + d_2 b + d_1 = \sum_{i=1}^{m} d_i b^{i-1}. \tag{2.35}$$

Then, a is uniquely written in base b as the sequence of digits $d_m d_{m-1} \ldots d_1$. The radical inversion operator ϕ_b in base b reflects these digits around the decimal point:

$$\phi_b(a) = .d_1 d_2 \ldots d_m \tag{2.36}$$

which is a value in $[0, 1)$.

The *van der Corput sequence* is a 1D low-discrepancy sequence given by radical inversion in base 2 for values of $a = 0, 1, \ldots$:

$$x_a = \phi_2(a). \tag{2.37}$$

Van der Corput sequence has discrepancy of $O(logN/N)$.

Halton Sequence. Low-discrepancy sequences in higher dimensions ($[0, 1)^d$) can be obtained by applying radical inversion separately for each dimension, using relatively prime

bases. Let b_i denote relatively prime bases for $i \in \{1, \ldots, d\}$, then the Halton sequence [11] is defined by:

$$x_a = (\phi_{b_1}(a), \phi_{b_2}(a), \phi_{b_3}(a), \ldots, \phi_{b_d}(a)). \tag{2.38}$$

A reasonable choice for the bases is to use the first d prime numbers. The Halton sequence has a discrepancy of $O((log\, N)^d/N)$. If the number of samples N is known beforehand, regular samples a/N can be used for the first dimension, resulting in a slightly lower discrepancy. Such a point set is defined by

$$x_a = (a/N, \phi_{b_1}(a), \phi_{b_2}(a), \ldots, \phi_{b_{d-1}}(a)) \tag{2.39}$$

and are called *Hammersley point set* [12] and have discrepancy of $O((log\, N)^{d-1}/N)$. Figure 2.3 compares 2D points from the Halton sequence and the Hammersley point set.

There exist a number of other low-discrepancy point sets such as the Sobol' sequence [32], rank-1 lattice, (t, s)-sequences, and (t, m, s)-nets. A survey of such point sets and their use in quasi-Monte Carlo rendering algorithms is provided by Keller [17].

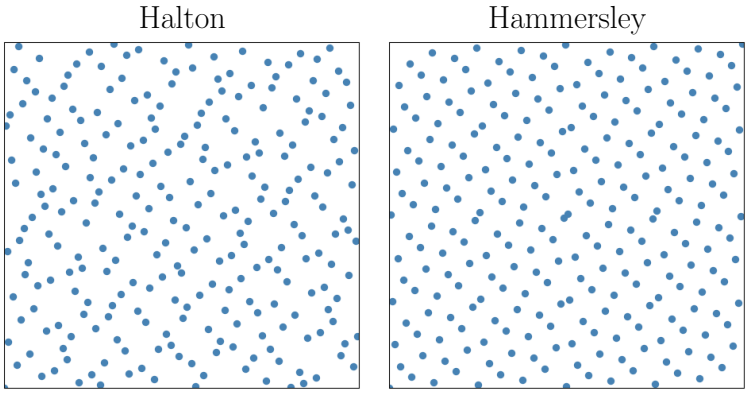


Figure 2.3: First 256 points of the Halton sequence and the Hammersley point set in 2D.

Chapter 3

Sampling

In the core of Monte Carlo integration lies the idea of *sampling* which is the process of generating discrete samples from continuous functions. In particular, sample generation according to a given probability density function is essential for importance sampling and for other variance reduction techniques. Sampling of general distributions frequently comes up in Monte Carlo rendering in the form of sampling BSDFs, light sources, scattering media. We may also be asked to sample a discrete probability distribution, in which case it is possible to treat the distribution as a piecewise constant continuous function.

This chapter reviews basic sample generation methods that can be used to sample both continuous and discrete distributions. Additionally, relevant methods concerning sample placement are discussed through related work.

3.1 Inverse CDF Sampling

Often times, it is necessary to generate non-uniform random variables from a specified probability distribution. Specialized sampling techniques may be available for general distributions such as normal, Poisson, binomial, and exponential [27] but in Monte Carlo rendering, sampling specific distributions defined by BSDFs, light sources or scattering media may be needed. Inverse CDF sampling is a general method to generate random variables from a particular probability distribution. It allows us to transform uniform random variables to non-uniform random variables by using the inverse of the cumulative distribution function. Depending on how the sampling is performed, we can generate continuous or discrete random variables.

Continuous Case. Suppose that X is a continuous random variable with CDF

$$F(x) = P(X \leq x). \tag{3.1}$$

Suppose that X has continuous PDF $p(x) > 0$ for all $x \in \mathbb{R}$. Then, the CDF $F(x)$ is also continuous and strictly increasing. Moreover, the inverse CDF F^{-1} exists and is strictly increasing as well. Now, let $U \sim \mathbf{U}(0,1)$ be a uniform random variable and let Y be a random variable such that $Y = F^{-1}(U)$. Then,

$$\begin{aligned} P(Y \leq x) &= P\left(F^{-1}(U) \leq x\right) \\ &= P\left(F\left(F^{-1}(U)\right) \leq F(x)\right) \\ &= P(U \leq F(x)) = F(x). \end{aligned} \tag{3.2}$$

Thus, Y has the same distribution as X. In other words, to generate samples from the distribution p, one can compute and invert its CDF, generate uniform random variables U and convert them into non-uniform random variables with the desired distribution by letting $Y = F^{-1}(U)$. Note that to perform this sampling method, a closed form expression for CDF that can be inverted is needed.

Discrete Case. A discrete random variable X can be specified by a discrete probability table (e.g. PMF or tabulation of a continuous PDF) such that $p_i = P(X = x_i)$, $i = 1, \ldots, n$. Let $U \sim \mathbf{U}(0,1)$ be a uniform random variable and let Y be a random variable such that $Y = F^{-1}(U)$ where F^{-1} is defined as follows:

$$Y = F^{-1}(U) = \begin{cases} x_1 & \text{if } U < p_1 \\ x_2 & \text{if } p_1 \leq U < p_1 + p_2 \\ \vdots & \vdots \\ x_n & \text{if } \sum_{i=1}^{n-1} p_i \leq U < \sum_{i=1}^{n} p_i \end{cases} \tag{3.3}$$

Then, Y has the same distribution as X since $P(Y = x_i) = p_i$.

Instead of a discrete probability table, we may be given elements x_i with associated weights w_i. Such a setting could be interpreted as sampling from an unnormalized PMF where probability of sampling x_i is $w_i \,/\, \overline{w}_{sum}$ where $\overline{w}_{sum} = \sum_{i=1}^{n} w_i$ is the total sum of weights.

One way to transform uniform random variables is to search over the given probability table (or its CDF). Naively, a linear search will suffice, but has complexity $O(n)$. A more

standard approach is to use a bisection-based inversion for $O(\log n)$ complexity. There exists various other approaches such as the *alias method* [39] and the approximate CDF inversion methods [7] for constant-time sampling of discrete probability distributions.

The aforementioned sampling approaches require storage of the probability table or a similar structure derived from it during a precomputation stage before sampling. If memory usage is strictly limited (as can be for a highly paralellized application on a GPU), one can still utilize inverse CDF sampling by a two-pass linear search; once to go over all the elements and compute $\overline{w}_{sum}$, and a second time to find the output sample x_i (Alg. 1). For such an algorithm, the first pass can be considered as the precomputation step. As long as the weights are computed on the fly, the elements don't need to be stored.

Algorithm 1: Inverse CDF Sampling (Two-pass linear search)

```
1  function inverseCDFSample(𝔼)
2      w̄_sum ← 0 // The sum of weights
3      for i ← 1 to M do
4          x_i ← 𝔼[i] // Generate the element
5          w̄_sum ← w̄_sum + weight(x_i)
6      u ← rand()
7      w̄_run ← 0 // The running sum of weights
8      for i ← 1 to M do
9          x_i ← 𝔼[i] // Generate the element
10         w̄_run ← w̄_run + weight(x_i)
11         if u · w̄_sum < w̄_run then
12             return x_i
```

Alg. 1 can easily be extended to select multiple final samples by having an array of input random variables u.

3.2 Rejection Sampling

Rejection sampling is another method to sample general distributions. Suppose we want to sample a distribution $\hat{q}$ and we have another distribution $\hat{p}$ that is easily sampled from. In addition, there exists a constant C such that $\hat{q}(x) \leq C\,\hat{p}(x)$. Then, rejection sampling can be used to generate samples with distribution $\hat{q}$.

Firstly, we draw a candidate sample x according to pdf $\hat{p}$ alongside a uniform random number $u \in \mathbf{U}(0, 1)$. Then, if $u \leq \hat{q}(x) \,/\, C\,\hat{p}(x)$, x is accepted as a sample. Otherwise, x is rejected and the previous steps are repeated until a candidate is accepted. We provide pseudocode in Alg. 2.

Algorithm 2: Rejection Sampling

Input : $\hat{q}$, $\hat{p}$, C such that $\hat{q}(x) \leq C\,\hat{p}(x)$
Output: Sample $y \sim \hat{q}(x)$

```
while true do
    x ~ p̂ // Sample candidate from p̂
    u ← rand()
    if u ≤ q̂(x) / C p̂(x) then
        y ← x
        return y // Candidate is accepted
```

The geometric interpretation of rejection sampling provides some intuition. The method generates uniform samples under the curve $C\,\hat{p}(x)$ and accepts the ones which are under $\hat{q}(x)$.

Although rejection sampling is very general, it can be quite inefficient if $C\,\hat{p}(x)$ does not bind $\hat{q}(x)$ tightly as many candidates will get rejected before an acceptance. If there is no additional information relating the distributions $\hat{q}$ and $\hat{p}$ finding an appropriate constant C can be difficult.

3.3 Weighted Reservoir Sampling

Weighted reservoir sampling (WRS) [5] is a family of algorithms that select N samples from a stream of M elements in a single pass, with probability proportional to elements' weights. The elements are processed one by one and storage is required only for the N selected samples. Moreover, M, the length of the stream need not be known in advance.

A reservoir is a data structure that holds the N selected samples alongside some data about the stream such as sum of the weights seen so far [4]. Processing the input stream in order, each element is either inserted into the reservoir or discarded based on its weight. If the element is inserted into the reservoir, it can replace existing samples inside the reservoir. If the element is discarded, the sampling operation simply proceeds to the next element.

It is possible to select the same element multiple times when $N > 1$. This is called reservoir sampling *with* replacement. Conversely, reservoir sampling *without* replacement doesn't allow selection of such subsets. In the context of Monte Carlo integration, samples should be independent. When sampling *without* replacement, selection of an element depends on if it has already been inserted to the reservoir or not. Sampling *with* replacement on the other hand, provides independent samples.

The problem setting of weighted reservoir sampling can be described as:

- Sampling from a stream $\{x_1, x_2, \ldots, x_M\}$ to select N of the them based on their weights
- Run as a single pass over the elements
- Only store the subset of N selected elements in the memory

Given that each element x_i of the stream has associated weight of $w(x_i)$, an element x_i should be selected with probability

$$\frac{w(x_i)}{\sum_{j=1}^{M} w(x_j)} \tag{3.4}$$

A reservoir $\mathcal{R}$ of N samples is maintained by randomly accepting or rejecting the elements in the stream. At any point in the stream, each selected sample inside the reservoir is distributed proportionally to the weights processed so far. In other words, for each of the N spots inside the reservoir, the probability of having selected x_i after processing k many elements ($1 \leq i \leq k$) is

$$\frac{w(x_i)}{\sum_{j=1}^{k} w(x_j)} \tag{3.5}$$

Once again, considering each of the N spots in $\mathcal{R}$ individually, the reservoir update rule is to stochastically select the element x_{k+1} and replace the previously selected sample x_i inside $\mathcal{R}$ with probability

$$\frac{w(x_{k+1})}{\sum_{j=1}^{k+1} w(x_j)} \tag{3.6}$$

Alternatively, x_{k+1} is rejected and x_i is kept inside the reservoir with probability

$$\frac{w(x_i)}{\sum_{j=1}^{k} w(x_j)} \left(1 - \frac{w(x_{k+1})}{\sum_{j=1}^{k+1} w(x_j)}\right) = \frac{w(x_i)}{\sum_{j=1}^{k+1} w(x_j)} \tag{3.7}$$

In either case, the sample inside the reservoir is selected with the desired probability for the sample kept in $\mathcal{R}$. This result also implies that the stream can be of infinite length; at any point along the stream, the samples inside the reservoir follow the desired distribution.

The algorithm for weighted reservoir sampling with replacement and $N = 1$ is given in Alg. 3. It follows the style of Bitterli et al. [4]. Elements come in as a stream and their weights are computed on the fly.

Algorithm 3: Weighted Reservoir Sampling

```
1  class Reservoir
2      y ← 0 // The output sample
3      w̄_sum ← 0 // The sum of weights
4      function update(x_i, w_i)
5          w̄_sum ← w̄_sum + w_i
6          if rand() < (w_i / w̄_sum) then
7              y ← x_i
8  function reservoirSampling($)
9      Reservoir R
10     for i ← 1 to M do
11         R.update($[i], weight($[i]))
12     return R
```

Note that we need as many uniform random numbers as the length of the stream, which is computationally wasteful. It turns out that it is possible to consume only a single random number and continuously rescale it at each reservoir update by the renormalization from [22]. Starting with an initial random number $u_1 \in [0, 1)$, if k'th element is accepted, set $u_{k+1} = \frac{u_k}{p_k}$, else $u_{k+1} = \frac{u_k - p_k}{1 - p_k}$ (Alg. 4). Ogaki [26] provides further analysis on rescaling and shows another way of consuming a single random number for reservoir sampling.

It is straightforward to extend Algs. 3 and 4 for the $N > 1$ case. For each of N spots in $\mathcal{R}$, the stochastic accept/reject operation is repeated (Alg. 5).

Weighted reservoir sampling recently came into greater attention in rendering research thanks to *Spatiotemporal reservoir resampling* [4], efficiently addressing the problem of real-time direct lighting in scenes lit by thousands of light sources. Making use of reservoir sampling and reservoir reuse, they are able to efficiently reuse multiple samples in rendering a pixel while avoiding large storage requirements. Thus, weighted reservoir sampling looks to be a promising tool for real-time GPU ray tracing [21].

Algorithm 4: Weighted Reservoir Sampling with rescaling

```
class Reservoir
    y ← 0 // The output sample
    w̄_sum ← 0 // The sum of weights
    function update(x_i, w_i, u)
        w̄_sum ← w̄_sum + w_i
        p_i ← (w_i / w̄_sum)
        if u < p_i then
            y ← x_i
            u ← u/p_i
        else
            u ← (u − p_i)/(1 − p_i)
        return u
function reservoirSampling($)
    Reservoir R
    u ← rand()
    for i ← 1 to M do
        u ← R.update($[i], weight($[i]), u)
    return R
```

3.4 Resampled Importance Sampling

Generally, importance sampling requires generating samples from a specified probability distribution (e.g. using inverse CDF sampling). However, it may not always be possible to directly sample the distribution (no closed form analytic expression or too complex to integrate and invert). Sampling importance resampling (SIR) is a method to approximately sample such difficult distributions [29].

Sampling importance resampling first generates a set of *candidates* (or *proposals*) $\{x_1, \ldots, x_M\}$ from a source distribution p that is easily sampled. Then, computes weight for each candidate as $w_i = \hat{q}(x_i)/p(x_i)$ where $\hat{q}$ is the (possibly unnormalized) target distribution. Lastly, *resamples* the candidates by selecting (with replacement) $N \leq M$ samples $\{y_1, \ldots, y_N\}$ from the candidates with probability proportional to their weights. The selected samples are approximately distributed according to the target $\hat{q}$.

Resampled importance sampling [34] (RIS) presents an unbiased Monte Carlo estimator

Algorithm 5: Weighted Reservoir Sampling with rescaling ($N > 1$)

```
1  class Reservoir
2      y ← Array[N] // N output samples
3      w_sum ← 0 // The sum of weights
4      function update(x_i, w_i, u)
5          w_sum ← w_sum + w_i
6          p_i ← (w_i / w_sum)
7          for i ← 1 to N do
8              if u[i] < p_i then // for w/o rescaling, use rand() instead
9                  y[i] ← x_i
10                 u[i] ← u[i]/p_i
11             else
12                 u[i] ← (u[i] − p_i)/(1 − p_i)
13         return u
14 function reservoirSampling($)
15     Reservoir R
16     u ← rand(N) // N random variables
17     for i ← 1 to M do
18         u ← R.update($[i], weight($[i]), u)
19     return R
```

that utilizes SIR in importance sampling for variance reduction. Samples are generated using SIR so that they approximately follow a target distribution $\hat{q}$. Then, these samples are used in the standard Monte Carlo estimator as if they were directly drawn from $\hat{q}$. However, the estimator has to be weighted to correct for the fact that $\hat{q}$ is unnormalized and the selected samples are only approximately distributed according to $\hat{q}$. This weight turns out to be the average of candidate weights:

$$\frac{1}{M}\sum_{j=1}^{M} w_j. \tag{3.8}$$

The standard Monte Carlo estimator with normalized sampling density q and integrand f is

$$\hat{I}_{MC} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(y_i)}{q(y_i)}. \tag{3.9}$$

Combining SIR with importance sampling, the basic RIS estimator is defined as

$$\begin{aligned}\hat{I}_{ris} &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(y_i)}{\hat{q}(y_i)}\right)\cdot\frac{1}{M}\sum_{j=1}^{M}w_j \\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(y_i)}{\hat{q}(y_i)}\right)\cdot\frac{1}{M}\sum_{j=1}^{M}\frac{\hat{q}(x_j)}{p(x_j)}.\end{aligned} \tag{3.10}$$

RIS estimator is unbiased as long as $\hat{q}$ and p are positive everywhere that f is non-zero and $M, N \geq 1$.

Variance Analysis. The variance of the RIS estimator is [35]

$$V(\hat{I}_{ris}) = \frac{1}{M}V\left(\frac{f(X)}{p(X)}\right) + \left(1-\frac{1}{M}\right)\frac{1}{N}V\left(\frac{f(Y)}{q(Y)}\right) \tag{3.11}$$

where X and Y are random variables with densities p and $q = \frac{\hat{q}}{\int \hat{q}}$ respectively. As $M \to \infty$, the first term of the sum goes to zero and we conclude that resulting samples are distributed exactly according to q. RIS achieves variance reduction over the standard Monte Carlo estimator provided that

$$V\left(\frac{f(X)}{q(X)}\right) < V\left(\frac{f(X)}{p(X)}\right) \tag{3.12}$$

which suggests that the target distribution $\hat{q}$ follows the integrand f more closely than the source distribution p. In addition, candidate generation should be computationally efficient compared to evaluating samples. For instance, for a direct lighting problem, target distribution can be designated as the unshadowed path contribution and the source distribution as the uniform distribution. In this case, target distribution is closer to the actual integrand compared to the source distribution and casting of the shadow rays is avoided during candidate generation.

Note that the resampling stage of RIS can be performed by either inverse CDF sampling or weighted reservoir sampling as they can both sample elements with probability proportional to their weights.

3.5 Blue-Noise Dithered Sampling

The main objective of variance reduction and sampling techniques is to reduce the numerical error of a Monte Carlo rendered image. However, distribution of this error in screen

space can also affect the visual fidelity of the image. To this end, sequences of random values corresponding to individual pixels are decorrelated. This results in the familiar *white noise*. Georgiev and Fajardo [9] show that random pixel decorrelation is not perceptually optimal. Instead, their method *blue-noise dithered sampling* (BNDS) introduces *negative correlation* among pixels which results in superior perceptual quality. Negative correlation means that neighboring pixels end up with random sequences that are as different as possible. The idea of BNDS is to negatively correlate pixel estimates by having a unique random sequence and offsetting it for every pixel by a *blue-noise dither mask*. Figure 3.1 compares a 1D random sample mask with a 1D blue-noise dither mask and shows their Fourier power spectra. Note that the blue-noise mask has minimal energy in the low frequency region.

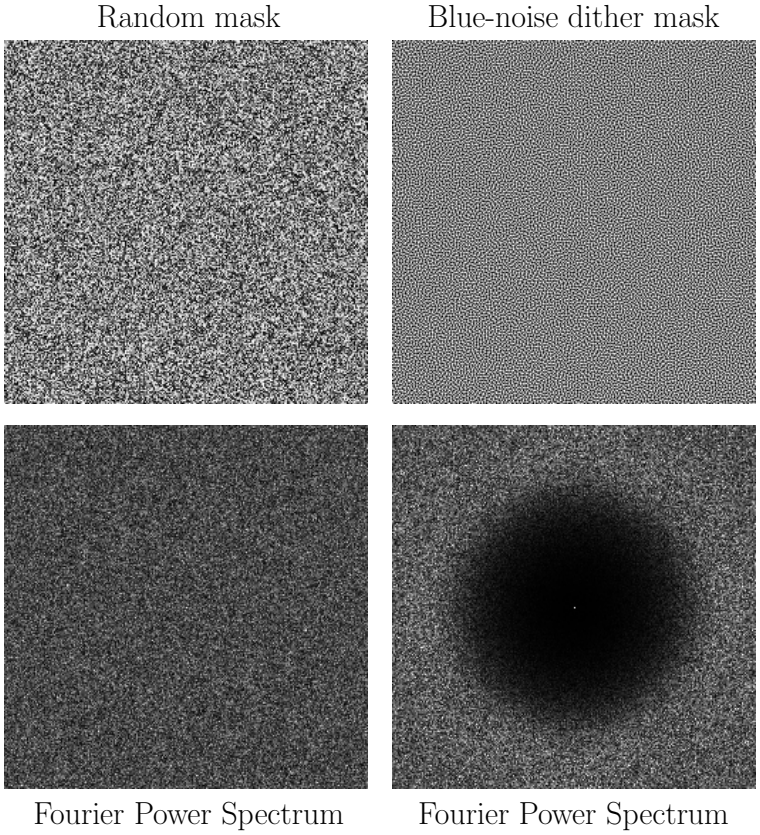


Figure 3.1: Comparison of a 1D random mask with decorrelated pixel values and a 1D blue-noise dither mask with negatively correlated pixels. The random mask displays white noise properties: It has a flat frequency spectrum meaning that the energy is distributed uniformly over all frequency bands. In contrast, blue-noise has low energy in low frequencies.

The blue-noise dither mask is constructed offline. Starting with a random mask, pixels of the mask are continuously swapped to minimize an energy function that is a product of image-space and sample-space Gaussians. The resolution of the mask is usually lower than that of the image to be rendered therefore it is tiled over the image. The dimension of the mask matches the dimension of random vectors in the unique sequence. A D-dimensional dither mask stores D-dimensional vectors in each pixel.

During rendering, each pixel offsets the random sequence $(x_1, x_2, \ldots, x_N)$ by looking up the vector in the blue-noise mask tiled over the image. The offsetting is a toroidal shift, also called Cranley Patterson rotation. Therefore, the sequence of pixel l is defined by

$$x_{i,l} = \text{mod}(x_i + d_l, 1) \tag{3.13}$$

where d_l is the random vector corresponding to the pixel l in the tiled blue-noise mask. Such offsetting is called *global offsetting* [18]. There is also *stratified offsetting* which, given a stratified sequence of samples, offsets them in their own strata.

BNDS does not reduce the magnitude of the numerical error of Monte Carlo estimation. Instead, it distributes the error over the image space for better perceptual quality. One way to measure the perceptual error is to convolve an error image by a Gaussian kernel [6]. Another is to observe variance after denoising the image [2]. Denoising an image with blue-noise error distribution will result in lower numerical error compared to denoising an image with white-noise. This is because of the fact that denoiser is a low-pass filter and a blue-noise image has energy in high-frequencies.

BNDS works well when the sample counts are low and integrands are smooth so that the samples and the pixel values are correlated [13]. In addition, high-dimensional dither masks with blue-noise properties are difficult to produce.

3.6 Stratified Sampling Using the Hilbert Curve

Hilbert space-filling curve is a continuous mapping from the unit interval $\mathcal{I} = [0, 1]$ to the n-dimensional hypercube $\mathcal{Q} = [0, 1]^n$. It is a curve that passes through all points in $\mathcal{Q}$ without intersecting itself. Hilbert [15] was the first to provide a general procedure to construct a class of space filling curves by a series of recursive partitioning and mapping of the unit interval domain and the unit square range. The Hilbert curve is usually defined as the limit of this recursive procedure. In practice, discrete approximation of the Hilbert curve is used, which is obtained after a finite number of recursive steps. The Hilbert curve

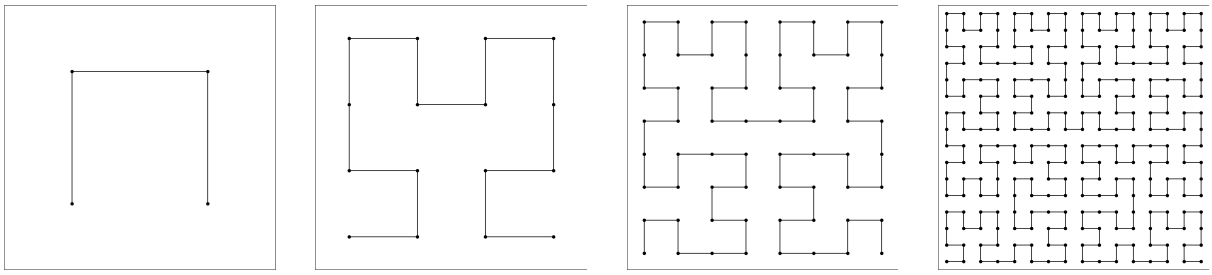


Figure 3.2: Generation of the Hilbert curve. The curve has a resolution of 2^m in each dimension.

goes over 2^{nm} points in total where m is called the *order* of the Hilbert curve. Figure 3.2 demonstrates the first few stages in generating a 2D Hilbert curve.

Compared to other space-filling curves, a significant advantage of Hilbert curves is their high coherence [38]. The curve sequentially visits adjacent areas. Thanks to high coherency, nearby points in the unit interval are likely to be nearby after the application of the Hilbert curve mapping.

Utilizing the coherency property of the Hilbert curve, Steigleder and McCool [33] introduced a general method to generate high dimensional stratified samples. The main idea is to first generate uniform stratified samples in the unit interval $\mathcal{I}$, then apply the Hilbert curve mapping to obtain uniform stratified samples in the unit hypercube $\mathcal{Q}$.

Chapter 4

Single-Pass Stratified Importance Resampling

This chapter develops the core technique presented in this thesis, a *single-pass stratified importance resampling* algorithm. The motivation for our method is that the previous discrete sampling methods used for resampling fail to achieve stratified sampling in a single pass, constant memory problem setting. Section 4.1 demonstrates the issues with the previous approaches and analyzes the reasons for their failures. Section 4.2 introduces our *bidirectional CDF sampling* method which well preserves the stratification of the input to the output in a single-pass over the candidates, using constant memory. Section 4.3 explains how to generate the candidates on the Hilbert space-filling curve. Lastly, Section 4.4 combines the previous sections to present our stratified resampling algorithm which produces well distributed output samples and blue-noise error distribution.

4.1 Lack of Stratification in Resampling

This section reviews the problem of stratification in resampling and offers two experiments to motivate the contributions.

4.1.1 Resampling in Monte Carlo Integration

Importance resampling [29] aims to sample proportionally to a target function $\hat{q}$ by selecting from M candidate samples $\{y_k\}_{k=1}^{M}$ drawn from some probability density function (PDF)

p. A weight $w_k = \hat{q}(y_k)/p(y_k)$ is associated with each candidate, and an index j is sampled with probability proportional to the weight: $P(j) = w_j / \sum_{k=1}^{M} w_k$. The PDF of the output sample $x = y_j$ approaches proportionality to the target $\hat{q}$ as $M \to \infty$. Talbot et al. [34] showed how to construct an unbiased Monte Carlo (MC) estimator for an integral:

$$\int_{\mathcal{H}} f(x)\,dx \;\approx\; \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{\hat{q}(x_i)} \left(\frac{1}{M}\sum_{k=1}^{M} w_k \right), \tag{4.1}$$

which averages over N samples drawn from M candidates. In this work we consider integration over a unit hypercube $\mathcal{H}$ of problem-dependent dimension, a.k.a. primary sample space. Sampling in this space is traditionally uniform; the integrand f may internally warp x to the native integration domain (e.g., unit sphere, or path space, with an appropriate Jacobian). In our case, the candidates y_k are uniform (with $p(y_k) = 1$) but the final samples x_i generally are not.

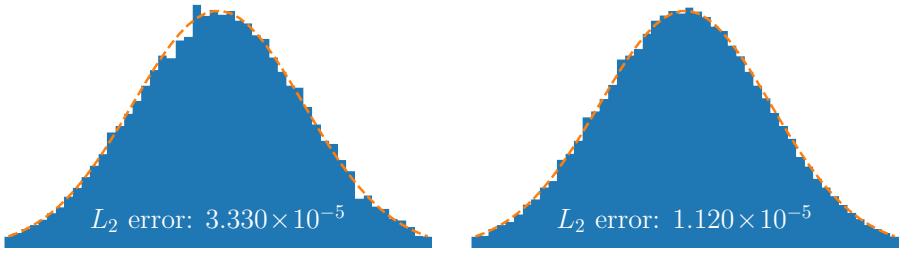
As usual in MC integration, variance is reduced when the samples x_i are stratified. Achieving this efficiently is the goal of our work. Next, we show that it requires a suitable resampling algorithm alongside a careful candidate ordering, and why existing methods struggle.

4.1.2 Stratification of Sampled Indices

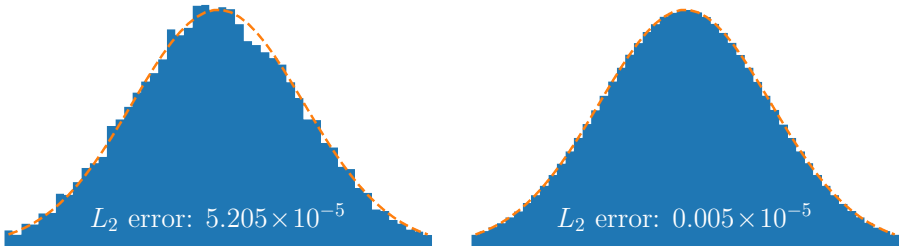
Consider the problem of sampling a candidate index j. The conventional approach is to select it as $j = F^{-1}(u)$, where $u \in [0, 1)$ is a canonical uniformly distributed input sample and F^{-1} is the inverse CDF of P. The mapping of the unit line to the index space via F^{-1} typically preserves the locality, thus a set of N well-distributed canonical input samples yield N well-distributed indices.

As an alternative to inverse CDF sampling, reservoir sampling avoids any CDF computation, and keeps track of only one selected candidate as it sweeps over the candidates once. For each observed candidate, a canonical sample is consumed to decide whether to replace the candidate in the reservoir. Reservoir sampling thus requires M canonical samples to sample a single index j. In an attempt to obtain N stratified output indices, we consider two options. One is to use a set of N M-dimensional stratified canonical samples and consume one dimension of each for every reservoir decision. A low-discrepancy sequence like Sobol's [32] can produce such an input. The second option is to use only N canonical random numbers but repeatedly shift and scale each for every reservoir decision, much like probability-tree traversal [22, 26].

Figure 4.1 compares histograms of inverse CDF sampling, with and without a stratified input, and reservoir sampling using the two aforementioned techniques. There are $M = 50$ sorted, regularly-spaced candidates with Gaussian weights. We repeat the resampling process N times before estimating the samples' distribution. Both reservoir variants fail to preserve input stratification, producing histograms resembling that of inverse CDF sampling with independent input samples. In contrast, feeding stratified samples to inverse CDF sampling produces a high-quality distribution, but at the cost of precomputing and storing a CDF.



(a) Reservoir + 1D QMC + rescaling (b) Reservoir + MD QMC



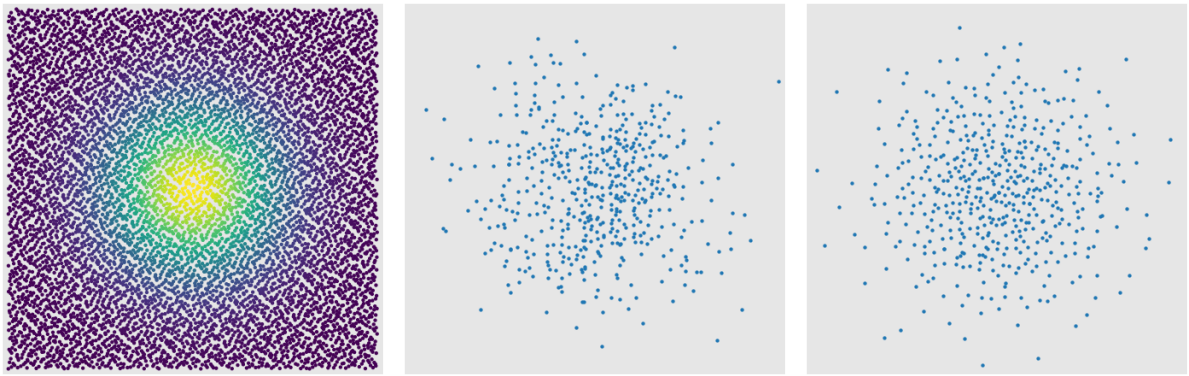
(c) CDF + independent samples (d) CDF + 1D QMC

Figure 4.1: Histogram comparison between reservoir sampling with rescaled 1D Sobol (i.e., van der Corput) samples (a) and M-dimensional Sobol samples (b), and inverse CDF sampling with 1D independent samples (c) and 1D Sobol samples (d). The candidates are regularly-spaced and sorted $(0.5, \ldots, 49.5)$. Sampling is done proportionally to Gaussian weights ($\mu = 25$, $\sigma = 10$). Neither reservoir sampling variant retains the input's stratification; inverse CDF sampling does, producing a substantially lower L_2 histogram error.

4.1.3 Impact of Candidate Order

One may intuitively expect a stratified resampling algorithm to yield stratified samples whenever the candidates themselves are stratified. However, this is not always true, as

we illustrate in Figure 4.2. We generate M stratified candidates on the unit square using a 2D Halton sequence (Fig. 4.2a), which we then resample to produce $N < M$ samples distributed approximately according to a 2D Gaussian function. Even though we employ stratified inverse CDF resampling (using a van der Corput sequence), the resulting samples in Fig. 4.2b are not well stratified. Contrast this result to that in Fig. 4.2c which exhibits improved stratification; the only difference is that we have applied our proposed candidate ordering scheme. The takeaway here is that resampling operates only on the candidate indices, oblivious to the fact that they may correspond to points in a high-dimensional integration space. Stratification in the 1D index space thus may not correspond to stratification in the integration space, unless care is taken when mapping samples to indices.



(a) 2D candidates (b) QMC order (c) Our method

Figure 4.2: The ordering of resampling candidates can substantially affect the stratification of the output samples. (a) We generate $M = 8,192$ Halton samples on the unit square and weight them by a 2D Gaussian target function. We then select $N = 256$ of them via stratified inverse CDF sampling. (b) Processing the candidates in their Halton-sequence generation order produces no stratification in the output. (c) Ordering the candidates using our proposed method (Section 4.3) yields stratified unit-square samples.

4.1.4 Problem Statement and Overview

The previous two experiments reveal two problems that need to be solved in order to achieve stratified resampling. The first is that stratification in the input canonical samples should be preserved by the index sampling algorithm. While inverse CDF sampling has the potential to achieve this, it may not be a viable option as it requires precomputation. As Bitterli et al. [4] pointed out, when resampling at every image pixel in parallel, it is

desirable to use a method that avoids precomputation, as reservoir sampling does. The second problem is that a stratified resampling algorithm must be paired with an appropriate candidate ordering that retains locality between the sampling domain and the index space, so that stratified index selection translates to stratified output samples.

To solve the first problem, in Section 4.2 we propose a new algorithm that returns the same output as inverse CDF sampling but in a single pass, without precomputation or candidate storage, like reservoir sampling. In Section 4.3 we show that the second problem can be solved by ordering the candidates along a space filling curve in the hypercube. Figure 4.3 illustrates our overall algorithm.

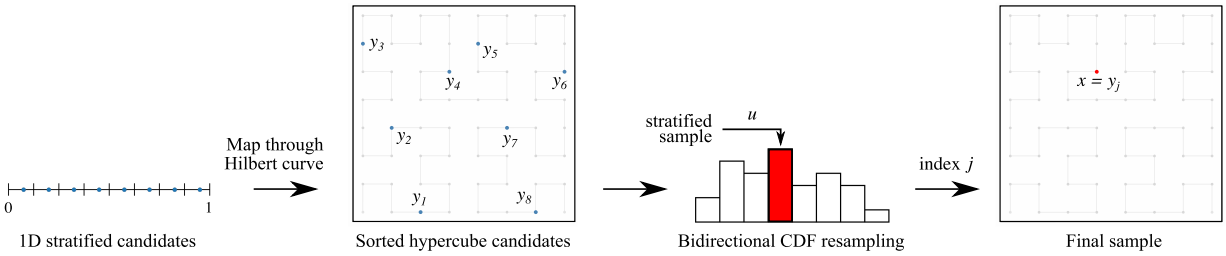


Figure 4.3: Overview of our algorithm. Sorted 1D stratified samples are mapped onto a space-filling curve to produce stratified candidates y_k on the unit hypercube. We resample these candidates along the curve proportionally to their associated weights using our on-the-fly bidirectional CDF sampling. Stratifying the resampling input u yields well-distributed output samples x in the integration domain.

4.2 Single-pass Bidirectional CDF Sampling

This section introduces a new discrete sampling algorithm that bands together the advantages of inverse CDF sampling and reservoir sampling. It scans the list of candidates only once, processing them one by one without keeping any in memory, similarly to reservoir sampling. Crucially, it can produce stratified output like inverse CDF sampling.

Reservoir sampling processes a list of candidates in a streaming manner. We additionally assume that it is possible to also read the list backward from its tail. As we will discuss later (Chapter 5), this additional constraint is typically not a limitation in practical rendering applications.

The list of candidates are processed one by one, from both ends. We keep track of a *front* index and a *back* index, respectively, and the corresponding running sums of weights

$\overline{w}_{front}$ and $\overline{w}_{back}$. Given an input sample $u \in [0, 1)$, at each step we increment *front* if $\overline{w}_{front} \leq u \cdot (\overline{w}_{front} + \overline{w}_{back})$, or decrement *back* otherwise, and update the corresponding running weight sum. The algorithm terminates when *front* = *back* which is taken as the sampled index j. We provide pseudocode in Alg. 6. Note that the algorithm does not require any precomputation and has a constant storage cost, just like reservoir sampling.

Algorithm 6: Our bidirectional CDF sampling algorithm.

Input : $\{w_1, \ldots, w_M\}$: weights, $u \in [0, 1)$: random number
Output: Sampled index

Function `BidirectionalCDF(`$\{w_1, \ldots, w_M\}$, u`):`
  $front \leftarrow 1$
  $back \leftarrow M$
  $\overline{w}_{front} \leftarrow w_{front}$
  $\overline{w}_{back} \leftarrow w_{back}$
  while *front* $\neq$ *back* **do**
    if $\overline{w}_{front} \leq u \cdot (\overline{w}_{front} + \overline{w}_{back})$ **then**
      $front \leftarrow front + 1$
      $\overline{w}_{front} \leftarrow \overline{w}_{front} + w_{front}$
    else
      $back \leftarrow back - 1$
      $\overline{w}_{back} \leftarrow \overline{w}_{back} + w_{back}$
  return *front*

In Chapter 6 we prove that the result of our algorithm is *exactly the same* as that of inverse CDF sampling. To that end, we show that in both algorithms, the output index j satisfies

$$\sum_{i=1}^{j-1} w_i \;\leq\; u \cdot \overline{w} \;<\; \sum_{i=1}^{j} w_i, \tag{4.2}$$

where $\overline{w} = \sum_{i=1}^{M} w_i$ is the sum of all weights. That is, both inverse CDF sampling and our bidirectional CDF sampling output the same index j given the same input sample u.

4.2.1 Taking Multiple Samples

When doing importance resampling, we often need to generate multiple samples ($N > 1$) from a given list of candidates. This is straightforward to do with inverse CDF sampling,

once the CDF has been computed and stored in memory. Reservoir sampling requires maintaining a reservoir of size N, feeding every candidate to each slot inside the reservoir. On the other hand, our bidirectional CDF sampling from Alg. 6 can generate only a single sample in one pass. This is because it traverses the candidates in an order that depends on the input sample u. The naive approach of regenerating the same set of candidates and performing N full passes for as many samples can be inefficient. To offer a better solution, we analyze the variance of resampling-based multi-sample MC integration.

The variance of the estimator in Eq. (4.1), taking N samples from M candidates, is [35]

$$\frac{1}{M}\operatorname{Var}\left(\frac{f}{p}\right)+\left(1-\frac{1}{M}\right)\frac{1}{N}\operatorname{Var}\left(\frac{f}{q}\right). \tag{4.3}$$

Recall that such sampling is not efficient to do with Alg. 6 as it would require N passes over all candidates. We therefore consider a variant of the estimator which takes one sample out of M/N candidates and we repeat this process N times to generate N samples. The total number of candidates is still $M/N \times N = M$. In other words, we split the set of M candidates into N subsets, take one sample from each subset, and average the N estimates. The variance of this alternative estimator is obtained from Eq. (4.3) by substituting $N \to 1$ and $M \to M/N$, and dividing the entire expression by N (i.e., the average over N repetitions). This yields

$$\frac{1}{M}\operatorname{Var}\left(\frac{f}{p}\right)+\left(1-\frac{N}{M}\right)\frac{1}{N}\operatorname{Var}\left(\frac{f}{q}\right). \tag{4.4}$$

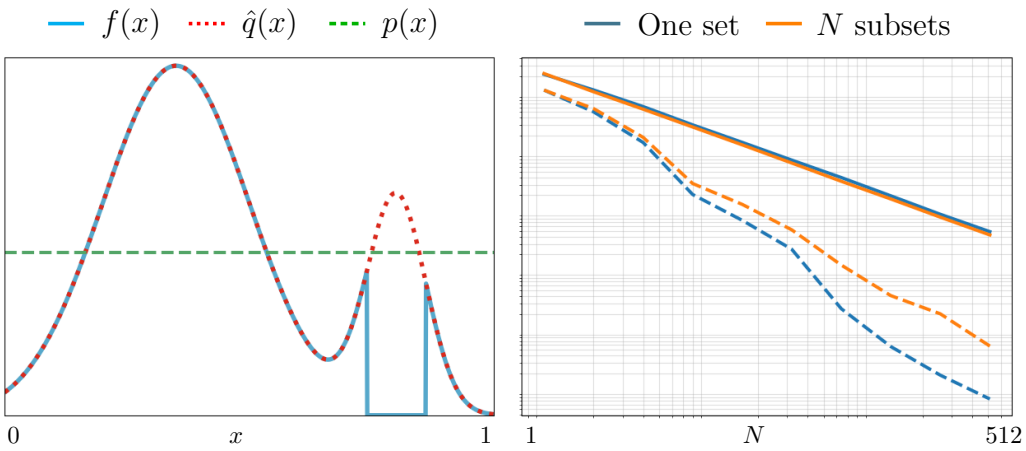
Interestingly, this variance is *smaller* than the variance (4.3) of the estimator which takes N samples out of the entire set of M candidates.

The above analysis shows that we never need to consider generating multiple samples from a single candidate set at once: taking one sample from each of multiple subsets is always better. It also outlines a simple recipe for taking multiple samples using our algorithm by sweeping over all candidates exactly once. Note that this recipe is, in fact, a *requirement* when using Alg. 6. In contrast, reservoir sampling does not require major changes to generate more than one sample in a single pass. Nevertheless, our variance analysis suggests that it too should benefit from candidate splitting.

In Figure 4.4 we numerically reproduce the results of the above variance analysis. We compare two variants of N-sample generation using our method: with repeated (One set) and split (N subsets) candidates. The former produces the same result as multi-sample inverse CDF sampling. The experiment is thus effectively as a comparison between inverse CDF sampling (One set) and our proposed single-pass method (N subsets). The integrand

f is zero in part of the domain, and the target function $\hat{q}$ mostly follows the integrand except for a small peak when f is zero. This setting can be interpreted as having two light sources where one is occluded and the target importance function excludes visibility. The candidate density p is uniform.

Using uncorrelated candidates and canonical input samples, subset resampling (solid orange curve in Fig. 4.4b) yields slightly lower variance, as the theory predicts. Stratifying candidates and samples (dashed curves) brings even more improvement, which is larger when not splitting the candidates. This result does not contradict the variance analysis which assumes fully independent sampling. Since repeating the same candidates may not be practical as we discussed above, we use the splitting approach in our experiments, even if it yields slightly higher variance.



(a) Experiment setup (b) Variance analysis

Figure 4.4: Variance analysis of 1D importance resampling using our bidirectional CDF method (Alg. 6). (a) Candidates are generated with density p and resampled according to $\hat{q}$ which better matches the integrand f. (b) Plots of integral-estimation variance as a function of sample count N, with $M = 4N$ candidates. When the candidates and input samples are independent, subset-based resampling (solid orange) yields lower variance than always resampling from the entire candidate set (solid blue), in line with the theory in Section 4.2.1. Stratifying the candidates and the input samples reduces variance further (dashed curves), though slightly less with subset-based resampling.

4.3 Ordering Candidates on a Space-Filling Curve

Our bidirectional CDF sampling algorithm can produce stratified indices, but this alone is not sufficient to achieve stratification after resampling. We need to ensure these indices map to stratified samples in the integration space. Typical candidate-generation orders, e.g., random, or the QMC order in Fig. 4.2b, do not provide a mapping with such correlation in the two different spaces. To that end, we propose to order the candidates along a space-filling curve.

Suppose we discretize the n-dimensional unit hybercube into a uniform grid with resolution 2^m along each axis and a total of 2^{nm} points. We can map a Hilbert curve onto that grid, which provides a bijective mapping between the index space $[1..2^{nm}]$ and the grid, allowing us to enumerate the grid points. Importantly, this mapping has the desirable property that any two points that are nearby along the curve are also nearby in the hypercube [30]. Steigleder and McCool [33] used this property to map 1D stratified samples to approximately stratify high-dimensional samples. We use it to generate stratified resampling candidates

$$y_k = \text{Hilbert}\left(\left\lfloor \frac{k}{M} 2^{nm} \right\rceil\right), \quad \text{where} \quad \text{Hilbert} : [1..2^{nm}] \to [0,1]^n \tag{4.5}$$

and where $\lfloor\cdot\rceil$ denotes rounding to the nearest integer. This analytic mapping allows enumerating candidates on-the-fly in the order along the curve. Since the Hilbert curve preserves proximity, stratified indices j map to stratified candidates y_j. We set the grid resolution to $m = 32$ to represent 32-bit floating-point coordinates.

We generate candidates directly on the Hilbert curve through Eq. (4.5), but this is not the only option. An alternative is to use a stratified hypercube sampling method (e.g., a Sobol sequence) to generate $M = 2^{nm}$ candidates, one per stratum, and enumerate the strata along a Hilbert curve. On-the-fly resampling then requires the ability to compute the sample location inside a given stratum [10].

4.4 Stratified Resampling Algorithm

The discrete sampling algorithm from Section 4.2 and the candidate ordering scheme from Section 4.3 can be combined to achieve efficient single-pass importance resampling that produces stratified samples.

Given a uniform sample $u \in [0,1)$, the resampling step R returns an index $j = R(u)$ by going over the candidates once. The final output sample is thus $x = y_{R(u)} \in [0,1)^n$. Feeding N stratified canonical samples u_i yields N stratified hypercube samples x_i. However, as discussed in Section 4.2.1, to avoid iterating over all M candidates for every sample, in practice we split the candidates into N subsets and resample from each. The splitting is done by interleaving the indices, i.e., the candidates for sample $i \in [1..N]$ are $\{y_{i+kN}\}_{k=0}^{M/N-1}$. In conjunction with the Hilbert-curve ordering, this interleaving effectively bins the candidates into M/N hypercube strata and puts one candidate from each into every subset. And since the N input canonical samples are stratified, the resampling selects N well-distributed strata (one per subset), producing N stratified output samples x_i.

4.4.1 Blue-noise Error Distribution

Apart from stratification, another way to effectively improve rendering quality is to distribute pixel error as blue noise over the image. Since our approach is based on 1D sampling, it can be easily combined with the dithering method of Georgiev and Fajardo [9]. Using the same set of N stratified canonical samples u_i for the entire image, the samples for pixel l are obtained through offsetting: $u_{l,i} = \mathrm{mod}(u_i + d_l/N, 1)$, where the offset $d_l \in [0,1)$ is taken from a dither mask [9].

Note that this sample dithering produces blue-noise error distribution only if there is a chain of correlation between u_i, x_i, and $f(x_i)$ [13]. Our stratified resampling method specifically aims to maintain high correlation between u_i and x_i, and we assume x_i and $f(x_i)$ are correlated (see discussion in Section 4.3). In reservoir-based resampling, the correlation between u_i and x_i is low, which inhibits both stratification and blue-noise error distribution.

If the same set of M candidates used for the entire image, aliasing may occur. To avoid it, we also dither the candidates along the Hilbert curve across pixels, by applying a fractional offset to the index k in Eq. (4.5). We then require a dither mask with two offsets per pixel [9], one for the candidates and one for the samples.

Chapter 5

Results

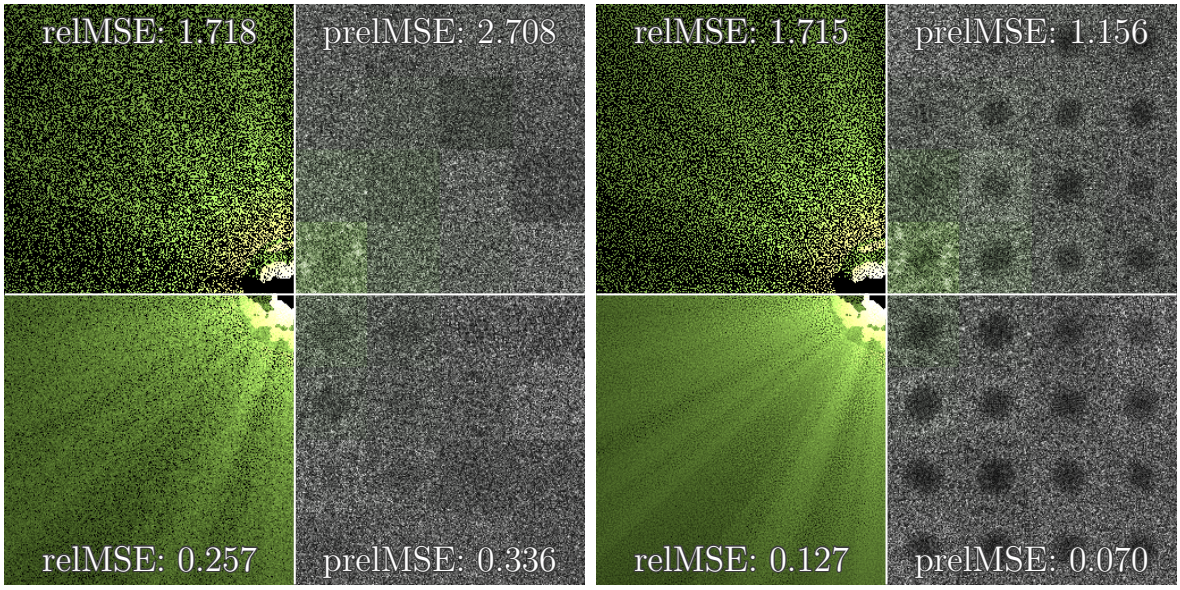
We implemented our method and other resampling strategies in the PBRT renderer [28]. Being agnostic to the rendering process, our method can be easily integrated into other rendering systems.

We show equal-sample-count comparisons since the overhead of our method is insignificant. We measure the relative mean squared error (relMSE): $E = \sum_{i=1}^{n} \frac{(e_i - r_i)^2}{(r_i^2 + \epsilon)}$, where r_i and e_i are respectively the reference and estimated values for the i-th pixel, and $\epsilon = 0.001$. We also show "perceptual" relMSE (prelMSE), which is the same metric but with the images first blurred with a 2D Gaussian kernel of standard deviation 2.1 pixels [6]. We additionally include tiled error power spectra to demonstrate that our method can generate a blue-noise error distribution when dithering is applied, similarly to Chizhov et al. [6]. We do not compare against inverse CDF sampling; while it can also produce stratified samples, it requires precomputation, and our goal is to stick to single-pass resampling.

5.1 Single Scattering in Media

Our first experiment renders single scattering from a point light in a homogeneous participating medium with isotropic phase function. This is a 1D integration problem per pixel where we can readily define 1D ordered candidates. We use $M = 32$ stratified dithered candidates, each representing a distance along a ray, distributed proportionally to transmittance. We resample them with weight equal to the unoccluded path contribution, a target function similar to existing resampling applications [35, 4].

Figure 5.1 compares our method and weighted reservoir sampling [5] with $N = 1$ and $N = 8$ final samples. Both methods use the same candidate sets and dithered stratified input samples. The tiled power spectra show that only our method produces a blue-noise error distribution. Also thanks to the stratification, our method has lower numerical and perceptual errors.



(a) Reservoir sampling (b) **Our method**

Figure 5.1: Importance resampled single scattering in a medium with $N = 1$ (top row) and $N = 8$ (bottom row) samples. Our method yields lower pixel and perceptual error than reservoir-based resampling, thanks to its effective sample stratification and blue-noise dithering. prelMSE values are scaled by 100×.

5.2 Direct Illumination

We also apply our resampling method to direct illumination from area emitters, where a candidate on the unit square maps to a point on an emitter [28]. The first dimension of the 2D candidate is used to choose an emitter. It is then rescaled so that the candidate can be used to sample the 2D point on the emitter. For the resampling target function, we again chose the unoccluded path contribution function. Figure 5.2 shows results with $N = 1$ sample from $M = 32$ candidates.

We use a 2D Halton sequence to generate candidates which we dither per pixel via 2D offsetting. We consider reservoir sampling with a stratified input (Fig. 5.2a) as a reasonable baseline. This method cannot produce blue-noise image-error distribution. We also show two other results where the candidates are reordered along a 2D Hilbert curve. Note that it is not the exact algorithm we propose since we actually store all the candidates and sort them afterward for this experiment. This is done only to demonstrate the importance of candidate ordering. Again, reservoir sampling fails to produce a blue noise error distribution (Fig. 5.2b) as its mapping of canonical samples to indices does not preserve locality. In contrast, our technique produces the desired error distribution from the same inputs (Fig. 5.2c).

Figure 5.3 shows another scene where we resample on the fly from $M = 32$ candidates coming from an (unsorted) Halton sequence or our Hilbert-curve distribution (Section 4.3). We apply dithering to both the candidates and the resampling ($N = 1$). Our ordered candidate generation and resampling together produce a blue-noise error distribution (Fig. 5.3a). However, we are unable to obtain blue noise if one of these components is missing, i.e., using unordered Halton-sequence candidates (Fig. 5.3b) or reservoir sampling (Fig. 5.3c).

5.3 Resampled Multiple Importance Sampling

Since we operate in primary sample space, to generate M MIS candidates from T techniques we can use the same set of M/T hypercube candidates, each producing one candidate per technique. This intentional correlation between the techniques facilitates stratification without adding bias. To draw N samples, we split the hypercube candidates into N subsets. The effective candidate count per subset is still M/N, as in non-MIS applications.

Figure 5.4 shows rendering of direct illumination for a scene lit with a complex environment map. We feed Halton-sequence and Hilbert-curve candidates to reservoir and bidirectional CDF resampling respectively. We combine emitter and BSDF techniques, and resample from $M = 32$ candidates (i.e., 16 hypercube candidates). Thanks to the careful stratification, our method produces lower error.

5.4 Higher-dimensional Integration

Our method easily scales to higher dimensions, by using a Hilbert curve of corresponding dimension. Figure 5.5 shows a 3D integration problem of computing single scattering from

multiple area emitters inside a homogeneous medium. The scene contains difficult visibility as the cage surrounding the main object heavily occludes light sources. We compare our method with 3D Hilbert-curve candidates to reservoir-based resampling of 3D Halton-sequence candidates. Here we do not observe blue noise even with our method. We believe this is due to the lack of sufficient correlation between resampling input and output. It still exhibits lower error with $N > 1$ samples thanks to their stratification.

5.5 Convergence

Figure 5.6 shows how error plots as functions of the number of final samples N and candidates M, with $M = 8N$. We compare our bidirectional CDF resampling and reservoir sampling with Hilbert-curve candidates to a reservoir baseline with Halton-sequence candidates, with dithering enabled for all. All three methods perform resampling in a single pass and have the same memory complexity. Our method performs best in all cases. Note that when the dimension of the integration problem increases, sample stratification becomes less effective and so does our method.

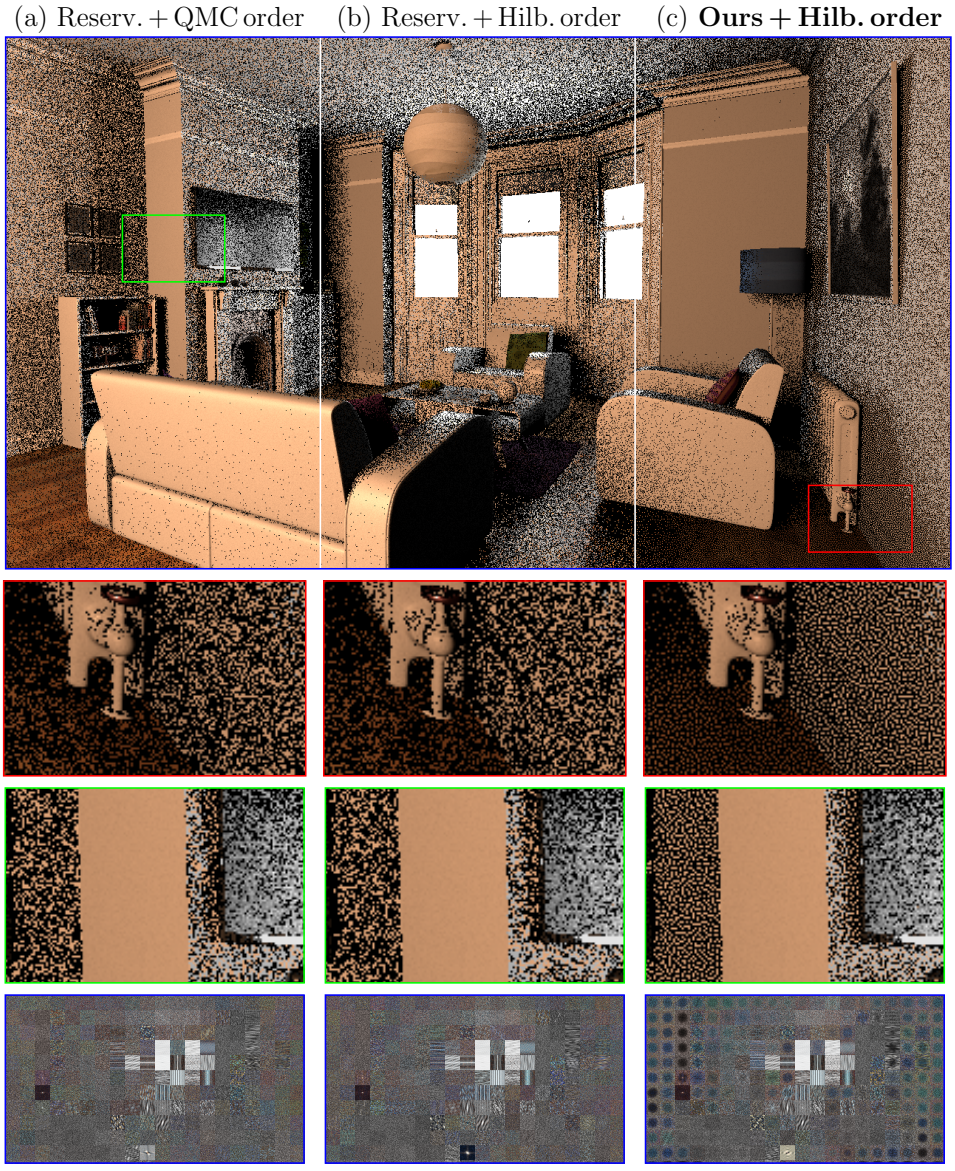
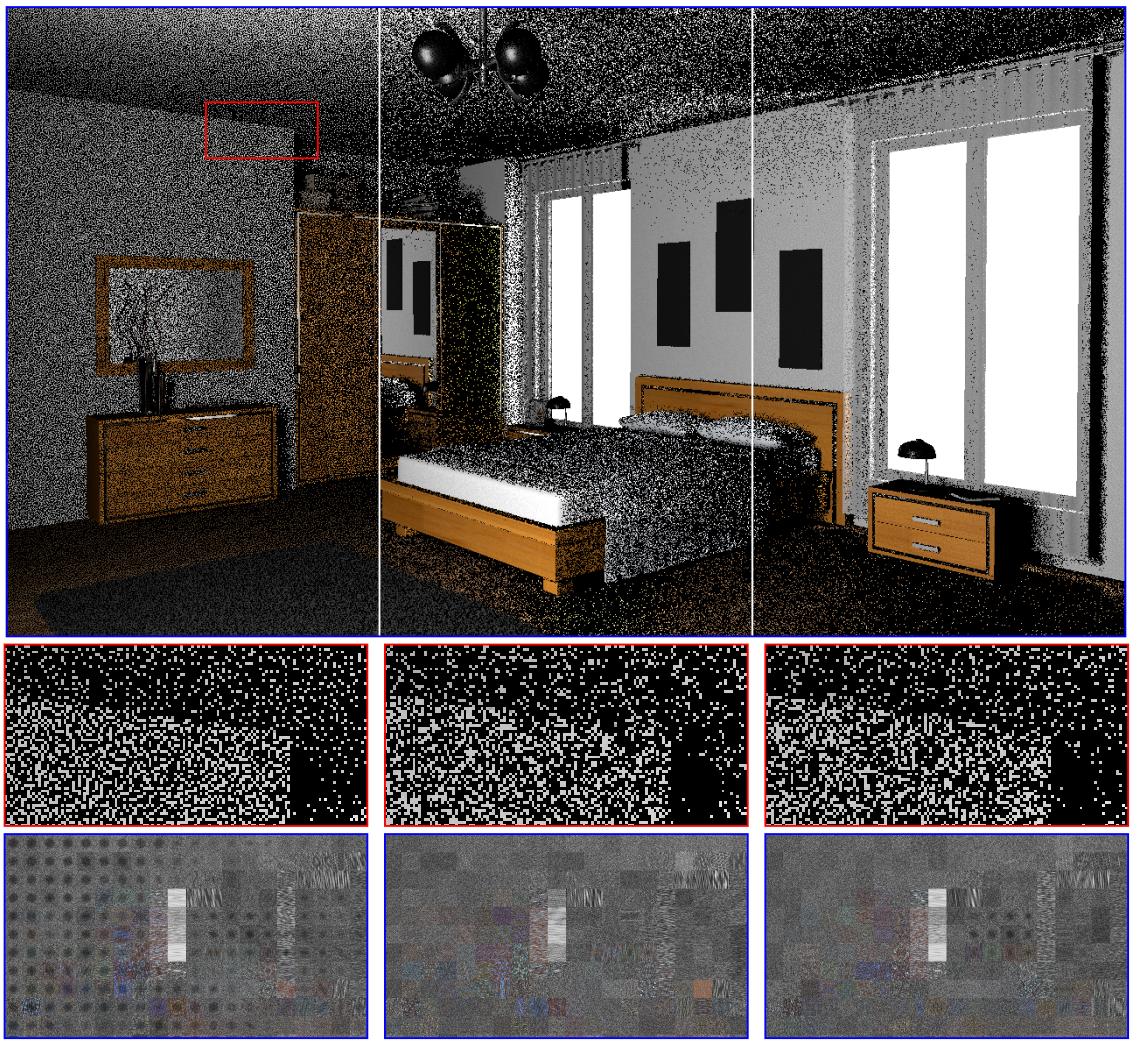


Figure 5.2: Comparison of our bidirectional CDF resampling against reservoir baselines on area-light illumination with $N = 1$ sample from $M = 32$ Halton-sequence candidates. Dithered reservoir resampling fails to produce a pleasing error distribution, whether using the candidate generation order (a) or along a Hilbert curve (b). In contrast, our dithered bidirectional CDF resampling with Hilbert-curve ordering (c) produces a high-quality blue-noise distribution.

(a) **Ours + Hilb. cand.** (b) **Ours + Halton cand.** (c) Reserv. + Hilb. cand.



prelMSE: 0.0272 prelMSE: 0.0396 prelMSE: 0.0401

Figure 5.3: Our candidate generation along a Hilbert curve, combined with our bidirectional CDF resampling, yields blue-noise error distribution (a). Using unsorted Halton-sequence candidates (b), or reservoir sampling (c), fails to attain such distribution.

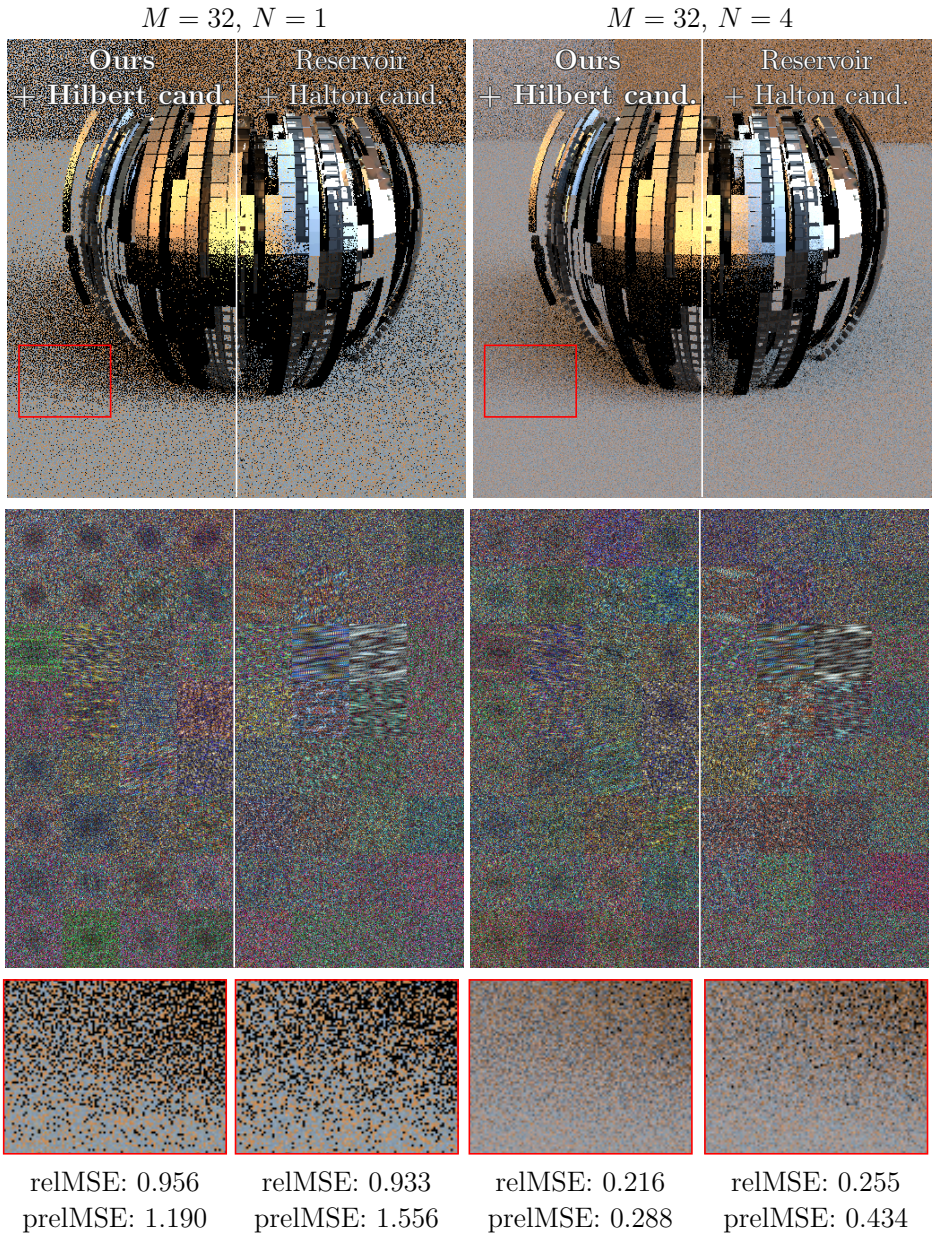


Figure 5.4: Resampling an MIS mixture of direct-illumination candidates. 16 unit-square candidates produce $M = 32$ hemispherical candidates via environment-map and BSDF warping. For both $N = 1$ and $N = 4$ samples, our technique produces a blue-noise error distribution and reduces the overall error over reservoir-based resampling. prelMSE values are scaled by $100\times$.

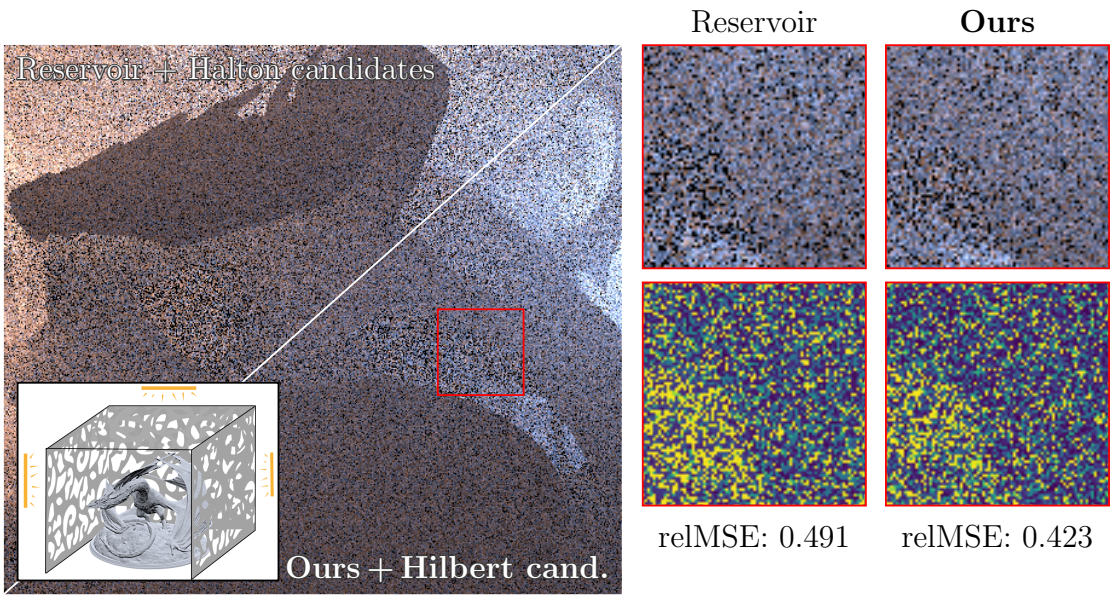


Figure 5.5: Single scattering from multiple area lights in a medium. We compare our bidirectional CDF resampling of Hilbert-curve candidates to reservoir resampling of 3D Halton-sequence candidates ($M = 32$, $N = 8$). We include false-color error zoom-ins.

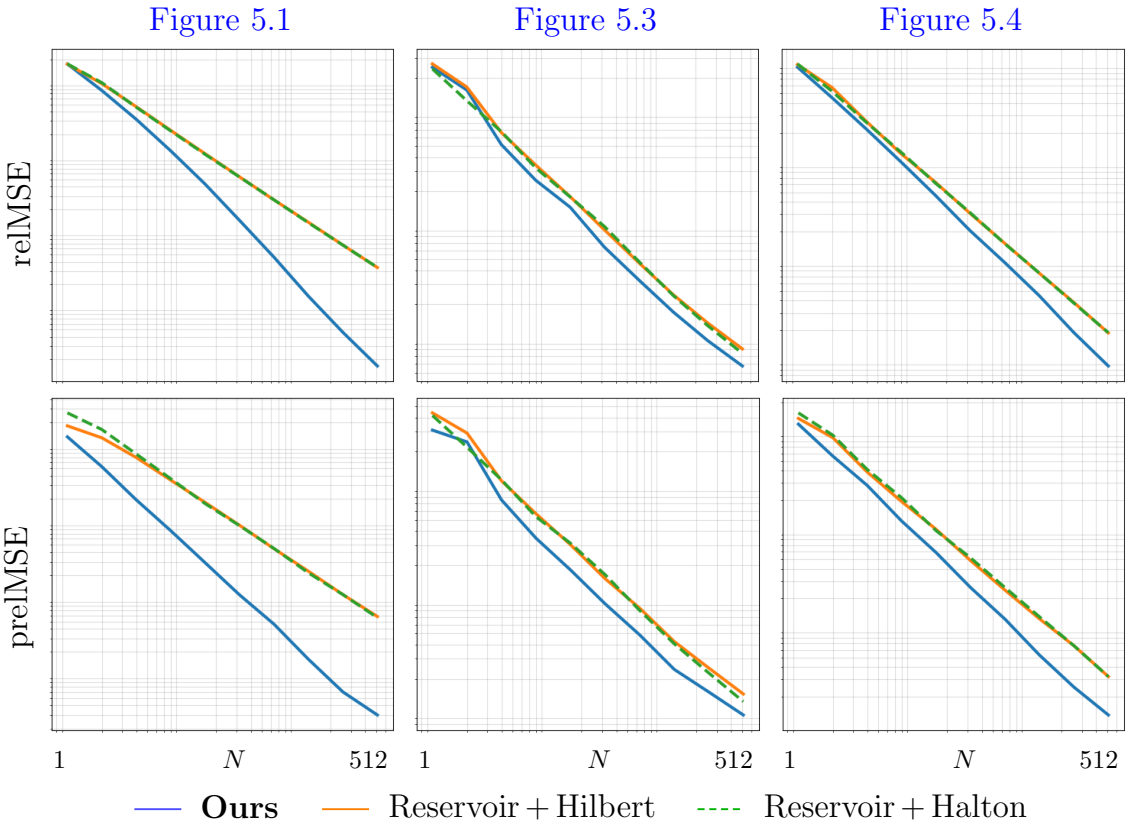


Figure 5.6: Error plots as functions of the sample count N for different scenes, with $M = 8N$ candidates. Our technique always outperforms reservoir sampling variants, achieving a better convergence rate. All of the methods shown are single pass.

Chapter 6

Conclusions

We have analyzed the issues in introducing stratification to resampling and demonstrated that a stratified sampling algorithm combined with carefully ordered candidates will yield well-distributed output samples after resampling. Accordingly, we have presented our bidirectional CDF sampling method and our candidate reordering scheme on the Hilbert curve.

Our bidirectional CDF sampling has similar properties as reservoir sampling since both methods do not need to store any candidates and avoid precomputation. We thus expect that our technique can replace reservoir sampling in many existing applications when stratification is desired.

Our work enables a combination of resampling and blue-noise error distribution for the first time. To that end, we rely on dithered sampling [9]. Other approaches also exist [1, 14, 2], and it would be interesting to investigate their utility. Since Monte Carlo denoising can benefit from blue-noise error distribution [13], our approach could also enable an efficient application of denoising to resampling.

Our work has a few limitations. While bidirectional CDF resampling achieves stratification in a single pass, it requires the ability to traverse the list of candidates from both ends, unlike reservoir resampling. Even though this requirement is typically not limiting in rendering applications, some applications may be incompatible with bidirectional CDF sampling.

Our method stratifies samples in the primary sample space, not the native integration domain. We assume that the mapping between the two preserves the stratification, as in quasi-Monte Carlo literature [17]. This assumption is empirically known to be violated in

higher dimensions [13]. For example, while it is conceivable to perform stratification over the 8D primary space of four consecutive direction-sampling decisions along a path, the resulting paths are unlikely to be stratified. This is not a problem for any method that operates in primary sample space.

We generate and resample candidates along a Hilbert curve with resolution 2^{nm} which is the number of points on the n-dimensional grid it traverses. In our experiments we have $m = 32$ and $n = 1..3$, but always use 32-bit numbers to offset and sample along the curve. This means that not all grid points might be sampled. At the tested moderate sample counts N this does not seem to be an issue, though ideally numbers with precision at least 2^{nm} should be used.

Uniform sampling along a Hilbert does not yield very high-quality stratification in the hypercube. The point sets lack some desirable properties such as well-stratified lower-dimensional projections. Devising a better ordering that achieves such properties is an important direction for future work.

Recent work has shown that reservoir resampling in combination with spatio-temporal sample reuse can be a very effective approach for rendering complex illumination [4]. It would be interesting to explore extending our approach to maintain stratification with spatio-temporal reuse, which our preliminary experiments show is not straightforward.

References

[1] Abdalla GM Ahmed and Peter Wonka. Screen-space blue-noise diffusion of Monte Carlo sampling error via hierarchical ordering of pixels. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. 53

[2] Laurent Belcour and Eric Heitz. Lessons learned and improvements when building screen-space samplers with blue-noise error distribution. In *ACM SIGGRAPH 2021 Talks*, SIGGRAPH '21, New York, NY, USA, 2021. Association for Computing Machinery. 32, 53

[3] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/. iv

[4] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4), July 2020. 3, 25, 27, 37, 44, 54

[5] M. T. CHAO. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 12 1982. 3, 25, 45

[6] Vassillen Chizhov, Iliyan Georgiev, Karol Myszkowski, and Gurprit Singh. Perceptual error optimization for monte carlo rendering. *ACM Trans. Graph.*, 41(3), 2022. 32, 44

[7] D. Cline, A. Razdan, and P. Wonka. A comparison of tabular pdf inversion methods. *Computer Graphics Forum*, 28(1):154–160, 2009. 24

[8] Pavlos S. Efraimidis. Weighted random sampling over data streams, 2010. 3

[9] Iliyan Georgiev and Marcos Fajardo. Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, New York, NY, USA, 2016. Association for Computing Machinery. 4, 31, 43, 53

[10] L. Grünschloß, M. Raab, and A. Keller. Enumerating quasi-monte carlo point sequences in elementary intervals. *Monte Carlo and Quasi-Monte Carlo Methods*, 2010. 42

[11] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, dec 1964. 21

[12] John M. Hammersley and David C. Handscomb. *Monte Carlo Methods*. Springer Dordrecht, 1964. 21

[13] Eric Heitz and Laurent Belcour. Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames. *Computer Graphics Forum*, 38, 2019. 32, 43, 53, 54

[14] Eric Heitz, Laurent Belcour, V. Ostromoukhov, David Coeurjolly, and Jean-Claude Iehl. A low-discrepancy sampler that distributes Monte Carlo errors as a blue noise in screen space. In *PROC SIGGRAPHTalks*, pages 1–2. PubACM, 2019. 53

[15] David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460, Sep 1891. 32

[16] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 143–150, New York, NY, USA, 1986. Association for Computing Machinery. 6, 11

[17] Alexander Keller. Quasi-monte carlo image synthesis in a nutshell. In Josef Dick, Frances Y. Kuo, Gareth W. Peters, and Ian H. Sloan, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 213–249, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 6, 21, 53

[18] Alexander Keller, Iliyan Georgiev, Abdalla Ahmed, Per Christensen, and Matt Pharr. My favorite samples. In *ACM SIGGRAPH 2019 Courses*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery. 32

[19] Alexander Keller, Simon Premoze, and Matthias Raab. Advanced (quasi) monte carlo methods for image synthesis. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery. 6

[20] Jack P. C. Kleijnen, Ad A. N. Ridder, and Reuven Y. Rubinstein. *Variance Reduction Techniques in Monte Carlo Methods*, pages 1598–1610. Springer US, Boston, MA, 2013. 6

[21] Adam Marrs, Peter Shirley, and Ingo Wald, editors. *Ray Tracing Gems II*. Apress, 2021. http://raytracinggems.com/rtg2. 27

[22] Michael D. McCool and Peter K. Harwood. Probability trees. In *Proceedings of the Graphics Interface 1997 Conference, May 21-23, 1997, Kelowna, BC, Canada*, pages 37–46, May 1997. 27, 35

[23] Don P. Mitchell. Consequences of stratified sampling in graphics. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 277–280, New York, NY, USA, 1996. Association for Computing Machinery. 18

[24] Harald Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of Number Theory*, 30(1):51–70, 1988. 6

[25] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992. 19

[26] Shinji Ogaki. Vectorized reservoir sampling. In *SIGGRAPH Asia 2021 Technical Communications*, SA '21 Technical Communications, New York, NY, USA, 2021. Association for Computing Machinery. 27, 35

[27] Art B. Owen. *Monte Carlo theory, methods and examples*. 2013. 6, 22

[28] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. PubMK, 3rd edition, 2016. 44, 45

[29] Donald B. Rubin. The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The sir algorithm. *Journal of the American Statistical Association*, 82(398):543–546, 1987. 28, 34

[30] Hans Sagan. *Space-Filling Curves*. Springer Science & Business Media, 2012. 42

[31] Peter Shirley. Discrepancy as a Quality Measure for Sample Distributions. In *EG 1991-Technical Papers*. Eurographics Association, 1991. 19

[32] Ilya M. Sobol. The distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7:86–112, 1967. 21, 35

[33] Mauro Steigleder and Michael D. McCool. Generalized stratified sampling using the hilbert curve. *Journal of Graphics Tools*, 8(3):41–47, 2003. 4, 33, 42

[34] Justin Talbot, David Cline, and Parris Egbert. Importance Resampling for Global Illumination. In Kavita Bala and Philip Dutre, editors, *Eurographics Symposium on Rendering (2005)*. The Eurographics Association, 2005. 3, 28, 35

[35] Justin F. Talbot. Importance resampling for global illumination. Masters thesis, Brigham Young University, 2005. 8, 30, 40, 44

[36] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1998. AAI9837162. 6, 13, 14, 18, 20

[37] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 419–428, New York, NY, USA, 1995. Association for Computing Machinery. 14

[38] Douglas Voorhies. I.8 - space-filling curves and a measure of coherence. In JAMES ARVO, editor, *Graphics Gems II*, pages 26–30. Morgan Kaufmann, San Diego, 1991. 33

[39] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, sep 1977. 24

Appendix

Equivalence of Inverse CDF Sampling and Bidirectional CDF Sampling

We show that Eq. (4.2) holds for the output index j. We first prove by induction that the following condition $\mathcal{C}$ is satisfied at any stage of the algorithm for the *front* and *back* indices:

$$\mathcal{C} \;:\quad \sum_{i=1}^{front-1} w_i \;\leq\; u \cdot \overline{w} \;<\; \sum_{i=1}^{back} w_i. \tag{1}$$

For the initial state, $front = 1$ and $back = M$, the inequalities hold trivially since $0 \leq u < 1$ and thus $0 \leq u \cdot \overline{w} < \overline{w}$.

Assume that $\mathcal{C}$ holds for indices $front < back$; we need to show that it still holds after one step of the algorithm. Recall that we keep track of the running sums $\overline{w}_{front} = \sum_{i=1}^{front} w_i$ and $\overline{w}_{back} = \sum_{i=back}^{M} w_i$. The algorithm handles the following two cases.

Case 1: If $\overline{w}_{front} \leq u \cdot (\overline{w}_{front} + \overline{w}_{back})$, we increment *front*. We then need to show that $\mathcal{C}$ still holds, i.e., that

$$\sum_{i=1}^{(front+1)-1} w_i \;=\; \sum_{i=1}^{front} w_i \;\leq\; u \cdot \overline{w} \;<\; \sum_{i=1}^{back} w_i. \tag{2}$$

The right inequality $u \cdot \overline{w} < \sum_{i=1}^{back} w_i$ holds by the inductive hypothesis since *back* remains unchanged. For the left inequality, noting that $(\overline{w}_{front} + \overline{w}_{back}) \leq \overline{w}$ for any $front < back$ and that $\overline{w}_{front} \leq u \cdot (\overline{w}_{front} + \overline{w}_{back})$ (the condition for this first case), we have

$$\sum_{i=1}^{front} w_i = \overline{w}_{front} \;\leq\; u \cdot (\overline{w}_{front} + \overline{w}_{back}) \;\leq\; u \cdot \overline{w}. \quad \square \tag{3}$$

Case 2: If $\overline{w}_{front} > u \cdot (\overline{w}_{front} + \overline{w}_{back})$, we decrement *back*. We then need to show that $\mathcal{C}$ still holds, i.e., that

$$\sum_{i=1}^{front-1} w_i \;\leq\; u \cdot \overline{w} \;<\; \sum_{i=1}^{back-1} w_i. \tag{4}$$

The left inequality $\sum_{i=1}^{front} w_i \leq u \cdot \overline{w}$ holds by the inductive hypothesis since *front* remains unchanged. For the right inequality, using the condition for this case $\overline{w}_{front} > u \cdot (\overline{w}_{front} + \overline{w}_{back})$, we have

$$\begin{aligned} \sum_{i=1}^{back-1} w_i = \overline{w}_{front} + \sum_{i=front+1}^{back-1} w_i \;&>\; u \cdot (\overline{w}_{front} + \overline{w}_{back}) + \sum_{i=front+1}^{back-1} w_i \\ &>\; u \cdot \left(\overline{w}_{front} + \overline{w}_{back} + \sum_{i=front+1}^{back-1} w_i \right) \;=\; u \cdot \overline{w}. \quad \square \end{aligned} \tag{5}$$

Therefore, the condition $\mathcal{C}$ holds at any stage of the algorithm. At the termination point, where the output index is $j = front = back$, that condition is identical to Eq. (4.2), and holds for j just like in inverse CDF sampling.