






Neural Sequence Transformation

Sabyasachi Mukherjee¹  Sayan Mukherjee^{2,3}  Binh-Son Hua⁴  Nobuyuki Umetani¹  Daniel Meister^{1†} 

¹The University of Tokyo ²University of Illinois at Chicago ³blueqat Co. Ltd. ⁴VinAI Research and VinUniversity

Abstract

Monte Carlo integration is a technique for numerically estimating a definite integral by stochastically sampling its integrand. These samples can be averaged to make an improved estimate, and the progressive estimates form a sequence that converges to the integral value on the limit. Unfortunately, the sequence of Monte Carlo estimates converges at a rate of $O(\sqrt{n})$, where n denotes the sample count, effectively slowing down as more samples are drawn. To overcome this, we can apply sequence transformation, which transforms one converging sequence into another with the goal of accelerating the rate of convergence. However, analytically finding such a transformation for Monte Carlo estimates can be challenging, due to both the stochastic nature of the sequence, and the complexity of the integrand. In this paper, we propose to leverage neural networks to learn sequence transformations that improve the convergence of the progressive estimates of Monte Carlo integration. We demonstrate the effectiveness of our method on several canonical 1D integration problems as well as applications in light transport simulation.

CCS Concepts

• **Mathematics of computing** → Numerical analysis; Probability and statistics; • **Computing methodologies** → Machine learning algorithms; Ray tracing;

1. Introduction

Numerical integration is one of the most ubiquitous problems in modern scientific computing. Integrals that cannot be analytically computed are abundant in nature, and thus such integrals are often approximated using numerical methods such as Monte Carlo integration [MU49]. Monte Carlo integration stochastically generates a sequence of estimates that converges to the integral value. While Monte Carlo integration is a powerful technique independent of the dimensionality of the integrand, its convergence is typically slow, in the order of $O(\sqrt{n})$, where n is the number of samples taken. In other words, to obtain a result with half the error, four times as much computational effort is required. Extensive research has been done on improving the convergence of Monte Carlo estimates.

On the other hand, the mathematical branch of numerical analysis provides a set of techniques known as sequence transformation methods to improve the convergence of analytical sequences. A sequence transformation aims to transform a convergent sequence into another with a higher convergence rate by exploiting properties of the initial sequence. Euler's transformation, Aitken's Δ^2 transformation [Ait27], Levin's u -transformation [Lev72], Brezinski's θ transformation [Bre71] are some of the widely used transformations in the literature. How fast a sequence converges determines how well a particular transformation improves its convergence. This is

quantified by the *convergence rate* of the sequence. Depending on its rate, a convergent sequence is classified into three major types: hyperlinear, linear and sublinear. A further subclass of sublinear convergence is logarithmic convergence.

The sequence of estimates in Monte Carlo integration is a stochastic sequence that converges to the value of the integral. Our key observation is that sequence transformation techniques can potentially improve the convergence of this sequence of estimates. A major challenge in this formulation is to determine the type of convergence of the sequence of Monte Carlo estimates, as well as estimating error bounds for the transformed sequences. Although the concept of type of convergence does not directly translate to stochastic sequences, we can show that Monte Carlo estimates are convergent close to a logarithmic rate (see Appendix A). This brings us to the next major challenge: very few logarithmically convergent sequences have been proved to achieve better convergence via analytical sequence transformations.

To overcome these limitations of analytical transformations, we propose a data-driven approach for sequence transformations based on neural networks. The neural network takes as input a sequence of Monte Carlo estimates, and generates a new sequence that converges faster to the value of the integral. A key component of our method is a novel loss function that optimizes a suitably weighted relative error and leads to better convergence behavior as a result.

To demonstrate our technique, we apply the data-driven sequence

† JSPS International Research Fellow

transformation to two different Monte Carlo integration problems: integration of one-dimensional Gaussian, step and products of Gaussian and step functions, as well as integration of various higher dimensional functions in light transport simulation. Our estimator provides improved convergence of Monte Carlo integration in both these cases, especially for lower sample counts where improved estimates are the most beneficial.

Our main contributions can be summarized as:

- Designing a feed-forward neural network to realize data-driven sequence transformation,
- Proposing a loss function tailored for sequence transformation of the Monte Carlo integration process, and
- Showing an analysis formalizing the near-logarithmic convergence rate of Monte Carlo integration.

2. Background and Related Work

2.1. Monte Carlo Integration

Let $f : [a, b] \rightarrow \mathbb{R}$ be a function that satisfies $\int_a^b f(x)^2 dx < \infty$. We aim to approximate the definite integral $I = \int_a^b f(x) dx$, where I is known as the *reference value*. Monte Carlo integration finds an estimate of I by first drawing n samples X_1, \dots, X_n based on a *probability density function* (PDF) $p(x)$ supported on $[a, b]$, and then computing the following estimate:

$$S_n = \frac{1}{n} \sum_{i=1}^n \frac{f(X_i)}{p(X_i)}, \quad (1)$$

where S_n is an n -sample *estimate* of the integral. It can be shown that $\mathbb{E}[S_n] = I$, and, via the law of large numbers, S_n converges to the integral value I as $n \rightarrow \infty$.

While Monte Carlo estimation is rather simple to implement, there are some inherent limitations. Firstly, the quality of the estimate S_n depends on the choice of $p(x)$ and the sample count n . Moreover, the convergence rate of the estimator S_n is known to be $O(\sqrt{n})$, slowing down as n becomes large. Several methods to improve the efficiency of Monte Carlo integration have been developed, including variance reduction techniques such as *importance sampling* and *control variates*, as well as more advanced sampling techniques such as *quasi-Monte Carlo* (QMC) and *Markov chain Monte Carlo* (MCMC).

2.2. Sequence Transformation

A *sequence* is an ordered set of real numbers. We denote a sequence of $n \in \mathbb{N}$ elements as $(s_n) = \{s_1, \dots, s_n\}$. A sequence (s_n) is said to be *convergent* if its terms approach a real number as n grows large. A *sequence transformation* \mathcal{T} is a mapping of a sequence (s_n) into another sequence (t_n) such that (t_n) potentially converges faster than (s_n) .

Various sequence transformation methods have been devised based on their *type of convergence*. Weniger [Wen89] defines this quantity as $\rho = \lim_{n \rightarrow \infty} \left| \frac{s_{n+1} - s}{s_n - s} \right|$ for any sequence (s_n) that converges to s . The type of convergence is said to be *hyperlinear* if $\rho = 0$, *linear* if $\rho \in (0, 1)$, and *sublinear* if $\rho = 1$. Further, if $\rho = 1$ and

$\lim_{n \rightarrow \infty} \left| \frac{s_{n+1} - s}{s_n - s} \right| = 1$, the convergence of (s_n) is called *logarithmic*. When $\rho > 1$, (s_n) diverges.

Convergent sequences with hyperlinear convergence do not benefit from sequence transformation [Wen89, p. 11], whereas convergent sequences with logarithmic convergence are difficult to accelerate, with examples few and far between (see [SF79, Kow81, Sab87, Rie90, Sed90, Osa96]). Depending upon the type of convergence of a sequence, different sequence transformations are used to improve the convergence. There is a large amount of literature on sequence transformation methods, and we refer the reader to works by Brezinski and Redivo-Zaglia [BRZ20], Weninger [Wen89] and Sidi [Sid03, Sid17] for details of different sequence transformation methods in use.

In computer graphics, a very successful application of sequence transformation can be seen in the context of geometry processing and physics simulation [PDZ*18], where the authors apply Anderson acceleration [And65] for improving the convergence of fixed point methods. However, Monte Carlo integration is difficult to formulate as a fixed point problem since we do not know the value of the integral beforehand. Therefore, Anderson acceleration cannot be used directly in our scenario.

2.3. Sequence Transformation with Monte Carlo Integration

Let us consider the Monte Carlo estimates of the integral $\int_a^b f(x) dx$ for a function $f : [a, b] \rightarrow \mathbb{R}$ as the sequence of random variables in Equation 1, and we are interested in applying sequence transformation to improve its convergence. However, calculating the type of convergence of the estimates in Equation 1 is not straightforward. This is because Monte Carlo integration is a stochastic process, and the concept of type of convergence has only been formulated for analytic sequences. In Appendix A, we use expected values of the random variables involved in the sequence to infer that the convergence of Monte Carlo integration can be best approximated as logarithmic in nature.

According to Delahaye and Germain-Bonne [DGB82], there does not exist any universal sequence transformation method that can accelerate all sequences. Specifically, there does not exist any universal sequence transformation method that can accelerate all sequences with logarithmic convergence. Further, only a few logarithmic sequences have been successfully accelerated, as mentioned in Section 2.2. Previous work that achieves acceleration for Monte Carlo integration, such as [O'B92, Lav94, WHKH13, JASF15, DH20] are scenario-specific, and only few use sequence transformations.

Brezinski and Zaglia [BZ91, BZ13] propose an analytical sequence transformation for Monte Carlo integration, which we call *the $a_n g_n$ transformation*. It is given by:

$$T_{n+1} = S_n - \frac{S_{n+1} - S_n}{g_{n+1} - g_n} g_n, \quad (2)$$

where g_n can be any function of n .

However, a variance analysis presented in Appendix A shows that this transformation does not accelerate Monte Carlo integration. In particular, one can show that the transformed sequence has slightly *higher* variance than the sequence of Monte Carlo estimates regardless of the choice of g_n .

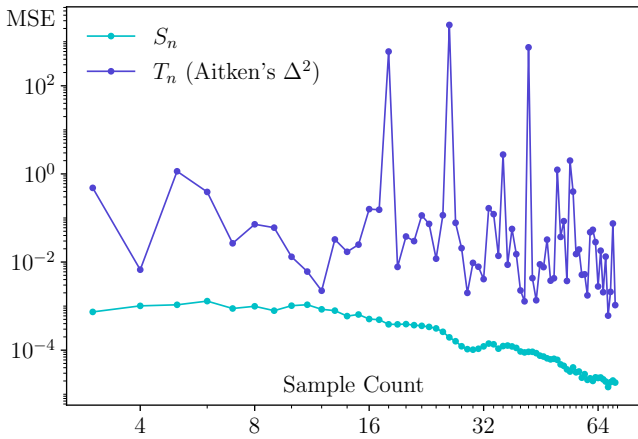


Figure 1: Mean Squared Error (MSE) for the output of Aitken’s Δ^2 method when integrating a step function. S_n denotes the input sequence of Monte Carlo estimates and T_n denotes the transformed sequence.

As an investigation into analytical sequence transformation methods, apart from the theoretical limitations of other sequence transformation methods such as Levin’s u -transformation and the $a_n g_n$ transformation, we implement and test Aitken’s Δ^2 -method on 1D integrands which are step functions. The results in Figure 1 show that no acceleration is achieved. Elek et al. [ETF19] attempted to reduce the variance of Monte Carlo integration by learning the patterns in the sample distribution, however, their technical report does not provide sufficient details on the convergence and quantitative results. These observations motivate us to consider data-driven methods to achieve sequence transformation for Monte Carlo integration.

3. Neural Sequence Transformation

3.1. Network Architecture

Our proposed method is inspired from the concept of sequence transformation. As there is no universal sequence transformation and it is challenging to design one for Monte Carlo estimates, we seek a data-driven approach by learning to transform a sequence using a neural network. More precisely, neural sequence transformation is a regression problem where a neural network takes convergent sequences as input, and attempts to output a sequence that converges faster. In order to do so, we design a fully-connected neural network that takes in M values of an input sequence and produces a better estimate of the converged value as output.

By viewing Monte Carlo estimates as a convergent sequence, i.e., (S_n) (Equation 1), we can apply a sliding window of length M to obtain an input sequence $\{S_{i-M+1}, \dots, S_i\}$ that can be transformed by the network to predict the improved value T_i . This results in a transformed sequence that potentially converges faster than the original sequence. Note that the output of the network is labelled as T_i instead of T_{i-M+1} for a fair comparison since we can only perform this prediction after S_i is computed.

Our choice of using partial sums as input to the network instead of

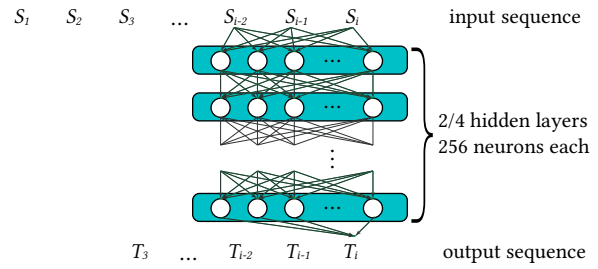


Figure 2: Illustration of our neural sequence transformation network that takes an input sequence of three samples $\{S_{i-2}, S_{i-1}, S_i\}$ as an example and outputs the transformed value T_i . The network architecture is an ordinary fully connected MLP.

unordered samples is driven by two factors. Firstly, the formulation of sequence transformation usually takes a convergent sequence and transforms it into another sequence that potentially converges faster. Secondly, in a sliding window framework with unordered samples as input, a neural network can have access to only a few raw samples (the window size), whereas partial sums contain richer information about the integration process that can help accelerate the convergence.

We model our sequence transformation as a *Multi-Layered Perceptron* (MLP) neural network as such a network has the capability to approximate a wide variety of functions [HSW89]. Specifically, our network has two or four fully-connected hidden layers, each containing 256 neurons and a leaky ReLU activation function (with threshold 0.01). An illustrative diagram of the network is shown in Figure 2.

3.2. Loss Function

A commonly used loss function for training neural networks is the *Mean Squared Error* (MSE). Although it works very well in several scenarios, it is not specific to our use case of Monte Carlo integration. Instead, we propose a loss function tailored to the sequence transformation of Monte Carlo estimates to potentially achieve better performance.

In Monte Carlo integration, the convergence rate is defined as the ratio of the logarithm of the MSE to the logarithm of the sample counts. Ideally, this rate is -1 in the limit for Monte Carlo and -2 for quasi-Monte Carlo integration. Therefore, in a log-log plot of the MSE vs. sample count, these graphs are ideally straight lines with slopes -1 and -2 , respectively.

The more negative the convergence rate of a transformed sequence, the better it performs as an estimate; additionally, the MSE of the transformed sequence should be lower than that of the original sequence. Without this additional constraint of having lower MSE, the slope of the transformed sequence in the convergence graph will be well optimized, but the output sequence might be arbitrarily shifted above the original sequence, which is not desirable for practical use. In our initial setup, we tried to fit a line through the points in the log-log convergence graph of MSE vs. sample counts using linear regression, where the line is defined by the slope and

the intercept. Nonetheless, it turned out that optimizing the slope and intercept directly requires complex mathematical operations (i.e., linear regression in logarithmic space) during the loss function computation. This results in unstable and slow training. Thus, we design a loss function that is simpler and yet works in accordance with our goal.

The key idea is to operate directly on the error estimates (i.e., points in the convergence graph) to avoid performing linear regression. Due to the stochastic nature of Monte Carlo integration, the values in a sequence are noisy estimates of the integral, which can result in a noisy loss function during training. To improve stability, we first estimate the error for the original and transformed sequences by averaging over multiple runs of the Monte Carlo method:

$$\mathcal{E}_{S_n} = \frac{1}{R} \sum_{r=1}^R (S_n^r - I)^2, \quad (3)$$

$$\mathcal{E}_{T_n} = \frac{1}{R} \sum_{r=1}^R (T_n^r - I)^2, \quad (4)$$

where R is the number of runs, I denotes the reference value for the given integral, T_n^r and S_n^r denote the n -th term of the r -th run of the output and original sequence, respectively. For the original sequence, the error \mathcal{E}_{S_n} is equal to the variance as these estimates are unbiased. However, for the transformed sequence, the error \mathcal{E}_{T_n} is mixture of bias and variance since the neural network may introduce some bias. For the sake of simplicity, we suppose that $k(n) = n$ in the following derivations. Recall that M denotes the size of the sliding window for our network. We define the loss function for the n -th point of the sequence as:

$$\mathcal{L}_n = \mathcal{E}_{T_n}^{1/\log(n)} / (\mathcal{E}_{S_n} + \epsilon), \quad (5)$$

where ϵ is a small positive constant to prevent division by zero. The loss function for the whole sequence is defined as a sum of loss values of the individual points:

$$\mathcal{L} = \sum_{n=M}^N \mathcal{L}_n = \sum_{n=M}^N \mathcal{E}_{T_n}^{1/\log(n)} / (\mathcal{E}_{S_n} + \epsilon). \quad (6)$$

Note that this is the loss function for a single integrand. To compute the total loss, we further take the mean of these values over all integrands, which would require an additional index corresponding to the integrand. For the sake of simplicity, we omit this additional index as taking the mean is straightforward. It is also important to realize that we need to transform the whole sequence first to compute the loss function as the neural network transforms only a partial sequence inside the sliding window (see Figure 2).

Further Analysis: The ratio of the error of the transformed and original sequences $\mathcal{E}_{T_n}/\mathcal{E}_{S_n}$ corresponds to the the signed distance between points in the convergence graph as division translates to difference in logarithmic space. Minimizing just this ratio would result in shifting each point of the original sequence down below by a constant offset in the convergence graph while preserving the same slope as the original sequence. Therefore, in order to improve the slope as well, we use the $\log n$ -th root of \mathcal{E}_{T_n} to incorporate the slope directly into the loss function. This is more clear if we transform

Equation 5 to the logarithmic space:

$$\log \mathcal{L}_n = \frac{\log(\mathcal{E}_{T_n})}{\log(n)} - \log(\mathcal{E}_{S_n} + \epsilon). \quad (7)$$

Let us assume a linear approximation $\log \mathcal{E}_{T_n} \approx a \log n + b$ in log-log space, and note that $\log(\mathcal{E}_{S_n}) \approx \log V - \log n$ for Monte Carlo integration, where V is the variance of a single sample $f(X_i)/p(X_i)$ from Equation 1. Then, we can rewrite Equation 7 into:

$$\log \mathcal{L}_n \approx a + \frac{b}{\log(n)} - \log V + \log n. \quad (8)$$

For large n , it is seen that $\frac{b}{\log n}$ becomes negligible, and $\log V$ is a constant, implying an optimization problem where $a + \log n$ is minimized for large n . Since $\log n$ contributes more to the loss function for large n , the slope a is not as important to our network at larger sample counts. This leads to a flattening out in the $\log \mathcal{E}_{T_n}$ - $\log n$ graphs in all our results. However, in practice, for large n , \mathcal{E}_{S_n} itself becomes small, and we need not focus on minimizing the slope a at that point. Conversely, when n is small, say around 10, we minimize $a + b$, whence the optimizer gives more focus on the slope a compared to when minimizing $a + \log n$. This leads to a sharp drop in error at lower sample counts in cases where the intercept b is small.

Although the behavior of our loss function is easier explained with the help of Equation 7, we observe that using logarithms in the loss function leads to unstable training. Thus we directly optimize the quantity \mathcal{L} in Equation 6 instead.

4. Results

Our method was implemented in PyTorch Lightning [Fal19] on top of PyTorch [PGM*19]. In our implementation, we use the Adam optimizer [KB14] and set the learning rate to 1×10^{-4} for 1D integrands, and 3×10^{-4} for images. For generating the images in PBRT, we turn off anti-aliasing. Unless otherwise stated, we use the random sampler to generate our input sequences, and a sliding window of size 8.

We remark here that equal-sample comparison was used to evaluate the performance of our method. At inference, our neural sequence transformation is applied as a post-processing step at the end of the Monte Carlo integration. Network training is regarded as a one-time precomputation cost, and not counted in the evaluation. In light transport applications, our network takes 6.5 ms per 128×128 image on an RTX 2070 GPU at inference (in PyTorch), whereas rendering an image with the same resolution with 32 spp takes about 300 ms on a hexa-core Ryzen 5 5600X CPU (in PBRT). Thus, the overhead of our method is small compared to rendering the source images, which makes equal-time comparison to be similar to equal-sample comparison.

4.1. 1D Integration

We run our method on several 1D integration problems, and compare the MSE against traditional Monte Carlo integration. Motivated by Christensen et al. [CKK18], we apply our method on three different classes of functions: step functions, Gaussian functions and Gaussian functions multiplied by a step function. These families

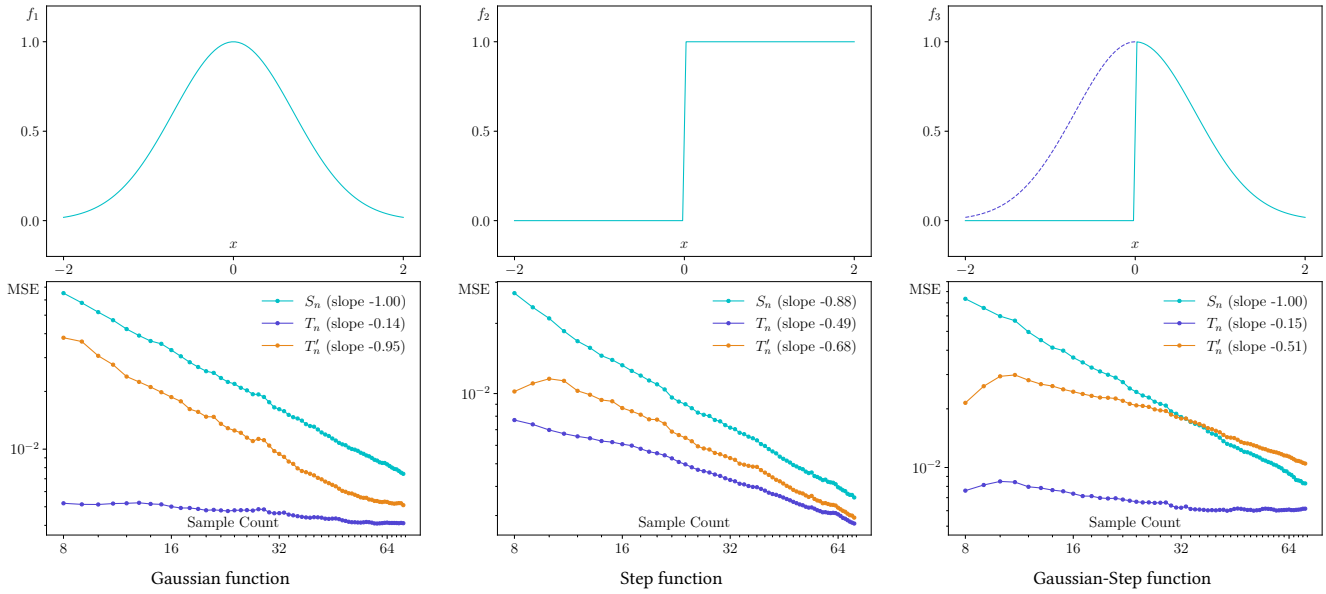


Figure 3: Results of applying our method to three different classes of 1D functions: (from left to right) the Gaussian function, the step function and the Gaussian times a step function. S_n denotes the input sequence, T_n denotes the transformed sequence when the network is trained using only the dataset for the corresponding function, and T'_n denotes the transformed sequence when the network is trained on a dataset containing all three functions. We observe that while the efficacy of our method decreases as the number of samples is increased, for lower sample counts, the network is very good at predicting the converged value of the sequence. In general, T'_n seems to perform worse than T_n ; however, it still achieves improvement in convergence.

of discontinuous and continuous functions appear frequently in the context of light transport simulation. The shapes of these functions are shown in Figure 3. For 1D Gaussian functions, we train on 100 runs of 81 integrands. Each integrand corresponds to a pair (μ, σ) uniformly covering the square $[0.1, 0.3] \times [0.1, 0.3]$, where the function is of the form $\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. We test on 100 runs of 9 such new integrands corresponding to (μ', σ') , different from the training data, by shifting them through adding a 0.05 offset. The net error of our network is computed as the average error of each transformed sequence generated over all runs of all integrands.

We train the network for 1000 epochs with batch size 12, and the results are shown in Figure 3. As can be seen, there is an improvement on the convergence of the transformed sequence predicted by the network. Additionally, we found that the improvement is more apparent at low sample count. At high sample count when the input sequence is close to convergence, the improvement becomes more negligible.

4.2. Ablation Study

Our loss function for the n -th point of the sequence is given by $\mathcal{L}_n = \mathcal{E}_{T_n}^{1/\log(n)} / (\mathcal{E}_{S_n} + \epsilon)$. It consists of a relative weighting of the error \mathcal{E}_{T_n} over the input error \mathcal{E}_{S_n} . To observe the effect of changing the relative weight over the entire sequence, we compare the performance of the following loss functions via their convergence

graphs:

$$\mathcal{L}_n^0 = \mathcal{E}_{T_n}, \mathcal{L}_n^1 = \frac{\mathcal{E}_{T_n}}{\mathcal{E}_{S_n} + \epsilon}, \mathcal{L}_n^2 = \frac{\mathcal{E}_{T_n}^{1/\log(n)}}{\mathcal{E}_{S_n} + \epsilon}, \mathcal{L}_n^3 = \mathcal{E}_{T_n}^{1/\log(n)} \quad (9)$$

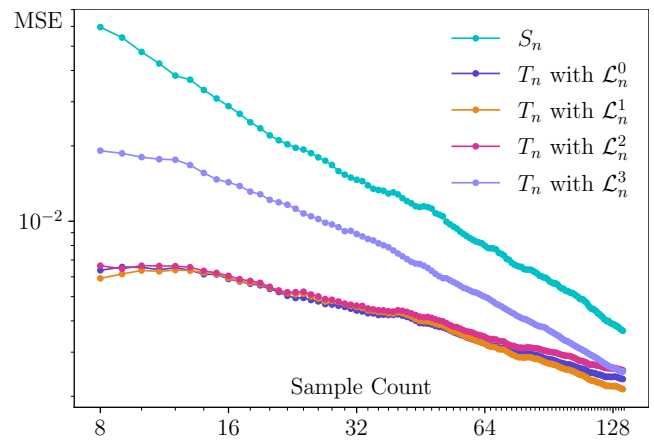


Figure 4: Ablation study. We compare the performance of different loss functions on the 1D integration of a Gaussian integrand. \mathcal{L}_n^0 is the MSE, \mathcal{L}_n^1 is the relative MSE, \mathcal{L}_n^2 is our proposed loss, and \mathcal{L}_n^3 is the absolute (non-relative) version of our proposed loss. The network was trained on a dataset containing all three 1D functions of Figure 3.

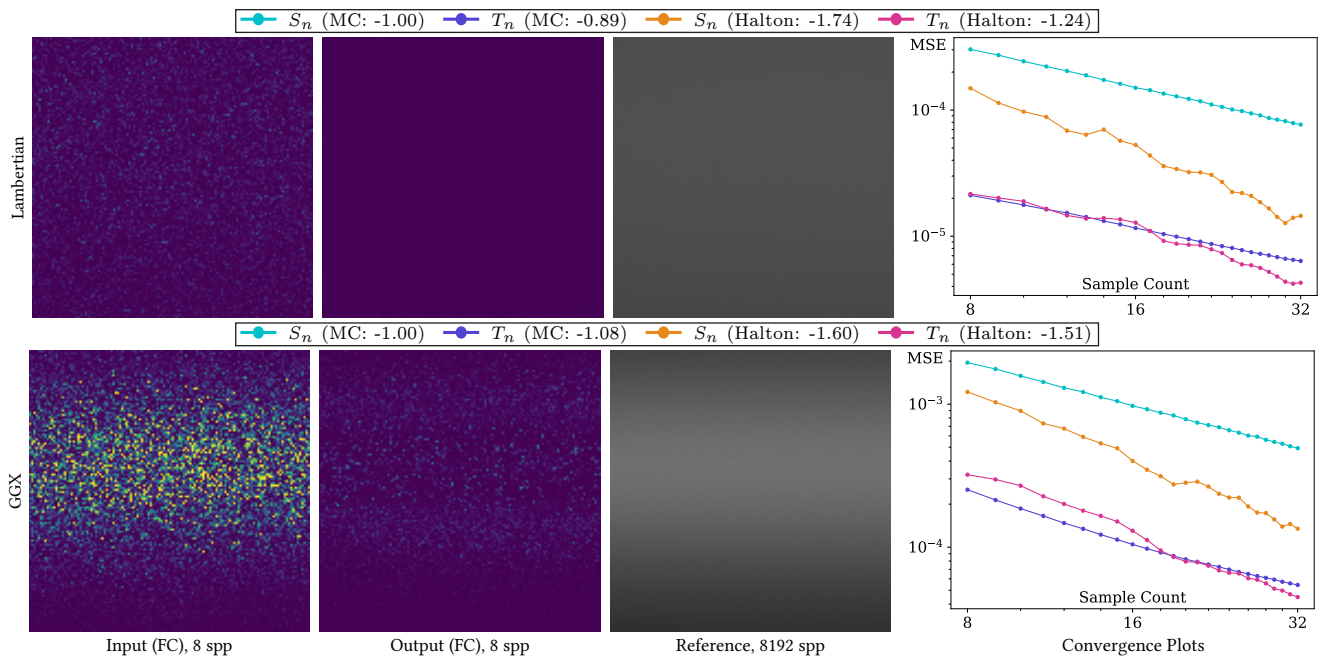


Figure 5: Results for a simple scenario in light transport simulation. The camera view is perpendicular to a plane lit by a rod-shaped area light (i.e., an elongated rectangular area light source) behind the camera. We use two simple materials: diffuse Lambertian (top) and Disney BRDF with GGX material (bottom). The first two subfigures in each row depict the error maps (false color) between the input and reference and the output and reference respectively. As can be seen, our method improves both input sequences produced by Monte Carlo (MC) and quasi-Monte Carlo (Halton) estimation.

Observe that \mathcal{L}_n^0 is the MSE, \mathcal{L}_n^1 is the relative MSE, \mathcal{L}_n^2 is our proposed loss function, and \mathcal{L}_n^3 does not contain the relative factor of the input error. We evaluate the performance of these loss functions on the Gaussian integrand considered earlier in Figure 3. The plot in Figure 4 suggests that \mathcal{L}_n^3 performs worse than its relative versions. While the choice between \mathcal{L}_n^0 , \mathcal{L}_n^1 and \mathcal{L}_n^2 is much more unclear, we decided to use \mathcal{L}_n^2 as it matches our theoretical derivations.

Designing a loss function that consistently achieves faster convergence is difficult. Via the loss function, we need to optimize two parameters (i.e., slope and intercept of a best fit line of the convergence graph) at once. However, loss functions which directly optimize the slope, intercept, or linear combinations of the two do not give reasonable results. Moreover, all loss functions that were tested in this work perform similar to MSE. Although our proposed loss function follows the theoretically predicted behavior in practice and does achieve slope minimization, it is far from being perfect. Designing better loss functions can be an interesting avenue for future work.

4.3. Light Transport Simulation

We further demonstrate the effectiveness of our method on integration problems commonly encountered in light transport simulation. Light transport simulation considers an integration problem over all paths light could have travelled between a virtual camera and the different light sources in a scene. In practice, this integral has no

closed form solution and we instead use Monte Carlo integration to approximate it. We implement path tracing [Kaj86], a Monte Carlo method for light transport, that, for each pixel of an image, randomly samples light paths and averages their contribution. By recording the progressive pixel estimates over multiple sampled paths, we obtain a sequence of noisy estimates that can be improved by sequence transformation. We implement our method on top of the PBRT rendering system [PJH16].

We start with a simple scene that consists of a single plane illuminated by a rod-shaped area light source outside the view of the camera. The camera only sees the plane being illuminated by the light source. We use two different materials for the plane: a Lambertian diffuse material and a Disney BRDF with GGX material [WMLT07]. For the Lambertian material, the rendering integrand is expected to be similar to the 1D case of the step function. For the GGX material, the integrand is expected to be similar to the 1D case of the Gaussian multiplied by a step function. Wang et al. [WRG*09] use one lobe to represent the glossy BRDF. This spherical Gaussian lobe in 3D is a Gaussian distribution in 1D. We also represent the visibility term using a step function. Therefore, the multiplication of the Gaussian and step function is a good one-dimensional representation of a typical light transport simulation use case scenario when a GGX material is used.

For this scene, we train the network on a dataset of 40 runs of 32 images of resolution 128×128 . Our network takes 6 seconds per epoch during training, and we train for 1000 epochs. A batch

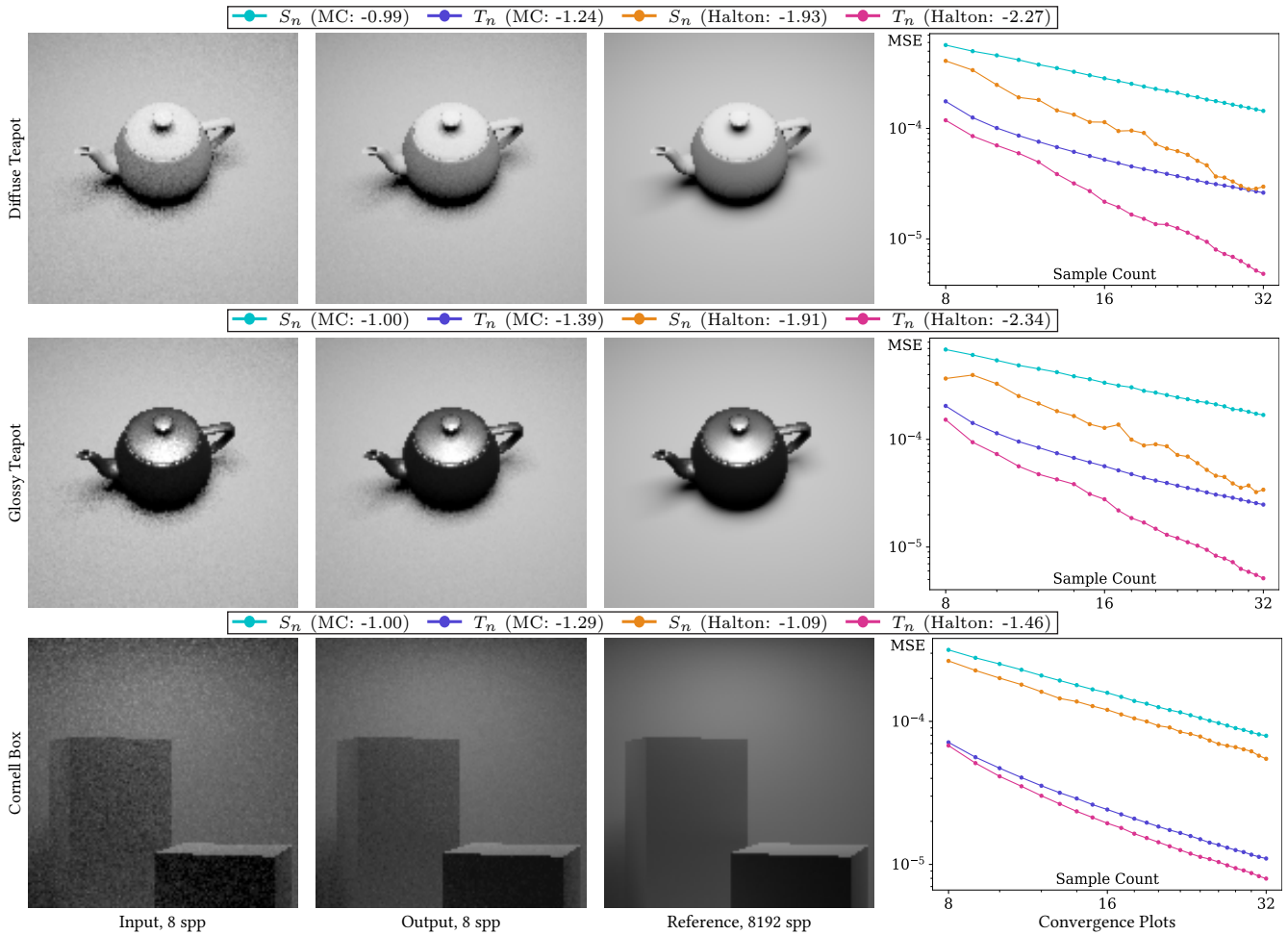


Figure 6: Results for scenes in light transport: A simple monochrome scene with a plane and a diffuse teapot, lit by an area light; the same scene but with a glossy teapot; and the cornell box scene, and with depth 8 as an example of more complex light transport. The two pairs of S_n and T_n in the convergence graphs correspond to using the random and Halton samplers respectively. In all cases, we observe decreased error at all sample counts and the behavior of the outputs match with the discussion in Section 3.2.

size of 256 is used, where each batch contains one integrand (in this case, a pixel) with all its runs. We mainly consider images rendered using the direct lighting integrator. Contrary to the 1D integration scenario, we train a network with four hidden layers (see Figure 2). We use different runs of the same scene for both training and testing: 40 runs for training, and a separate set of 10 runs for testing are used. The results are shown in Figure 5. As can be seen, the network can successfully predict pixel values closer to the reference value compared to the input sequence.

We then experiment with a scene that has more complex geometry. In this case, the rendering integrand is a product of a smooth material function and a step function due to light occlusions. The results are shown in Figure 6. The plot in Figure 6 shows that our method improves upon Monte Carlo integration. The qualitative results also show that our method works well across different sample counts. However, we found that our method provides more limited

improvement on some sequences generated by quasi-Monte Carlo integration using a Halton sampler compared to a random sampler. We suspect that the sequence has a faster convergence and lower error so the amount of optimization possible within our current framework is limited.

Relevance to Monte Carlo Image Denoising. Neural sequence transformation on light transport images can yield results effectively similar to Monte Carlo sampling and reconstruction in computer graphics, especially Monte Carlo image denoisers [ZIL*15]. A typical Monte Carlo image denoiser works by exploiting spatial coherence in a local window centered at a pixel to filter noise and predicts the pixel values in the window. Such filtering can be realized by multiple techniques such as reducing wavelet coefficients [ODR09], cross-bilateral filters [LWC12, SD12], first-order regression [MCY14, MIGYM15, BRM*16], Bayesian filtering [BB17], kernel-based denoisers [CKS*17, BVM*17, XZW*19],

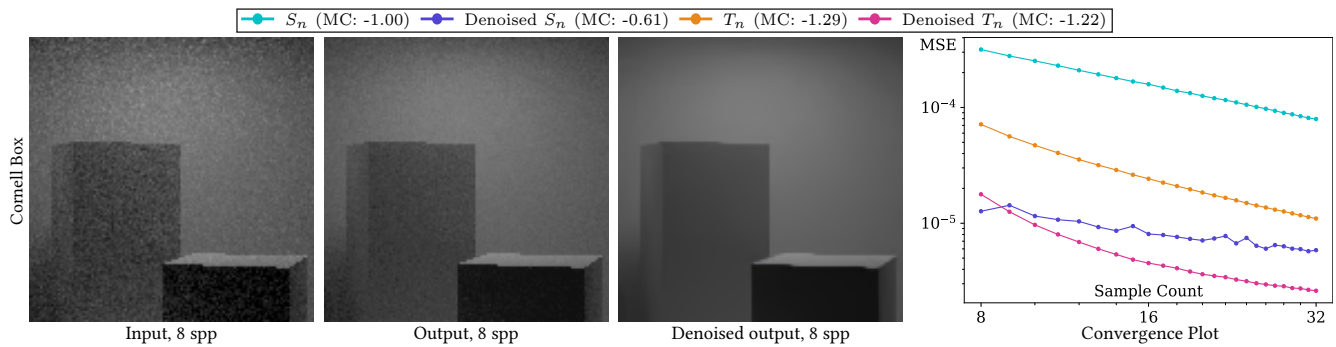


Figure 7: Results for the cornell box scene, and with depth 8. We show the convergence graph of the sequences after applying the OptiX denoiser [CKS*17, PBD*10] to both the input and output images. We observe that it is possible to apply denoising to the output images of our method and get improved results.

and sample-based denoisers [GLA*19, MH20]. Recent variance reduction [MRKN20] and deep combiner methods [BHHM20] can further reduce noise and other correlated visual artifacts.

Nevertheless, our method is not a denoising method, strictly speaking. We treat sample estimates for each pixel as an independent sequence, and apply our method to each individual sequence to obtain its corresponding pixel value estimate. Our method also differs from sample-based denoising [GLA*19, MH20] in that their samples are unordered and often include auxiliary features such as depth and normals. Our method can be considered as orthogonal to denoising. This is empirically shown in Figure 7, where we applied a denoiser in OptiX [CKS*17, PBD*10] to both the input and output sequences of our neural network. It can be seen that using a denoiser can further reduce the error of the transformed sequences. Further investigations on the connection of sequence transformation to sample-based filtering and reconstruction [HJW*08, BEJM15] and image denoising [ZJL*15] would be an interesting future work.

5. Conclusion and Future Work

In this work, we proposed a neural network architecture that attempts to learn sequence transformation in order to accelerate Monte Carlo integration. We designed a custom tailored loss function that is modeled after the convergence of Monte Carlo estimates. The proposed method was evaluated on simple 1D integrands including the step and the Gaussian functions, and on higher dimensional problems in light transport simulation. Noticeable improvements were achieved in both cases, demonstrating the effectiveness of learning a sequence transformation from data.

Our method suffers from a few limitations. First of all, similar to all other data-driven approaches, our neural sequence transformation performs poorly for integrands that are very different from the ones in the training data. While it is very challenging to generalize the network to all possible integrands, improving the robustness for unseen data is an interesting research direction.

Secondly, the current network has limited applicability for problems in light transport simulation. This limitation is due to the complexity of the integrands in light transport simulation, which are

typically multi-dimensional and discontinuous. Priors such as spatial coherence can also be explored to improve the convergence. This might be possible by employing convolutional layers in the neural network similar to image denoisers. On the other hand, advanced architectures such as RNNs with LSTM layers might also be able to increase the robustness of our method. However, using recurrent networks is not straightforward as computation of the loss requires transforming the entire sequence. For applications, extending the current implementation to handle RGB rendering is also a practical task.

Another interesting research direction would be a combination of analytical sequence transformation methods with data-driven methods. We showed that in expectation, the sequence of Monte Carlo estimates satisfies the definition of logarithmic convergence (see Appendix A). Hence, by using analytical sequence transformation techniques known to be effective for logarithmic convergence in combination with our method, improvement in its robustness might be possible.

Acknowledgments

The authors would like to thank Toshiya Hachisuka, Jamorn Sriwasansak, and Rex West for their valuable guidance, discussions and suggestions. They are also grateful to the reviewers whose thoughtful comments and feedback helped improve the exposition. This work is partially funded by the Japanese Government (Monbukagakusho - MEXT) Scholarship ID JP160005 and JSPS KAKENHI Grant Numbers JP19F19759, JP21K11910.

References

- [Ait27] AITKEN A. C.: Xxv.—on bernoulli’s numerical solution of algebraic equations. *Proceedings of the Royal Society of Edinburgh* 46 (1927), 289–305.
- [And65] ANDERSON D. G.: Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)* 12, 4 (1965), 547–560.
- [BB17] BOUGHIDA M., BOUBEKEUR T.: Bayesian collaborative denoising for monte carlo rendering. *Computer Graphics Forum* 36, 4 (2017), 137–153.

- [BEJM15] BAUSZAT P., EISEMANN M., JOHN S., MAGNOR M.: Sample-based manifold filtering for interactive global illumination and depth of field. *Comput. Graph. Forum* 34, 1 (2015), 265–276.
- [BHHM20] BACK J., HUA B.-S., HACHISUKA T., MOON B.: Deep combiner for independent and correlated pixel estimates. *ACM Trans. Graph.* 39, 6 (2020).
- [Bre71] BREZINSKI C.: Accélération de suites à convergence logarithmique. *CR Acad. Sci. Paris* 273 (1971), 727–730.
- [BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J. A., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. *Computer Graphics Forum* 35, 4 (2016), 107–117.
- [BRZ20] BREZINSKI C., REDIVO-ZAGLIA M.: *Extrapolation and Rational Approximation*. Springer, 2020.
- [BVM*17] BAKO S., VOGELS T., MCWILLIAMS B., MEYER M., NOVÁK J., HARVILL A., SEN P., DEROSE T., ROUSSELLE F.: Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics* 36, 4 (2017).
- [BZ91] BREZINSKI C., ZAGLIA M. R.: Construction of extrapolation processes. *Applied numerical mathematics* 8, 1 (1991), 11–23.
- [BZ13] BREZINSKI C., ZAGLIA M. R.: *Extrapolation methods: theory and practice*, vol. 2. Elsevier, 2013.
- [CKK18] CHRISTENSEN P., KENSLER A., KILPATRICK C.: Progressive multi-jittered sample sequences. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 21–33.
- [CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZHAI R., AILA T.: Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Transactions on Graphics* 36, 4 (2017).
- [DGB82] DELAHAYE J., GERMAIN-BONNE B.: The set of logarithmically convergent sequences cannot be accelerated. *SIAM Journal on Numerical Analysis* 19, 4 (1982), 840–844.
- [DH20] DORAN A. E., HIRATA S.: Convergence acceleration of monte carlo many-body perturbation methods by direct sampling. *The Journal of Chemical Physics* 153, 10 (2020), 104112.
- [ETF19] ELEK O., THOMAS M. M., FORBES A.: Learning Patterns in Sample Distributions for Monte Carlo Variance Reduction. *arXiv e-prints* (2019), arXiv:1906.00124.
- [Fal19] FALCON WA E. A.: Pytorch lightning. *GitHub. Note: <https://github.com/PyTorchLightning/pytorch-lightning>* 3 (2019).
- [GLA*19] GHARBI M., LI T.-M., AITALLA M., LEHTINEN J., DURAND F.: Sample-based monte carlo denoising using a kernel-splatting network. *ACM Trans. Graph.* 38, 4 (2019).
- [HJW*08] HACHISUKA T., JAROSZ W., WEISTROFFER R. P., DALE K., HUMPHREYS G., ZWICKER M., JENSEN H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph.* 27, 3 (2008), 1–10.
- [HSW89] HORN K., STINCHCOMBE M., WHITE H.: Multilayer feed-forward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.
- [JASF15] JACQUELIN E., ADHIKARI S., SINOUE J.-J., FRISWELL M. I.: Polynomial chaos expansion in structural dynamics: Accelerating the convergence of the first two statistical moment sequences. *Journal of Sound and Vibration* 356 (2015), 144–154.
- [Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), pp. 143–150.
- [KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [Kow81] KOWALEWSKI C.: Accélération de la convergence pour certaines suites à convergence logarithmique. In *Padé approximation and its applications, Amsterdam 1980 (Amsterdam, 1980)*, vol. 888 of *Lecture Notes in Math*. Springer, Berlin-New York, 1981, pp. 263–272.
- [Lav94] LAVASTRE H.: On the stochastic acceleration of sequences of random variables. *Applied numerical mathematics* 15, 1 (1994), 77–98.
- [Lev72] LEVIN D.: Development of non-linear transformations for improving convergence of sequences. *International Journal of Computer Mathematics* 3, 1-4 (1972), 371–388.
- [LWC12] LI T.-M., WU Y.-T., CHUANG Y.-Y.: SURE-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph.* 31, 6 (2012), 194:1–194:9.
- [MCY14] MOON B., CARR N., YOON S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph.* 33, 5 (2014).
- [MH20] MUNKBERG J., HASSELGREN J.: Neural denoising with layer embeddings. *Computer Graphics Forum* 39, 4 (2020), 1–12.
- [MIGYM15] MOON B., IGLESIAS-GUITIÁN J. A., YOON S.-E., MITCHELL K.: Adaptive rendering with linear predictions. *ACM Trans. Graph.* 34, 4 (2015), 121:1–121:11.
- [MRKN20] MÜLLER T., ROUSSELLE F., KELLER A., NOVÁK J.: Neural control variates. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–19.
- [MU49] METROPOLIS N., ULAM S.: The monte carlo method. *Journal of the American statistical association* 44, 247 (1949), 335–341.
- [O'B92] O'BRIEN D.: Accelerated quasi monte carlo integration of the radiative transfer equation. *Journal of Quantitative Spectroscopy and Radiative Transfer* 48, 1 (1992), 41–59.
- [ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHY R.: Adaptive wavelet rendering. *ACM Trans. Graph.* 28, 5 (2009), 140:1–140:12.
- [Osa96] OSADA N.: Vector sequence transformations for the acceleration of logarithmic convergence. In *Proceedings of the Sixth International Congress on Computational and Applied Mathematics (Leuven, 1994)* (1996), vol. 66, pp. 391–400.
- [PBD*10] PARKER S., BIGLER J., DIETRICH A., FRIEDRICH H., HOBEROCK J., LUEBKE D., MCALLISTER D., MCGUIRE M., MORLEY K., ROBISON A., STICH M.: OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics* 29, 4 (2010), 66:1–66:13.
- [PDZ*18] PENG Y., DENG B., ZHANG J., GENG F., QIN W., LIU L.: Anderson Acceleration for Geometry Optimization and Physics Simulation. *ACM Transactions on Graphics* 37, 4 (2018).
- [PGM*19] PASZKE A., GROSS S., MASSA F., LERER A., BRADBURY J., CHANAN G., KILLEEN T., LIN Z., GIMELSHEIN N., ANTIGA L., DESMAISON A., KOPF A., YANG E., DEVITO Z., RAISON M., TEJANI A., CHILAMKURTHY S., STEINER B., FANG L., BAI J., CHINTALA S.: Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* 32, Wallach H., Larochelle H., Beygelzimer A., d'Alché Buc F., Fox E., Garnett R., (Eds.). Curran Associates, Inc., 2019, pp. 8024–8035.
- [PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*, 3rd ed. 2016.
- [Rie90] RIEDEL R.: Zur Konvergenzbeschleunigung von Reihen mit logarithmischer Konvergenz. *Z. Anal. Anwendungen* 9, 4 (1990), 379–384.
- [Sab87] SABLONNIÈRE P.: Convergence acceleration of logarithmic fixed point sequences. *J. Comput. Appl. Math.* 19, 1 (1987), 55–60.
- [SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph.* 31, 3 (2012).
- [Sed90] SEDOGBO G. A.: Convergence acceleration of some logarithmic sequences. vol. 32. 1990, pp. 253–260. Extrapolation and rational approximation (Luminy, 1989).
- [SF79] SMITH D. A., FORD W. F.: Acceleration of linear and logarithmic convergence. *SIAM J. Numer. Anal.* 16, 2 (1979), 223–240.
- [Sid03] SIDI A.: *Practical extrapolation methods: Theory and applications*, vol. 10. Cambridge University Press, 2003.
- [Sid17] SIDI A.: *Vector extrapolation methods with applications*. SIAM, 2017.

- [Wen89] WENIGER E. J.: Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series. *Computer Physics Reports* 10, 5-6 (1989), 189–371.
- [WHKH13] WILLOW S. Y., HERMES M. R., KIM K. S., HIRATA S.: Convergence acceleration of parallel monte carlo second-order many-body perturbation calculations using redundant walkers. *Journal of chemical theory and computation* 9, 10 (2013), 4396–4402.
- [WMLT07] WALTER B., MARSCHNER S. R., LI H., TORRANCE K. E.: Microfacet models for refraction through rough surfaces. *Rendering techniques 2007* (2007), 18th.
- [WRG*09] WANG J., REN P., GONG M., SNYDER J., GUO B.: All-frequency rendering of dynamic, spatially-varying reflectance. In *ACM SIGGRAPH Asia 2009 papers*. 2009, pp. 1–10.
- [XZW*19] XU B., ZHANG J., WANG R., XU K., YANG Y.-L., LI C., TANG R.: Adversarial Monte Carlo Denoising with Conditioned Auxiliary Feature. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2019)* 38, 6 (2019), 224:1–224:12.
- [ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHY R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum* 34, 2 (2015), 667–681.

Appendix A: Variance Calculations

Our goal in this section is twofold: to prove that the convergence of the sequence of Monte Carlo estimates is closest to logarithmic in nature, and to prove that the $a_n g_n$ transformation is unable to decrease the mean squared error for Monte Carlo integration.

For the remainder of this section, let $f : [a, b] \rightarrow \mathbb{R}$ be a function with $\int_a^b f(x)^2 dx < \infty$, and $I = \int_a^b f(x) dx$. Let X_i be sampled independently and uniformly at random from the interval $[a, b]$. Denote

$$S_n = \frac{1}{n} \sum_{i=1}^n \frac{f(X_i)}{p(X_i)}.$$

Recall that (S_n) converges to I (in probability) as $n \rightarrow \infty$.

Convergence Type for Monte Carlo Integration

Proposition 1 For f , p and S_n defined as above,

$$\frac{\mathbb{E}(S_{n+1} - I)^2}{\mathbb{E}(S_n - I)^2} = \frac{n}{n+1} \xrightarrow{n \rightarrow \infty} 1, \text{ and}$$

$$\frac{\mathbb{E}(S_{n+1} - S_n)^2}{\mathbb{E}(S_n - S_{n-1})^2} = \frac{(n+1)(n-1)}{n^2} \xrightarrow{n \rightarrow \infty} 1.$$

Proof Let $Y_i := \frac{f(X_i)}{p(X_i)}$. Then observe that Y_i are all i.i.d random variables. As $f \in L^2[a, b]$, $\text{Var}(Y_i) < \infty$. Further, $S_n = \frac{1}{n} \sum_{i=1}^n Y_i$ has expectation I , implying $\mathbb{E}(S_n - I)^2 = \frac{1}{n} \text{Var}(Y_1)$, leading to the first equality. ■

For the second equality, we note that $\mathbb{E}(S_{n+1} - S_n) = 0$, implying $\mathbb{E}(S_{n+1} - S_n)^2 = \text{Var}(S_{n+1} - S_n)$. Moreover, by the independence of Y_{n+1} and S_n ,

$$\begin{aligned} \text{Var}(S_{n+1} - S_n) &= \text{Var}\left(\frac{1}{n+1}(Y_{n+1} - S_n)\right) \\ &= \frac{1}{(n+1)^2} (\text{Var}(Y_{n+1}) + \text{Var}(S_n)) \\ &= \frac{n+1}{n} \text{Var}(Y_1). \end{aligned}$$

The second equality follows as a direct consequence. □

Variance Calculation for the $a_n g_n$ Transformation

For this part, we let f , p , I , S_n and Y_i be defined the same way as in Proposition 1. Let g_n be any decreasing sequence. Define the transformed sequence (T_n) as:

$$T_{n+1} = S_n - \frac{S_{n+1} - S_n}{g_{n+1} - g_n} \cdot g_n. \quad (10)$$

Proposition 2 The Variance of T_{n+1} equals

$$\text{Var}(T_{n+1}) = \frac{\text{Var}(Y_1)}{n(n+1)^2} \left(\frac{((n+1)g_{n+1} - ng_n)^2 + ng_n^2}{(g_n - g_{n+1})^2} \right).$$

In particular,

$$\frac{\text{Var}(T_{n+1})}{\text{Var}(S_{n+1})} = 1 + \left(\frac{g_{n+1}}{g_{n+1} - g_n} \right)^2 \cdot \frac{1}{n},$$

which is at least 1, independent of the sequence $\{g_n\}$.

Proof Let us replace S_{n+1} with $\frac{1}{n+1}(Y_{n+1} + nS_n)$ in (10). Then, we can write

$$\begin{aligned} T_{n+1} &= S_n - \frac{Y_{n+1} - S_n}{g_{n+1} - g_n} \cdot g_n \\ &= \frac{S_n \cdot ((n+1)g_{n+1} - ng_n) - Y_{n+1}g_n}{(n+1)(g_{n+1} - g_n)}. \end{aligned}$$

Therefore, $\text{Var}(T_{n+1})$ can be computed by the independence of S_n and Y_{n+1} , and we obtain

$$\begin{aligned} \text{Var}(T_{n+1}) &= \frac{((n+1)g_{n+1} - ng_n)^2 \text{Var}(S_n) + g_n^2 \text{Var}(Y_{n+1})}{(n+1)^2 (g_{n+1} - g_n)^2} \\ &= \frac{\text{Var}(Y_1)}{n(n+1)^2} \left(\frac{((n+1)g_{n+1} - ng_n)^2 + ng_n^2}{(g_n - g_{n+1})^2} \right). \end{aligned}$$

Now we compare $\text{Var}(T_{n+1})$ and $\text{Var}(S_{n+1})$. Note that $\text{Var}(S_{n+1}) = \frac{\text{Var}(Y_1)}{n+1}$, and hence,

$$\begin{aligned} \frac{\text{Var}(T_{n+1})}{\text{Var}(S_{n+1})} &= \frac{((n+1)g_{n+1} - ng_n)^2 + ng_n^2}{n(n+1)(g_n - g_{n+1})^2} \\ &= 1 + \frac{g_{n+1}^2}{n(g_n - g_{n+1})^2}, \end{aligned}$$

as desired.

As the above identity holds regardless of the choice of g_n , we obtain $\text{Var}(T_{n+1}) \geq \text{Var}(S_{n+1})$. In other words, T_{n+1} is not able to shrink the error in the Monte Carlo estimate S_{n+1} . However, for certain choices of g_n , it can be ensured that $\frac{\text{Var}(T_{n+1})}{\text{Var}(S_{n+1})}$ approaches 1 as $n \rightarrow \infty$. □