Efficient Image-space Shape Splatting for Monte Carlo Rendering

by

Xiaochun Tong

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, 2024

© Xiaochun Tong 2024

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

Photorealistic image synthesis through physically based rendering is essential to achieve high visual fidelity. Monte Carlo rendering provides a scalable solution to accurately simulating the complex interactions between light and geometries within a scene. Monte Carlo integration numerically integrates a function by taking multiple pointwise estimations of the function and computing the average. Therefore, a typical Monte Carlo rendering algorithm independently samples one light path or path tree at a time for each pixel in the image. One approach to significantly reduce computational costs and enhance efficiency is to reuse light paths across multiple pixels and therefore amortize sample generation. The current state of the art of path reusing methods employs a technique known as shift mapping to deterministically map light paths from one pixel to another at low cost, yet the total computation cost is still linear w.r.t to the number of pixels processed in shift mapping. We proposed a general framework, that generalizes reusing light paths to multiple pixels arranged in arbitrary two-dimensional shapes as *image space shape splatting*. Our shape is defined as a set of multiple pixels, and the framework allows us to sample such shapes more efficiently than by evaluating each pixel individually through shift mapping. Our key insight is to design a fast biased estimator that sparsely evaluates the contribution of shifted paths at random pixels within the shape and interpolates the contribution to the other pixels. We then employ a debiasing estimator to eliminate the bias from approximation. Our method can be seamlessly integrated with many state-of-the-art rendering methods and improving their efficiency over traditional pointwise sampling.

Acknowledgements

I would like to thank my supervisor Toshiya Hachisuka for making me part of his group and his support throughout this research project. I would also like to thank Weijie Zhou for his technical support in implementing ReSTIR; Wenyou Wang, Shaokun Zheng, Jiawei Huang, and Maxwell Fang for their feedback on the draft of the submitted paper that this thesis is based on. Lastly, I would like to thank my parents for their love and support that make this thesis possible.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

List of Figures viii

1 Introduction 1

1.1 Contribution . . . 2

1.2 Organization . . . 2

2 Background 4

2.1 Monte Carlo Integration . . . 4

2.1.1 Random Variable and Probability Distribution . . . 4

2.1.2 Expected Value . . . 5

2.1.3 Monte Carlo Estimator . . . 6

2.1.4 Variance and Correlation . . . 7

2.1.5 Unbiasedness and Consistency . . . 8

2.1.6 Debiasing . . . 8

2.1.7 Importance Sampling . . . 9

2.1.8 Multiple Importance Sampling . . . 10

2.2 Physically Based Rendering . . . 11
2.2.1 Light Transport Equation . . . 11
2.3 Path Integral Formulation . . . 12
2.3.1 Monte Carlo Estimators for LTE . . . 14
2.3.2 Markov Chain Monte Carlo Rendering . . . 14

3 Shape Sampling and Path Reusing **19**
3.1 Line segment sampling . . . 19
3.2 High dimensional samples in volume rendering . . . 19
3.3 Path reusing . . . 20
3.3.1 Shift Mapping . . . 20
3.3.2 Bekaert style-path reusing . . . 25
3.3.3 ReSTIR . . . 25
3.3.4 Gradient-domain rendering . . . 27

4 Image-space Shape Splatting **29**
4.1 A Simplified Example . . . 30
4.2 Shape Splatting . . . 32
4.2.1 Amortized Shape Splatting via Mutations . . . 34
4.2.2 MCMC-based Weighting for Shape Splatting . . . 36
4.2.3 Efficient Shape Estimator via Debiasing . . . 37
4.2.4 Generalization to 2D Shapes . . . 39
4.2.5 Spatially Varying Shape . . . 39
4.2.6 Decorrelating Shape Splats . . . 40
4.2.7 Optional Mixing with Point Splats . . . 41
4.2.8 Alternative Weighting via Pairwise MIS . . . 41

5 Results **44**

5.1 Implementation 44

5.2 Numerical Results 45

5.3 Ablation Study 45

6 Conclusion **54**

References **55**

List of Figures

2.1 The three-point form of LTE, the essential building block of path integral formulation. 13

3.1 (A) A typical setup of shift mapping involving specular and diffuse vertices. (B) Replaying random numbers handles specular vertices well but is prone to produce very different paths due to lack of path space information. (C) Reconnection guarantees the shifted path shares a common suffix with the base path but is likely to produce invalid specular interactions. 22

3.2 Hybrid shift mapping postpones reconnection until a diffuse vertex is encountered, handling both type of vertices well while sharing most base path as possible. 23

3.3 Bekaert style-path reusing produces visible blocky artifact due to its tile-wise path reuse strategy. 26

4.1 We compare sampling a 2D distribution using points and shapes under the *equal point-sample counts*. The shape is a rotated quad with 61 points (pixels). Our method sample each shape with the cost of 4 point samples on average. Shift mapping in this example has the same cost as generating a new sample, so any gain is not coming from sample reuse. Shape samples fill in more pixels at the same cost, making it far more efficient than point sampling. *Relative Mean Square Error (relMSE)* is defined as $relmse(I, I_{gt}) = \frac{(I-I_{gt})^2}{I_{gt}^2+0.01}$. *Mean Absolute Percentage Error (MAPE)* is defined as $mape(I, I_{gt}) = \frac{|I-I_{gt}|}{|I_{gt}|+0.01}$ 31

4.2 We insert strides and randomly rotating the shape to greatly reduce low frequency artifacts and correlation between neighbor pixels. The decorrelated image can be further denoised by a machine learning denoiser without introducing outstanding artifacts. 40

5.1 We demonstrate the performance of different components of our method. Note while **E** has lower error, it is inconsistent and has blurring artifact. . 46

5.2 Our debiasing estimator outperforms naive shape splatting, mixing with point samples improves efficiency further by removing outliers. 46

5.3 Error plots of shape splatting vs point splatting baselines on GPU. 47

5.4 Error plots of shape vs point splatting on top of PSSMLT 47

5.5 Top: On a challenging scene, our method consistently out performs its point splatting baseline. Bottom: Our method suppresses color noise when integrated with ReSTIR. 48

5.6 We integrated our method on top of PT and ReSTIR PT on GPU. Our method consistently outperforms point splatting in a variety of scenes. . . 49

5.7 We integrated our method on top of PSSMLT on CPU. Our method consistently outperforms point splatting in a variety of scenes. 50

5.8 Visualizing the weights when mixing with point samples. Blue corresponds to a higher weight for shape splatting than points. Our method works best when the geometry and lighting is smooth, but it can also handle non-smooth shading such as image textures. 51

5.9 Compared different PMF for debiasing. While $p_k = k^{-1}$ has better equal sample (left) performance, under equal time (right) its larger sampling cost makes it less competitive than our choice of $p_k = k^{-2}$. 52

5.10 We compared our debaising estimator with naive shape splatting under equal time. In MCMC Rendering (right), our debiasing estimator outperforms naive shape splatting by a large margin. In MC rendering with path tracer, while debiasing without mixing with point samples produces higher relMSE due to the present of outliers, mixing with point samples effectively suppresses outliers and has lower variance than naive shape splatting. 52

Chapter 1

Introduction

Physically based rendering (PBR) has received wide adaption in movie production as well as video games to generate virtually indistinguishable photorealistic images. The core of Physically Based Rendering (PBR) lies in the accurate modeling of how light propagates, scatters on surfaces, and interacts with various media. This is achieved using a wide class of powerful Monte Carlo rendering algorithms, which are based on the path space formulation of the light transport equation [58, 30]

The flexibility of Monte Carlo integration stems from its only requirement that the integrand can be evaluated at any point, making it possible to seamlessly integrate complex material models into the rendering pipeline, whereas traditional rasterization approaches use adhoc tricks to mimic the appearance of various materials. Nevertheless, Monte Carlo methods are known for their noisy result and slow convergence. Reducing variance, hence noise, in Monte Carlo rendering is thus crucial for rendering increasingly complex scenes. For problems where part of the integration is analytically available, it is sometimes possible to evaluate the integrand by a "line" or "plane", effectively utilizing non-point samples. Several works in stochastic rasterization [1] and volumetric density estimation [28] have shown that sampling higher dimensional primitives or shapes (such as lines and planes) can be beneficial despite their higher sampling costs as one shape is equivalent to many (or even infinite) point samples. However, a key question remains - is it possible to use non-point samples in general problems? Current shape sampling methods require analytic solution available for a part of the integral, which limits its application. Applying these methods to general Monte Carlo rendering with complex material and light transport is challenging since such analytic integration over path space is difficult in many cases.

While it is still unclear whether sampling continuous shapes at practical cost is possible

for general problems, we attempt to tackle a slightly simplified variant: to sample pixels arranged in arbitrary 2D shapes in image space, assuming only point evaluation of the integrand, by sharing light path contribution across pixels. Albeit sharing similarity to path reusing methods, existing methods requires a linear computation cost w.r.t the number of pixels in each shape. Our framework instead fundamentally reduces the computational cost of sampling a shape to *sublinear* to the number of points in the shape, making shape samples an effective replacement of point samples.

We named our method "image-space shape splatting", where we "sample" shape samples and accumulate those shape samples to the film, since the latter is usually referred to as "splatting" in rendering.

1.1 Contribution

We summarize the two main contributions of our work as:

1. A formulation of image-space shape splatting as MCMC mutations that allows efficient evaluation of shape contribution as well as flexible shape configuration
2. A novel debiasing estimator that enables a sublinear sampling cost w.r.t the number of pixels within each shape.

1.2 Organization

Chapter. 2 provides the necessary background knowledge about Monte Carlo rendering, including a review on probability theory and Monte Carlo integration, and a brief introduction to light transport equation and its path space formulation. We also introduced the necessary techniques that this work is based on, such as Markov Chain Monte Carlo rendering.

Chapter. 3 discusses prior work related to the thesis and the shift mapping technique which most path reusing methods are based on. These works can be loosely classified in two categories: various application of shape sampling in computer graphics as well as recent advances in path reusing. Prior work in shape sampling requires analytical integration of, at least part of, the integrand and is not applicable to the problem we aim to tackle in the thesis; existing path reusing methods, while being orthogonal to ours, have fundamentally higher cost.

Chapter. 4 presents the key contributions of the thesis: the MCMC-weighted shape splatting formulation and the efficient debiasing estimator. We first discussed our method on a simplified 1D case and later extend it to 2D. We also discussed various (optional) techniques, such as decorrelating shape splats and constructing spatially varying shapes, that are beneficial for a practical implementation.

Chapter. 5 provides numerical experiments that validate our claims. We implemented our method on top of many existing point-splatting rendering methods and observed consistent improvement. We also provide a comprehensive ablation study that demonstrates the effectiveness of various components of our method.

Part of materials in the thesis is a reproduction of:

Xiaochun Tong and Toshiya Hachisuka, "Efficient Image-space Shape Splatting for Monte Carlo Rendering", Submitted, 2024

Chapter 2

Background

2.1 Monte Carlo Integration

Monte Carlo integration is a widely used method to numerically compute a high dimensional integral. Deterministic numerical integration methods such as quadratures have their computational cost grows exponentially as dimensionality increases. The key idea of Monte Carlo methods is to solve the integration problem in a probabilistic view, by replacing integrals with expectation and leveraging the properties of random variables to achieve a convergence rate independent of dimension.

2.1.1 Random Variable and Probability Distribution

A random variable $X \in \Omega$ is a value determined by a random process. The particular value we observed from X is called a *realization* of X. A random variable is associated with a distribution $p(X)$ that describes the probability of a particular realization of X. When X can only take discrete values, we refer to X as discrete random variable and $p(X)$ as a *Probability Mass Function (PMF)*, where $p(X = x) \geq 0$ denotes the *frequency* that X is observed to be x. The sum of PMF over all possible events is one $\sum_{i=1}^{N} p(X = x_i) = 1$. For 1D variables, The *Cumulative Distribution Function* is the sum of probability that X is less or equal to some value:

$$P(x) = Pr\{X \leq x\} \tag{2.1}$$

When X is defined on a continuous space, it is a continuous random variable and $p(X)$ is a *Probability Density Function (PDF)*, which is defined using the CDF of X:

$$p(x) = \frac{dP(x)}{dx}, \tag{2.2}$$

in the 1D case. The PDF is a nonnegative measure defined on the domain of the random variable and have a total measure of 1. That is we have $\int_\Omega p(x)dx = 1$ and $p(x) \geq 0, \forall x \in \Omega$.

Let's consider two random variable $X \in \Omega_1, Y \in \Omega_2$. The tuple $(X, Y) \in \Omega_1 \times \Omega_2$ is also a random variable. The probability of the tuple $p(X, Y)$ is called a *joint probability* of X, Y. If X and Y are *independent variables*, we have $p(X, Y) = p(X)p(Y)$ for all X, Y. If X, Y are instead dependent variables, we have $p(X, Y) = p(X|Y)p(Y) = p(X)p(Y|X)$, where $p(X|Y)$ is a *conditional distribution* of X given Y. $p(X)$ and $p(Y)$ are called *marginal distributions*. If either is unknown beforehand, they can be computed as $p(X) = \int_\Omega p(X|y)p(y)dy$.

Given a random variable X, we can produce another random variable Y by applying an transformation T on top of X. That is $Y = T(X)$. If T is differentiable *almost everywhere* and one-to-one, we have:

$$p_Y(Y) = \frac{p_X(X)}{|\partial_x T(X)|}, \tag{2.3}$$

where $|\partial_x T(x)|$ is the determinant of the Jacobian matrix of T at X. For the general case of T is not one-to-one, it is very difficult to write down a simple formula for $p_Y(Y)$ and beyond the scope of this thesis. Eq. (2.3) is also known as the *change of variable* formula for distributions.

Under certain context, we may refer to a PDF as *unnormalized* if it is proportional to some true PDF but itself does not necessarily integrate to 1.

$$p(x) = \frac{p_{unnormalized}(z)}{Z}, Z = \int_\Omega p_{unnormalized}(x)dx, \tag{2.4}$$

where Z is the *normalization factor* of $p_{unnormalized}$ and $p(\cdot)$ is the true PDF of x.

2.1.2 Expected Value

The expected value of X is an average of all possible values of X weighted by $p(X)$. We have $\mathbf{E}[X] = \sum_{i=1}^N x_i p(x_i)$ or $\mathbf{E}[X] = \int_\Omega xp(x)dx$ for discrete and continuous X respectively. We

can also define expected value of a function over a distribution $p(\cdot)$ as:

$$\mathbf{E}_p[f(x)] = \int_\Omega f(x)p(x)dx \tag{2.5}$$

The expected value is a linear operator, meaning that:

1. $\mathbf{E}[a] = a$ for any constant a
2. $a\mathbf{E}[X] = \mathbf{E}[aX]$
3. $\mathbf{E}[X] + \mathbf{E}[Y] = \mathbf{E}[X + Y]$
4. More generally, $T(\mathbf{E}[X]) = \mathbf{E}[T(X)]$ for any linear operator T.

Additionally, for independent variables X, Y we also have $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$.

2.1.3 Monte Carlo Estimator

Consider a function $f(\cdot)$ defined over Ω. We can estimate its integral $I = \int_\Omega f(x)dx$ by viewing it as an expected value under some distribution $p(\cdot) : \Omega \to \mathcal{R}^{\geq 0}$.

$$\begin{aligned} \mathbf{E}_p\left[\frac{f(X)}{p(X)}\right] &= \int_\Omega \frac{f(x)}{p(x)} p(x)dx \\ &= \int_\Omega f(x)dx \\ &= I \end{aligned} \tag{2.6}$$

This representation allows us to use the Monte Carlo method to estimate I by sampling from the distribution $p(x)$. If we draw N independent samples $\{x_i\}_{i=1}^N$ from $p(\cdot)$, a Monte Carlo estimator for the integral I is given by:

$$\langle I_N \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{2.7}$$

According to the law of large numbers, as $N \to \infty$, $\langle I_N \rangle$ converges to I almost surely.

2.1.4 Variance and Correlation

Now we look into deriving the rate of convergence of Monte Carlo estimators. The expected square error between a Monte Carlo estimator and its expected value is called the *variance*:

$$\mathbf{V}[\langle I_N \rangle] = \mathbf{E}[(I - \mathbf{E}[\langle I_N \rangle])^2] \tag{2.8}$$

Two important property of variance is that for *independent* variables X, Y:

1. $\mathbf{V}[X] + \mathbf{V}[Y] = \mathbf{V}[X + Y]$
2. $\mathbf{V}[aX] = a^2\mathbf{V}[X]$

Therefore we can express the variance of an N-sample Monte Carlo estimator using that of a one-sample Monte Carlo estimator:

$$\mathbf{V}[\langle I_N \rangle] = \mathbf{V}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right] \tag{2.9}$$

$$= \frac{1}{N^2} N \mathbf{V}\left[\frac{f(X)}{p(X)}\right] \tag{2.10}$$

$$= \frac{1}{N}\mathbf{V}[\langle I_1 \rangle], \tag{2.11}$$

which suggests the variance of Monte Carlo estimator decreases at a rate of $\mathcal{O}\left(\frac{1}{N}\right)$, where the error is of rate $\mathcal{O}\left(\frac{1}{N^{0.5}}\right)$.

Eq. (2.1.4) only holds when two variables are independent. If X, Y are dependent, $\mathbf{V}[X + Y] \neq \mathbf{V}[X] + \mathbf{V}[Y]$ in general. In particular, it is equal to:

$$\mathbf{V}[X + Y] = \mathbf{V}[X] + \mathbf{V}[Y] + cov(X, Y), \tag{2.12}$$

where $cov(X, Y)$ is the *covariance* of X, Y, measuring the *correlation* between the two variables. The covariance is defined as:

$$cov(X, Y) = \mathbf{E}[(X - \mathbf{E}[X])(Y - \mathbf{E}[Y])] \tag{2.13}$$

$$= \mathbf{E}[XY] - \mathbf{E}[X]\mathbf{E}[Y] \tag{2.14}$$

We can take advantage of correlation to accelerate convergence when estimating sum or difference of random variables. For example, sampling two negatively correlated variables would reduce the variance as in Eq. (2.12), whereas positively correlated samples would facilitate variance reduction in estimating the difference of expected values.

2.1.5 Unbiasedness and Consistency

Two important yet distinct properties of a Monte Carlo estimator are unbiasedness and consistency. If $\mathbf{E}[\langle I_{MC}\rangle] = I$, the estimator $\langle I_{MC}\rangle$ is considered *unbiased*. Conversely, if this equality does not hold, the estimator is *biased*, with the bias defined as $\mathbf{B}[\langle I_{MC}\rangle] = I - \mathbf{E}[\langle I_{MC}\rangle]$. A key characteristic of an unbiased estimator is that averaging multiple instances of it guarantees convergence to the correct solution as the number of instances increases. In contrast, a biased estimator would converge to an incorrect solution.

An estimator is *consistent* if it accepts a user-defined parameter N and satisfies $\lim_{N\to\infty}\langle I_N\rangle = I$. For instance, if N represents the number of samples taken, the simple Monte Carlo estimator in Eq. (2.6) is clearly consistent. However, it is crucial to note that unbiasedness does not imply consistency and vice versa. For example, taking only one sample in Eq. (2.6) results in an unbiased but inconsistent estimator, as it is highly unlikely that a single sample would yield the correct solution.

2.1.6 Debiasing

Ideally, we aim for our estimators to be unbiased, as unbiased estimators do not introduce systematic errors beyond variance. However, in certain situations, deriving an unbiased estimator directly is impossible, or unbiased estimators may incur unacceptable computational costs, forcing us to resort to a biased one. Fortunately, if the biased estimator is consistent, we can still achieve unbiased estimation through a technique called *debiasing* [43].

Given a biased yet consistent estimator $\langle I_N\rangle$, we can unbiasedly estimate its bias by evaluating an infinite telescoping series:

$$\begin{aligned} I &= \lim_{N\to\infty}\langle I_N\rangle \\ &= \langle I_0\rangle + \mathbf{B}[\langle I_0\rangle] \\ &= \langle I_0\rangle + \lim_{N\to\infty}\langle I_N\rangle - \langle I_0\rangle \\ &= \langle I_0\rangle + \sum_{N=0}^{\infty}\Big(\langle I_{N+1}\rangle - \langle I_N\rangle\Big) \\ &\approx \langle I_{tele}\rangle = \langle I_0\rangle + \frac{\langle I_{k+1} - I_k\rangle}{p_k}, \end{aligned} \tag{2.15}$$

where k is an random integer sampled with probability p_k and evaluate one term of the imfinite series at a time. The Note that we estimate the difference $\langle I_{k+1} - I_k\rangle$ as a whole

to take advantage of positive correlation on two estimators to reduce variance. Eq. (2.15) is *almost surely* guaranteed to converge to the correct solution given large enough samples by the law of large numbers. However, for certain problems or a bad choice of p_k, $\langle I_{tele} \rangle$ might not have finite variance or even requires an infinite amount of time to evaluate.

2.1.7 Importance Sampling

We learned from Sec. 2.1.4 that the main source of error of Monte Carlo integration comes from its variance. Thus we can accelerate its convergence by reducing the variance of the estimator. By examining Eq. (2.9), we can observe that the variance depends on both the integrand $f(\cdot)$ and the sampling distribution $p(\cdot)$. While we have no control over $f(\cdot)$, we are free to choose any sampling distribution $p(\cdot)$ as long as $p(x) > 0$ whenever $f(x) > 0$. By carefully choosing a sampling distribution, we can significantly reduce the variance of Monte Carlo estimator. We refer to such sampling distribution as an *importance sampling distribution*.

The effect of importance sampling is most easily seen on this idealized example. Suppose we use a distribution that is proportional to a non-negative $f(\cdot)$, that is $p(x) \propto f(x)$. Normalizing it producing $p(x) = f(x)/I$. If we plug it in Eq. (2.6) we obtain:

$$\langle I_{IS} \rangle = \frac{f(x)}{f(x)/I} = I \tag{2.16}$$

$$\mathbf{V}[\langle I_{IS} \rangle] = 0, \tag{2.17}$$

resulting in a perfect zero-variance estimator. However, such perfect estimator is usually not possible in practice. While we can indeed sample from such $p(\cdot)$ directly (see Sec. 2.3.2), computing its normalizing constant is the same as solving the integral in the first place. In a more realistic case, we can utilize the insight learned from Eq. (2.16) to design a normalized importance sampling distribution where sampling is relatively easy. We refer to such distribution as *tractable* distribution. We are able to reduce the variance significantly if our chosen $p(\cdot)$ has similar shape to $f(\cdot)$.

However, it is also important to note that important sampling does not guarantee a reduction on variance. A poorly chosen distribution could increase the variance or even produce infinite variance. In general, for importance sampling to not increase variance, $p(\cdot)$ must approach 0 slower than $f(\cdot)$. To prevent such failure of importance sampling, we sometimes use a mixture of the importance sampling distribution and a uniform sampling so that the mixture distribution never approaches to zero faster than $f(\cdot)$. This technique is called *defensive sampling* and we will discuss the generalized version of it in the next section.

2.1.8 Multiple Importance Sampling

Usually $f(\cdot)$ is highly complex where it is difficult to find a single $p(\cdot)$ that fits $f(\cdot)$ well everywhere. However, it is usually easier to find multiple sample distributions that match $f(\cdot)$ well *locally*. For example, in rendering we usually find $f(\cdot)$ to be a form of $f(x) = g(x)h(x)$, where we have importance sampling distributions for $g(\cdot)$ and $h(\cdot)$ respectively, but not their product. By drawing samples from both distributions, we now have multiple estimators of the same integral but with different variance. Let's denote the estimator that importance samples $g(x)$ as $\langle I_g \rangle$ and the other as $\langle I_h \rangle$. A naive combination of both estimators generally does not reduce the variance,

$$\langle I_{naive} \rangle = \frac{\langle I_g \rangle + \langle I_h \rangle}{2} \tag{2.18}$$

$$\mathbf{V}[\langle I_{naive} \rangle] = \frac{\mathbf{V}[\langle I_g \rangle] + \mathbf{V}[\langle I_h \rangle]}{2} \tag{2.19}$$

as $\langle I_g \rangle$ would have high variance in regions it matches $f(\cdot)$ poorly and spill that variance into the combined estimator. It can be shown that if only constant weights are allowed, the estimator that minimizes the variance weighs $\langle I_g \rangle$ and $\langle I_h \rangle$ according to their inverse variance.

$$\langle I_{opt} \rangle = \frac{\langle I_g \rangle / \mathbf{V}[\langle I_g \rangle] + \langle I_h \rangle / \mathbf{V}[\langle I_h \rangle]}{\mathbf{V}[\langle I_g \rangle] + \mathbf{V}[\langle I_h \rangle]} \tag{2.20}$$

However, in practice computing the variance of each estimator is not feasible.

A technique called *Multiple Importance Sampling(MIS)* [58] allows us to simultaneously draw samples from multiple importance sampling distributions and weight each sample individually to reduce variance. The general form of an MIS estimator can be described as:

$$\langle I_{MIS} \rangle = \sum_{t=1}^{T} \sum_{i=1}^{n_t} w_t(x) \frac{f(x_{t,i})}{n_t p_t(x_{t,i})}, \tag{2.21}$$

where $\{p_t(\cdot)\}$ is a set of importance sampling distributions or *techniques*, $w_t(\cdot)$ is the MIS weight for each technique and n_t is the number of samples drawn for t-th technique. $\{x_{t,i}\}$ is a set of *independent* samples drawn from those techniques. We are allowed to choose $w_t(\cdot)$ is almost arbitrarily. As long as the following conditions are satisfied, we would have an unbiased estimator of I:

1. $\sum_{t=1}^{T} w_t(x) = 1$ whenever $f(x) > 0$

2. $w_t(x) = 0$ if $p_t(x) = 0$

The most commonly used weighting scheme is the *balanced heuristic* which is shown [58] that no other weighting scheme has significantly lower variance:

$$w_t(x) = \frac{n_t p_t(x)}{\sum_{t'=1}^{T} n_{t'} p_{t'}(x)} \tag{2.22}$$

$$\langle I_{BALANCED} \rangle = \sum_{t=1}^{T} \sum_{i=1}^{n_t} \frac{f(x_{t,i})}{\sum_{t'=1}^{T} n_{t'} p_{t'}(x)} \tag{2.23}$$

In essence, the balanced heuristic computes the actual PDF of a sample by considering all possible ways a sample could be generated. In practice, the balanced heuristics effectively shuts off the sampling strategy that is performing poorly on the current sample and leaving high weights to only the excelling strategies.

2.2 Physically Based Rendering

2.2.1 Light Transport Equation

In the context of photorealistic image synthesis, we aim to numerically solve a high dimensional integration equation known as the *rendering equation* or *light transport equation* (LTE) [30]. LTE describes the physics of light being transported and scattered across the scene through various interaction with surface materials such as reflection and refraction. LTE is built upon the assumption of geometry optics and linearity of scattering, both of which are sufficient for most application in computer graphics. The equation can be formulated as:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathcal{S}^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\omega_i \cdot n| d\omega_i, \tag{2.24}$$

where p is a point on surface, n is the geometric normal at p. $L_i(p, \omega_o)$ represents the radiance leaving point p in direction ω_o. The term $L_e(p, \omega_o)$ denotes the emitted radiance from point p in direction ω_o, in cases where the point p is a light source or on a emissive surface. The integral over the unit sphere $\mathcal{S}^2$ accounts for the incoming radiance $L_i(p, \omega_i)$ from direction ω_i, multiplied by the bidirectional scattering distribution function (BSDF) $f(p, \omega_o, \omega_i)$, which characterizes the behavior of light being scattered by the surface at

point p. To render an image, we would solve the LTE for each visible point at each pixel, recording the outward radiance L_o as the pixel colors.

Since the quantity of interest L appears in both sides of the Eq. (2.24), LTE can be seen as a Volterra integral equation of second kind. Solving such equation usually involves expanding the recursive terms into an infinite sum of increasingly longer series, known as the Neumann series. The Neumann series formulation of Eq. (2.24) leads to an alternative formulation of LTE, which we refer to as the *path integral formulation*.

2.3 Path Integral Formulation

In path integral formulation, instead of recursively integrating over unit sphere, we integrate over all possible light paths. First we need to convert Eq. (2.24) into *surface form*, removing the direction variable ω_i by performing a change of variable to integrate over all possible point p'' that are visible to p. We similar remove the dependency on ω_o by expressing it with the previous visible surface point p' along the path.

$$L_o(p \to p') = L_e(p \to p') + \int_A f(p'' \to p \to p', p'' \to p') L_i(p'' \to p') G(p'' \leftrightarrow p) dp'', \quad (2.25)$$

$$G(p'' \leftrightarrow p) = V(p'' \leftrightarrow p) \frac{|\omega_i n| \cdot |\omega_i n''|}{||p - p''||^2}, \omega_i = \frac{p'' - p}{||p - p''||} \quad (2.26)$$

where $G(p'' \leftrightarrow p')$ is the *geometry term* that captures the attenuation of light due to Lambert's law and inverse-squared distance falloff. $V(p'' \leftrightarrow p) = 1$ if p'' is visible to p and 0 otherwise. Eq. (2.25) allows us to express the rendering equation in terms of only points in 3D space. Starting from a point representing the camera, we recursively expanding Eq. (2.25) until we hit a light source. Each ordered set of vertices connecting the camera to a light source is a complete light path.

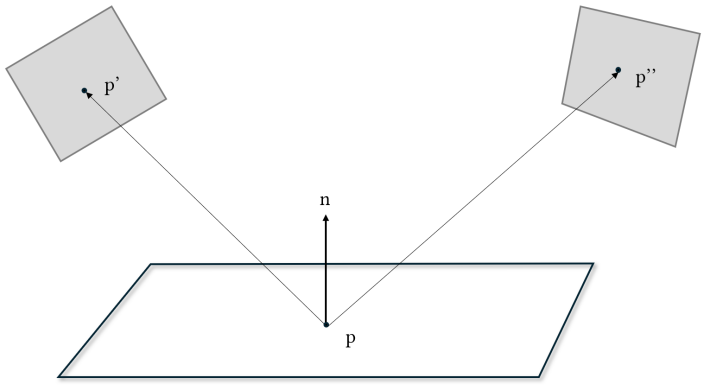


Figure 2.1: The three-point form of LTE, the essential building block of path integral formulation.

$$
\begin{aligned}
L_p(p_1 \to p_0) &= L_e(p_1 \to p_0) \\
&+ \int_A L_e(p_2 \to p_1) f(p_2 \to p_1 \to p_0) G(p_2 \Leftrightarrow p_1) dA(p_2) \\
&+ \int_A \int_A L_e(p_3 \to p_2) f(p_3 \to p_2 \to p_1) G(p_3 \Leftrightarrow p_2) \times \\
&f(p_2 \to p_1 \to p_0) G(p_2 \Leftrightarrow p_1) dA(p_3) dA(p_2) \\
&+ ... \\
&= L_e(p_1 \to p_0) \\
&+ \sum_{n=1}^{\infty} \int_{\mathcal{P}_n} L_e(p_{n+1} \to p_n) \prod_{i=0}^{n} f(p_{n+1} \to p_n \to p_{n-1}) G(p_{n+1} \Leftrightarrow p_n) d\mu_n(\overline{p}_n)
\end{aligned}
\tag{2.27}
$$

where A is the surface of all geometries in the scene, $\overline{p}_n = \{p_1, p_2, ...p_{n+1}\}$ as a complete light path of depth n, $\mathcal{P}_n = \prod_{i=1}^{n} A$ is the path space of depth n. The path space $\mathcal{P} = \bigcup_{n=1}^{\infty} \mathcal{P}_n$ contains all complete light paths.

2.3.1 Monte Carlo Estimators for LTE

The intensity of a pixel j is defined by restricting the path contributions to the area of the pixel using a pixel filter h_j.

$$I_j = \int_{\mathcal{P}} h_j(\mathbf{x}) L(\mathbf{x}) d\mathbf{x}, \tag{2.28}$$

where we use $L(\cdot)$ to represent the path contribution. To solve Eq. (2.28) via Monte Carlo, we sample $\mathcal{P}$ by performing a random walk to incrementally construct light paths, and compute the PDF of the path $p(\mathbf{x})$.

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h_j(\mathbf{x}_i) L(\mathbf{x}_i)}{p(\mathbf{x}_i)} \tag{2.29}$$

It is important to note that Eq. (2.29) has an alternative interpretation. Consider the random numbers $\mathbf{u}_j \in \mathcal{U}$ that we used to generate the paths $\mathbf{x}_j$, where $\mathcal{U}$ is the *primary sample space (PSS)*, an infinite dimensional hybercube. The path sampling function $P(\mathbf{u})$ is an bijective function between PSS and path space [10]. Essentially we are performing a change of variable in Eq. (2.28) with $|\partial_{\mathbf{u}} P(\mathbf{u})| = \dfrac{1}{p(\mathbf{x})}$.

$$\begin{aligned} I_j &= \int_{\mathcal{P}} h_j(\mathbf{x}) L(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathcal{U}} h_j(P(\mathbf{u})) L(P(\mathbf{u})) |\partial_{\mathbf{u}} P(\mathbf{u})| d\mathbf{u} \\ &= \int_{\mathcal{U}} \frac{h_j(\mathbf{x}) L(\mathbf{x})}{p(\mathbf{x})} d\mathbf{u}, \end{aligned} \tag{2.30}$$

Both path space formulation and PSS formulation are equivalent. For certain algorithms, it is easier to perform computation in one space than another.

2.3.2 Markov Chain Monte Carlo Rendering

In Sec. 2.1.7, we discussed that one could use an importance sampling distribution that is proportional to the integrand to achieve perfect importance sampling and that one could indeed perform such sampling asymptotically, albeit the normalizing constant still needs to be estimated. In this section, we would discuss a method that achieve asymptotic perfect sampling known as Markov Chain Monte Carlo (MCMC).

Markov Chain

A Markov chain is a stochastic process where each state depends only on the previous state. Given a state space Ω, a Markov chain is a sequence of states $\{X_i\}$ where $X_{i+1} \sim K(X_i \to X_{i+1})$, where $K(X_i \to X_{i+1})$ is the probability density of producing X_{i+1} given X_i, known as the *transition function* or *kernel*. The kernel satisfies that

$$\int_\Omega K(x \to y)dy = 1, \forall x \in \Omega \tag{2.31}$$

Given the distribution p_1 of the initial state X_1, we can iteratively compute the distribution of each state X_i as:

$$p_{i+1}(x) = \int_\Omega K(y \to x)p_i(y)dy \tag{2.32}$$

The kernel K can be seen as a linear operator acting on an distribution over Ω. Under certain conditions such as Ω being finite or K being ergodic (Sec. 2.3.2), p_i could converge to some (not necessarily unique) distribution $p(\cdot)$ regardless of p_1; such distribution is referred to as the *stationary distribution* of $\{X_i\}$, which satisfies:

$$p(x) = \int_\Omega K(y \to x)p(y)dy \tag{2.33}$$

Eq. (2.33) is also referred to as the *global balance condition*.

Metropolis-Hastings

MCMC is a family of algorithms that enables importance sampling from arbitrary, intractable distributions $\pi(\cdot)$ by constructing a Markov chain with $\pi(\cdot)$ as its unique stationary distribution. The most common MCMC method is the Metropolis-Hastings (MH) algorithm.

In the Metropolis-Hastings algorithm, given a current state x, we generate a proposal y from the proposal distribution $\mathcal{T}(x \to y)$. To ensure that the resulting Markov chain has the desired stationary distribution, the global balance condition in Eq. (2.33) must be satisfied. However, it is generally difficult to directly enforce such condition as it requires the knowledge of $p(\cdot)$ over the entire domain. Fortunately, a sufficient but not necessary condition for global balance condition called *detailed balance condition* is easier to satisfy.

$$\pi(x)K(x \to y) = \pi(y)K(y \to x) \tag{2.34}$$

Metropolis-Hastings algorithm constructs such K from $\mathcal{T}$ by conditionally accepting or rejecting the proposal y:

$$\pi(x)\mathcal{T}(x \to y)a(x \to y) = \pi(y)\mathcal{T}(y \to x)a(y \to x), \tag{2.35}$$

where $a(x \to y)$ is the probability that the proposal y should be accepted given x. The acceptance probability $a(x \to y)$ is designed to allow the chain to reach stationary distribution as fast as possible:

$$a(x \to y) = min\left(\frac{\pi(y)\mathcal{T}(y \to x)}{\pi(x)\mathcal{T}(x \to y)}, 1\right), \tag{2.36}$$

It is important to note that evaluating Eq. (2.36) does not require knowing the actual $\pi(\cdot)$ but only an unnormalized version of it as the normalizing constant cancels out. We refer to such unnormalized density function as the *target function*. For the Markov chain to converge to the correct stationary distribution regardless of initial state, $\mathcal{T}$ must be ergodic, meaning that $\mathcal{T}$ eventually ensures the chain visits every possible state in the state space. Running such Markov chain for sufficiently long time, the asymptotic distribution of states $\{x_i\}$ will approach $\pi(\cdot)$.

Despite the eventually convergence to target distribution, if the initial states are not at the target distribution, the actual distribution of a finite number samples differs from the target. If we directly use them for estimating expected values, it would result in a biased estimator. This phenomenon is referred to as *start-up bias*. Fortunately, Veach [58] proposed a method to remove start-up bias almost completely.

MCMC method can be used to solve integrals of the form $I = \int f(x)g(x)dx, g(\cdot) \geq 0$ by using MCMC to sample the unnormalized distribution $g(\cdot)$. This integral can be approximated by drawing $\{x_i\}$ from $g(\cdot)$:

$$I \approx \langle I_{MCMC} \rangle = \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)\int g(x)dx, \tag{2.37}$$

where $G = \int g(x)dx$ is the normalizing constant. For MCMC integration to be useful, estimation of G is required to be much cheaper to compute than I directly using Monte Carlo methods.

One caveat of MCMC is that it cannot be easily applied to solve integrals of $I = \int f(x)dx$ directly. If we use MCMC to sample from $f(\cdot)$ directly, will the obtained samples are asymptotically perfectly distributed according to $f(\cdot)$, but we have no ways of estimating the probability density function of each x_i, since that requires the knowledge

of $F = \int f(x)dx$, the exact value we want to compute. Veach [58] proposed a novel way of circumventing the expensive estimation of the normalizing constant by sharing a single normalizing constant across *all* pixels in the image.

To apply MCMC in rendering, we need to solve a few issues. First, MCMC requires a scalar target function to define the target distribution whereas rendering integrates of vector-valued spectra. Second, we need to compute the normalizing factor efficiently. Finally, we want to eliminate start-up bias.

Scalar Importance Function. The first issue is the easiest to solve. Given the vector-valued path contribution $f(x)$, any scalar function $I(x)$ that is positive if any component of $f(x)$ is positive would be a valid target function. To obtain the best importance sampling over $f(x)$, a reasonable choice is to define $I(x)$ as the luminance of $f(x)$ as it is proportional to the energy of the contribution.

Compute Normalizing Factor. In Sec. 2.3.1 we learned that the intensity I_j of each pixel has the form $I_j = \int_{\mathcal{P}} h_j(x)L(x)dx$, where h_j is the pixel filter at pixel j and $f(\cdot)$ is the path contribution function. Instead of running a Markov chain per pixel, we run a single chain over the image-space.

$$I_j \approx \frac{1}{N}\sum_{i=1}^{N}\frac{h_j(x_i)L(x_i)}{I(x_i)}\int_{\mathcal{P}} I(x)dx \tag{2.38}$$

$$= \left(\frac{1}{N}\sum_{i=1}^{N}\frac{h_j(x_i)L(x_i)}{I(x_i)}\right)\left(\frac{1}{M}\sum_{i=1}^{M} I(x_i)\right), \tag{2.39}$$

where we estimate the normalizing factor by sampling M independent paths using a traditional Monte Carlo sampler. Usually the number of MCMC samples N is much larger than M, and since normalizing factor is defined over the image, it converges much faster than per pixel colors, rendering the normalization effectively costless.

Eliminate Start-up Bias. The final remaining piece is to sample the initial states while eliminating start-up bias. Outside computer graphics, the most widely used method known as burn-in is to first run the Markov chain for sometime and discard those samples. However, this method does not guarantee elimination of start-up bias and wastes computation. Veach [58] proposed to reuse the M samples we used to compute the normalizing factor

as initial states and weigh each Markov chain using $w_i = I(x_i)$. While the individual initial states are not distributed according to the target, their weighted distribution is, and therefore completely removes the start-up bias. The only drawback is that the resulting estimator is no longer consistent. For example, if one very unlucky run produces initial samples with zero contribution, then the MCMC rendering would produce a black image. An better alternative is to resample K states out of the initial M states with resampling with proportional to $I(x_i)$ such that initial states with zero contribution will never be selected. This approach guarantees a consistent estimator. While the distribution of resampled states is not exactly the target distribution, it is a very good approximation and usually the remaining bias is imperceptible. We would use the alternative approach in all MCMC rendering method presented in this thesis.

Reusing Rejected Samples

A common practice in rendering is to accumulate the contributions coming from both the proposal and the current state regardless of the outcome of the acceptance of the proposal [60]. The contribution from the proposal y is weighted by $a(x \to y)$ and the contribution from the current state is weighted by $1 - a(x \to y)$. This technique would not introduce bias to the result because y is essentially replacing the stochastic process of accumulating either x or y alone by its mean.

Chapter 3

Shape Sampling and Path Reusing

3.1 Line segment sampling

Line segments are one of the simplest shapes other than points. Previous work has explored accelerating Monte Carlo rendering by either generating segments in the image space or analytically integrating over line segments. Segment samples were used to compute analytic aliasing [29], motion blur [18, 19], depth-of-field [57]. Barringer et al. [4] employ line samples to compute accurate visibility of thin curves. Sun et al. [56] construct line segment samples with blue-noise properties using frequency analysis. Shirley et al. [52] generate stratified 2D segments by wrapping stratified points in 2D unit squares. Singh et al.[53] analyze the variance and convergence properties of line and segment samples. Our work focuses on sampling a set of 2D points arranged in a specified manner (shape) based on a point sample, which has not been addressed before.

3.2 High dimensional samples in volume rendering

Volumetric density estimation is a popular method for efficiently simulating volumetric light transport. Since the original volumetric photon mapping algorithm [28], numerous improvements have been made to replace point estimates with high dimensional photon primitives. Jarosz et al. [27, 25, 26] replace point-wise query of nearby photons with querying along beams and extended density estimation to beam samples from both camera and light rays and was later followed up to introduce a progressive estimator similar to progressive photon mapping [22], combined with a proposed unbiased transmittance estimator for

heterogeneous volume. Novak et al. [44] extend the virtual point light method by replacing point lights with ray lights. Krivanek et al. [35] use MIS to combine multiple point and beam estimators with Monte Carlo path sampling. Bitterli et al. [11] generalize point and beams estimates to 2D photon planes and 3D volumes, later Deng et al.[14] extend it to include photon surfaces. Liu et al. [39] extend high dimensional photon primitives to the temporal domain for time-of-flight rendering.

High-dimensional photon samples are usually viewed as replacing free-flight distance sampling with its expected value. Many methods thus take advantage of the fact that such expected behavior can be computed analytically in homogeneous media. While heterogeneous volume is not a theoretical limit to many of these methods, their performance deteriorates due to the need to evaluate visibility and transmittance via point sampling. Our work instead builds upon a conversion of a shape into a set of points, making it generalizable for any problems as long as we can perform point sampling.

3.3 Path reusing

In contrast to many other problems solved using Monte Carlo integration, Monte Carlo rendering has one unique property that one needs to solve the integral equation not only once, but millions of times, once at each pixel. The integrand of those equations are usually very similar or *correlated*, since ray traced for nearby pixels usually undergo similar scattering event and producing similar colors. Therefore another approach to improve efficiency is to reuse the light paths sampled at nearby pixels with cheaper cost, which increases the effective samples at each pixel.

3.3.1 Shift Mapping

First we discuss a technique that is not only highly related to our proposed method but also make other path reusing method possible. Shift mapping is a deterministic transformation that maps light paths sampled in one pixel to another efficiently. Given a light path $\mathbf{x} \in \Omega_i$ sampled at pixel i, the shift mapping $T_{i\to j}$ generates a highly correlated path $\mathbf{y} = T_{i\to j}(\mathbf{x}) \in \Omega_j$. To use $\mathbf{y}$ in Monte Carlo rendering, we would also like to evaluate the probability density of generating $\mathbf{y}$. Unfortunately, evaluating such density is difficult for some general $T_{i\to j}$ as it requires evaluating a marginal density that considers all possible $\mathbf{x}$ that generates the same $\mathbf{y}$. To make the computation tractable, we enforce shift mappings to be *invertible*. Under the invertibility constraint, we are able to efficiently compute the probability density

of $\mathbf{y}$ by $p(\mathbf{y}) = p(\mathbf{x})/\left|\partial_{\mathbf{x}} T_{i\to j}\right|$ according to the change of variable formula, where $\left|\partial_{\mathbf{x}} T_{i\to j}\right|$ is the Jacobian determinant of the shift mapping. Accurately evaluating the Jacobian is essential for unbiased rendering with shift mappings. Jacobian determinants can be either evaluated under PSS parameterization or path space parameterization. For certain shift mappings, it is much easier to evaluate the Jacobian in one parameterization than another.

There exists many different strategies for shift mappings. We will briefly discuss those variants:

Random Replay. As path sampling is completely deterministic w.r.t the underlying random numbers producing the path, the simplest shift mapping consists of reusing the random number sequence for sampling paths at different pixels. For nearby pixels, it is likely that the light path interacts with similar geometry and undergoes the same type of scattering events at each vertex, therefore producing a similar path as the original. Random replay works most well for glossy and specular surfaces as the scattered direction is highly concentrated given similar incoming directions. The main benefit of random replay is that it is almost trivial to implement. Under PSS parameterization, the Jacobian determinant of random replay is simply 1. While being simple, the two main drawbacks of random replay is that 1. the cost of shift mapping is the same as sampling a new path; 2. the shifted path can diverge significantly from the base path as it might interaction with completely different materials or geometries.

Reconnecting First Vertex. If we want to decrease the cost of shift mapping as much as possible, we would like to reuse most of the vertices already generated in the base light path. Reconnecting first vertex allows maximum use of the base path by reconnecting the shifted path to the base path as early as possible: Once the camera ray hits any surface, a visibility test is then perform between the hit point and the next vertex in the base path. The shifted path is accepted if both points are visible and rejected otherwise. This strategy works well for diffuse surfaces but works poorly for high specular surfaces as the shifted path is likely to be invalid or have very little contribution.

Half-vector Copy. Half-vector copy is another shift mapping strategy designed to handle high specular vertices. As the name suggests, it copies all half-vectors in the base path to the shifted path and compute ω_i based on surface normal and copied half-vectors. This strategy naturally preserves the incoming direction under local tangent frame thus prevent generating invalid specular paths.

Manifold Exploration. While random replay and half-vector copy is guaranteed to produce valid specular paths, the resulting shifted path might diverge significantly from the base if the shifted path hits surfaces with very different normals. Manifold exploration [24] attempts to resolve this issue by first generating an invalid but structurally highly

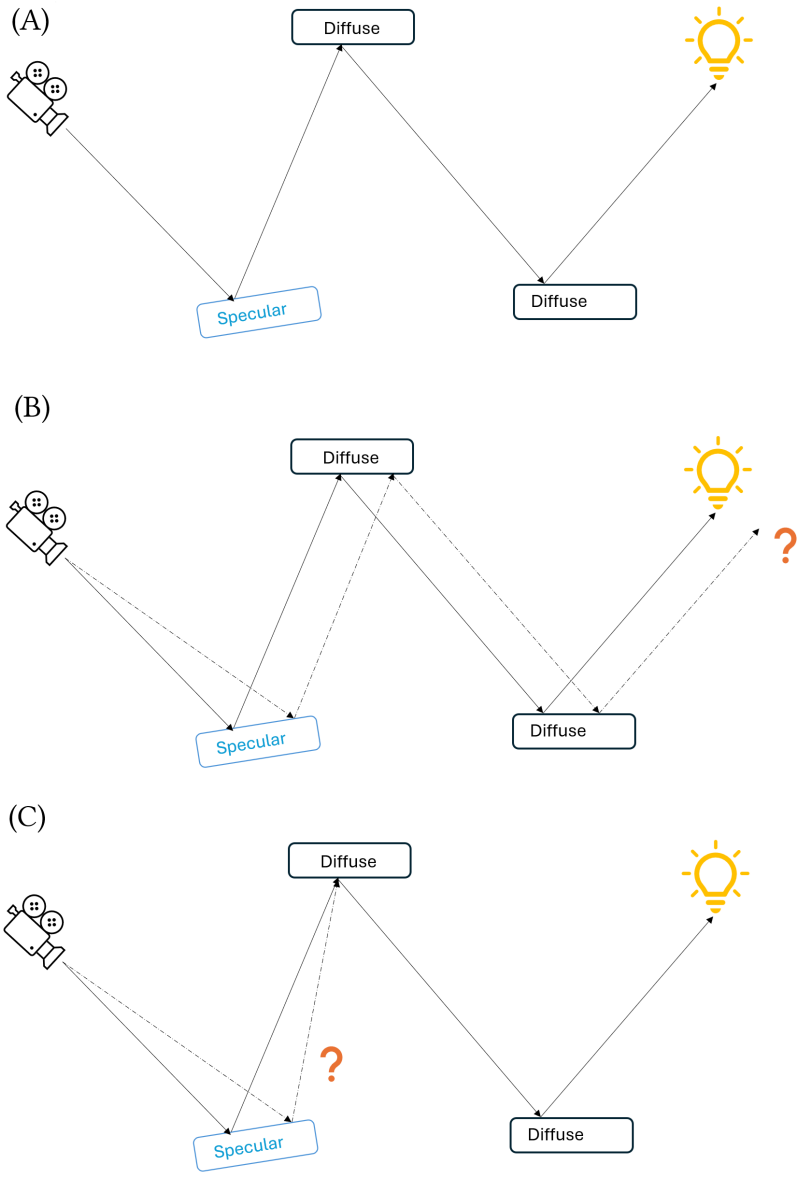


Figure 3.1: (A) A typical setup of shift mapping involving specular and diffuse vertices. (B) Replaying random numbers handles specular vertices well but is prone to produce very different paths due to lack of path space information. (C) Reconnection guarantees the shifted path shares a common suffix with the base path but is likely to produce invalid specular interactions.

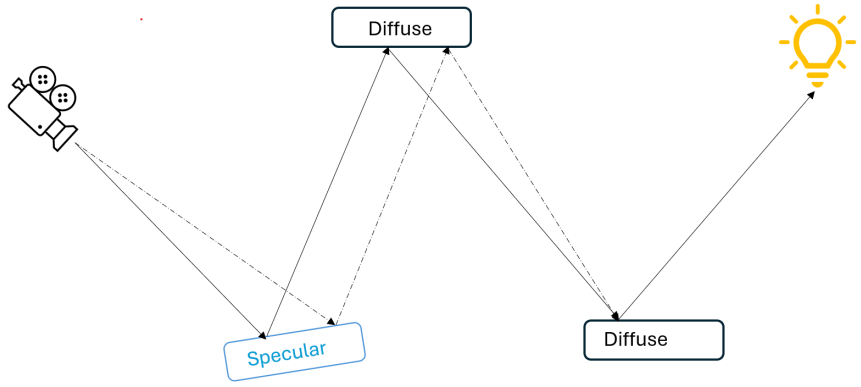


Figure 3.2: Hybrid shift mapping postpones reconnection until a diffuse vertex is encountered, handling both type of vertices well while sharing most base path as possible.

similar shifted path and then run an iterative Newton solver to perturb the shifted vertices to correct for invalid specular interactions, resulting in high quality shifted paths. The main drawback of manifold exploration is its high computation overhead and implementation complexity of the Newton solver.

Hybrid (Random Replay + Reconnection.) Since random replay and reconnection works well for specular and diffuse surfaces respectively, a more robust shift mapping strategy is to carefully combine both of them to handle different kinds of surfaces. In hybrid shift mapping, we employ random replay to handle specular surfaces and attempt to reconnect to the first available diffuse vertex in the base path whenever possible. Hybrid shift mapping offers a simple yet robust solution for specular and diffuse surfaces, as its Jacobian determinant is of simpler form compared to the more robust but complicated half-vector copy or manifold shift mapping.

Another benefit of hybrid shift mapping is that it is efficient to implement on GPUs. Half-vector copy and manifold shift mapping requires storage of a entire path. On most offline rendering systems, storing each vertex requires hundreds of bytes of storage and storing the entire path requires both significant memory overhead as well as huge performance penalty due to register spilling. Hybrid shift mapping only requires storing the first connectable vertex thus require much smaller memory overhead.

With these advantages, we elected to implement hybrid shift mapping as part of our rendering system. We now discuss in more detail about the implementation.

Jacobian Determinants Given a pair of base path $\overline{\mathbf{x}} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ and shifted path $\overline{\mathbf{y}} = \{\mathbf{y}_1, ...\mathbf{y}_{k-1}, \mathbf{x}_k, ..., \mathbf{x}_n\}$, where $\{\mathbf{y}_1, ..\mathbf{y}_{k-1}\}$ are generated through random replay. It is easier to evaluate the Jacobian determinant under PSS parameterization since most of the differing vertices are generated by random replay. For vertices 1 to $k-1$, the Jacobian determinant under PSS is simply 1.

$$\left|\frac{\partial \mathbf{u}_i^{\mathbf{y}}}{\partial \mathbf{u}_i^{\mathbf{x}}}\right| = 1, 1 \leq i < k-1 \tag{3.1}$$

Connecting $\mathbf{y}_{k-1}$ to $\mathbf{x}_k$ changes the direction hence the random numbers for generating $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$. The Jacobian determinant for those vertices is given by [38]

$$\begin{aligned}
\left|\frac{\partial \mathbf{u}^{\mathbf{y}}}{\partial \mathbf{u}^{\mathbf{x}}}\right| &= \left|\frac{\partial \mathbf{u}_{k-1}^{\mathbf{y}}}{\partial \mathbf{u}_{k-1}^{\mathbf{x}}}\right| \left|\frac{\partial \mathbf{u}_k^{\mathbf{y}}}{\partial \mathbf{u}_k^{\mathbf{x}}}\right| \\
\left|\frac{\partial \mathbf{u}_{k-1}^{\mathbf{y}}}{\partial \mathbf{u}_{k-1}^{\mathbf{x}}}\right| &= \left|\frac{\partial \mathbf{u}_{k-1}^{\mathbf{y}}}{\partial \omega_{k-1}^{\mathbf{y}}}\right| \left|\frac{\partial \omega_{k-1}^{\mathbf{y}}}{\partial \omega_{k-1}^{\mathbf{x}}}\right| \left|\frac{\partial \omega_{k-1}^{\mathbf{x}}}{\partial \mathbf{u}_{k-1}^{\mathbf{x}}}\right| \\
&= \frac{p(\mathbf{y}_{k-2} \to \mathbf{y}_{k-1} \to \mathbf{y}_k)}{p(\mathbf{x}_{k-2} \to \mathbf{x}_{k-1} \to \mathbf{x}_k)} \left|\frac{\partial \omega_{k-1}^{\mathbf{y}}}{\partial \omega_{k-1}^{\mathbf{x}}}\right|, && (3.2) \\
\left|\frac{\partial \mathbf{u}_k^{\mathbf{y}}}{\partial \mathbf{u}_k^{\mathbf{x}}}\right| &= \left|\frac{\partial \mathbf{u}_k^{\mathbf{y}}}{\partial \omega_k^{\mathbf{y}}}\right| \left|\frac{\partial \omega_k^{\mathbf{y}}}{\partial \mathbf{u}_k^{\mathbf{x}}}\right| && (3.3) \\
&= \frac{p(\mathbf{y}_{k-1} \to \mathbf{y}_k \to \mathbf{y}_{k+1})}{p(\mathbf{x}_{k-1} \to \mathbf{x}_k \to \mathbf{x}_{k+1})}, && (3.4)
\end{aligned}$$

where $\left|\frac{\partial \omega_{k-1}^{\mathbf{y}}}{\partial \omega_{k-1}^{\mathbf{x}}}\right|$ is the Jacobian determinant of reconnection under solid angle parameterization:

$$\left|\frac{\partial \omega_{k-1}^{\mathbf{y}}}{\partial \omega_{k-1}^{\mathbf{x}}}\right| = \left|\frac{cos\theta_k^{\mathbf{y}}}{cos\theta_k^{\mathbf{x}}}\right| \frac{||\mathbf{x}_k - \mathbf{x}_{k-1}||^2}{||\mathbf{x}_k - \mathbf{y}_{k-1}||^2}, \tag{3.5}$$

where $\theta_k^{\mathbf{x}}$ is angle between $\omega_{k-1}^{\mathbf{x}} = \mathbf{x}_{k-1} \to \mathbf{x}_k$ and surface normal at $\mathbf{x}_k$

Reconnection Criteria Classification of diffuse and specular vertices is usually done by examining the roughness parameter of BSDFs and the vertex is classified to be non-diffuse if its roughness is below a threshold. In addition to vertex roughness, we also impose a distance criteria such that two connectable vertices must be separated by a minimal distance to ensure that inverse distance squared term in Eq. (3.5) does not blow up.

Ensuring Bijectivity We need to carefully when deciding whether a vertex is connectable, due to the invertibility constraint: We want to make sure that if we apply the shift mapping on the shifted path, it always result in the same base path. Therefore, when checking connectability we not only need to check if the next vertex in the base path is diffuse, the current shifted vertex is also required to be diffuse as well. If $\mathbf{y}_{k-1}$ does not satisfy the roughness criteria, the shift is rejected. Furthermore, if either the distance or roughness criteria is satisfied for any pair of $\mathbf{y}_{j-1}, \mathbf{y}_j, j < k$ the shift is also rejected.

3.3.2 Bekaert style-path reusing

Bekaert style-path reusing (BPR) [6] and its generalization [5] amortizes sampling cost shift mapping the current paths to all other pixels in an $M \times M$ tile $\mathcal{T}$. The reused paths are weighted by balanced heuristics over all paths in the tile.

$$L_i \approx \frac{1}{N} \sum_{i=k}^{N} \sum_{j \in \mathcal{T}} w_{i,j}(\mathbf{x}_{j,k}) \frac{L(T_{j \to i}(\mathbf{x}_{j,k})) |\partial_{\mathbf{x}} T_{j \to i}|}{p_j(\mathbf{x}_{j,k})} \tag{3.6}$$

$$w_{i,j}(\mathbf{x}) = \frac{p_j(\mathbf{x})}{\sum_{j \in \mathcal{T}} p_j(\mathbf{x}) |\partial_{\mathbf{x}} T_{i \to j}|}, \tag{3.7}$$

where we use the identity $|\partial_{\mathbf{x}} T_{j \to i}| = |\partial_{\mathbf{x}} T_{i \to j}|^{-1}$. The weight $w_{i,j}(\mathbf{x})$ is constant for all pixels in $\mathcal{T}$ and can be computed efficiently. While BPR usually improves efficiency over a baseline path tracer, it produces blocky artifact which makes it less appealing in practice.

3.3.3 ReSTIR

Resampled Importance Sampling (RIS) is a powerful method to produce sample that approximately distributed according to a target distribution $p(\cdot)$, which we briefly mentioned in Sec. 2.3.2. The idea is to first produce M i.i.d. samples from a simpler distribution $q(\cdot)$ and randomly select one sample Y from those samples with probability proportional to $w_i \propto \frac{p(X_i)}{q(X_i)}$. It can be shown that with increased N, the actual distribution of resampled samples approaches $p(\cdot)$. While computing the actual PDF of Y is intractable, the *unbiased contribution weight* $W_Y = \frac{1}{p(Y)} \sum_{i=1}^{M} w_i$ unbiasedly estimates the PDF of Y. The resampling is usually implemented using reservior sampling for its constant memory cost.



Figure 3.3: Bekaert style-path reusing produces visible blocky artifact due to its tile-wise path reuse strategy.

```
class Reservoir:
    x: Sample
    w_sum: float
    W: float # unbiasedly estimates pdf(x)

    def update(self, x: Sample, w: float):
        self.w_sum += w
        if rand() < w / self.w_sum:
            self.x = x
        self.W = self.w_sum / TargetFunction(x)

    def merge(self, r: Reservior):
        y, jacobian = ShiftMapping(r.x)
        self.update(y, MIS(y) * r.W \
            * TargetFunction(y) * jacobian)
```

Listing 3.1: RIS using weighted reservoir sampling and merging reservoir across different domains via MIS and shift mapping.

Spatio-temporal reservoir sampling (ReSTIR) [9, 38] further improves resampling efficiency by merging reservoirs from possibly different domains. The weights are modified to $w_i = m_i(T_i(X_i))p_i(T_i(X_i))W_i|\partial_X T(X_i)|$ to account for the density change due to shift mapping. An MIS weight $m_i(\cdot)$ is also included to prevent overcounting of sampling techniques. ReSTIR first samples a set of initial paths per pixel and reuses resampled light paths spatio-temporally via shift mapping, amortizing sampling cost. Using balanced heuristics for $m_i(\cdot)$ is not practical as it requires evaluating $p(\cdot)$ and $|\partial T_X|$ over all possible pixels, resulting in an quadratic cost of the number of reusing pixels. Instead, Bitterli [9] proposed a simplified variant as *pairwise MIS* that only computes balanced heuristics between the pair of samples involved during shift mapping, reducing the cost to linear. We provide a more detail discussion on pairwise MIS and its relation to our work in Sec. 4.2.8.

While shift mapping reduces computational cost in sample reusing across domains, ReSTIR gained most of its advantage by using RIS to quickly approximate the target distribution, instead of simply reusing light paths via shift mapping, as demonstrated by Lin et al [38]

3.3.4 Gradient-domain rendering

Gradient-domain rendering (GDR) [37, 33, 40, 55, 20] is a family of method that estimates image gradients and reconstruct the original image via screened Poisson reconstruction.

The difference, hence gradient, between pixel i and j can be estimated by:

$$\Delta_{ij} = I_j - I_i = \int_{\mathcal{P}} h_i(\mathbf{x}) \Big(f(T_{i\to j}(\mathbf{x}))|\partial_{i\to j} T_{\mathbf{x}}| - f(\mathbf{x}) \Big) d\mathbf{x} \tag{3.8}$$

$$\langle \Delta_{ij} \rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{h_i(\mathbf{x}_k)(f(T_{i\to j}(\mathbf{x}_k)))|\partial_{i\to j} T_{\mathbf{x}_k}| - f(\mathbf{x})}{p(\mathbf{x}_k)} \tag{3.9}$$

Here we assume the shift mapping is always invertible for simplicity. A practical implementation also considers MIS for sampling gradients but these discussions are beyond the scope of this thesis. After obtaining low variance gradient images, the final image is reconstructed by

$$I = arg \min_{I} ||D_x I - G_x||_p^p + ||D_y I - G_y||_p^p + \lambda ||I - I_0||_p^p \tag{3.10}$$

where G_x, G_y are rendered gradient images, D_x, D_y are finite difference operators on the image I, λ controls the strength of noise filtering of the reconstruction process. GDR accelerates Monte Carlo rendering in two ways:

1. GDR leverages shift mapping to produce correlated paths and thus be able to estimate image gradient with lower variance than the image directly, as positive correlation reduces variance in estimating differences.

2. GDR uses a reconstruction step that filters high frequency noises, leaving a noise-free final image.

GDR can also be interpreted as a form of path reuse since a sampled path will contributed to multiple pixels through shift mapping and Poisson reconstruction. Our method does not rely on a separate reconstruction step, and it can be seen as yet another form of path reuse besides GDR.

Chapter 4

Image-space Shape Splatting

Monte Carlo rendering estimates the expected values of sensor responses by sampling light paths between light sources and sensors. This process involves sampling the contributing light paths for each pixel (sensor) in an image. For example, path tracing, as described by [30], samples a path that goes through each pixel from the sensor side and repeats this process multiple times for all the pixels. The path integral formulation [58] models this process by considering a single point on the image plane for each sampled light path and accumulating its contribution to the relevant pixels. Nearly all existing Monte Carlo rendering methods build upon this concept, with each path sample representing a *point* sample on the image.

We propose a novel framework that extends the traditional correspondence between a light path and a single point to multiple points on the image. These multiple points often correspond to neighboring pixels forming a shape (e.g., a square centered at each pixel). Therefore, we refer to the set of corresponding pixels as *shapes* and the light sample shared across them as *shape samples*.

While it may be tempting to think that path reuse [6, 5, 38]already addresses this task, path reuse only achieves to generate multiple paths at an amortized cost per sample with the use of shift mappings and the overall cost is directly proportional to the number of shift mapping performed. If shift mapping is as costly or slower than generating a new sample from scratch, path reuse provides no benefit.

Our proposed method instead achieve a *sublinear* cost to the number of pixels within a shape, allowing us to retain some improvement even when shift mapping is slow or as costly as sampling a new path. We can also apply our shape splatting to the existing path reuse methods to further improve its efficiency.

The key idea is to approximate the contributions of all the points in a shape using a biased yet cheap estimator and apply debiasing [43] to remove the bias. With a carefully constructed debiasing estimator, we can compute the unbiased contributions of a shape of ≈ 60 pixels by generating only 4 point samples on average, for example. We formulate shape sampling and splatting as a multi-proposal MCMC step and weight the contributions according to MCMC acceptance probabilities. Our MCMC formulation allows us to effectively combine path samples from all the overlapping shapes without relying on multiple importance sampling which would otherwise incur a quadratic computation cost. We demonstrate the effectiveness of our framework by adding shape splatting to various Monte Carlo rendering methods. Shape-splatting variants of those methods consistently outperform the point-splatting counterparts, and can be added even on top of path reuse. Our technical contributions are a debiasing estimator for unbiased shape splatting with sublinear cost to the shape size, and an MCMC formulation of shape splatting that allows a fast evaluation of the sample weights with lower variance than alternative pairwise MIS (Sec. 4.2.8).

4.1 A Simplified Example

Let us go over a simplified example of sampling shapes from a given 2D distribution to establish the concept of shape splatting in this thesis. Figure 4.1 shows this example where we are given a PDF $p(x, y)$ in the 2D space of (x, y) as an image and (x, y) represents a 2D pixel index. The variables (x, y) are discrete since they are integers indexing pixels. Let us consider the problem of copying the image $I(x, y)$ by sampling from $p(x, y)$, which is equivalent to estimating a histogram of samples generated according to $p(x, y)$ as was also used by [12] to explain MCMC.

Point samples Samples are usually 2D *points* $(x_i, y_i) \propto p(x, y)$ and the histogram is estimated by counting the number of samples within each pixel. Due to the stochastic nature, any pixel can potentially have zero samples given a finite number of total samples. Figure 4.1 shows the results of this baseline, and one can see that there are many zero histogram bins (black pixels) when the number of samples is small, and each sample contributes to *one* pixel.

Shape samples Instead of point samples, we propose to sample shapes that are still distributed according to the same PDF $p(x, y)$. We define a shape as a set of (usually

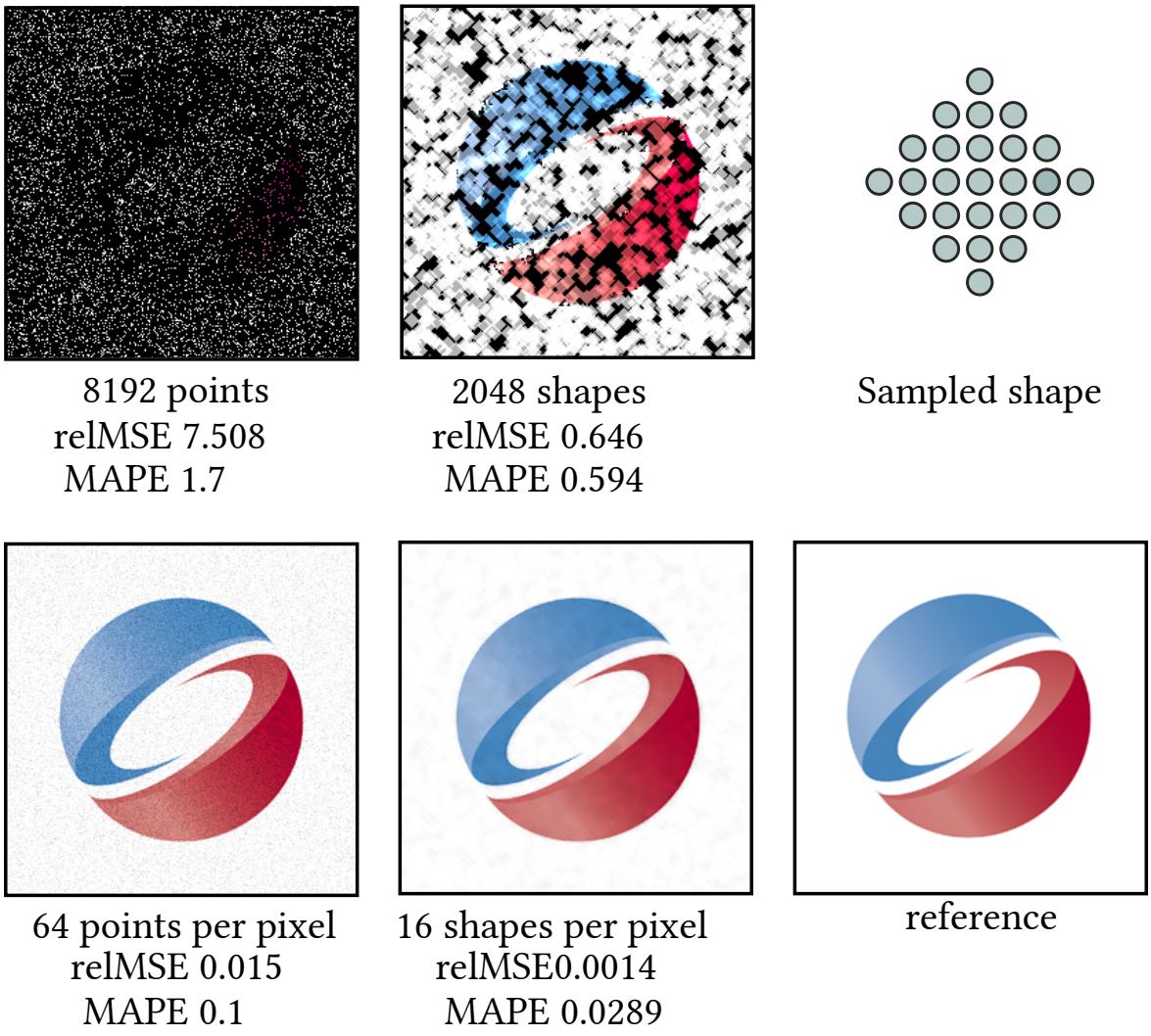


Figure 4.1: We compare sampling a 2D distribution using points and shapes under the *equal point-sample counts*. The shape is a rotated quad with 61 points (pixels). Our method sample each shape with the cost of 4 point samples on average. Shift mapping in this example has the same cost as generating a new sample, so any gain is not coming from sample reuse. Shape samples fill in more pixels at the same cost, making it far more efficient than point sampling. *Relative Mean Square Error (relMSE)* is defined as $relmse(I, I_{gt}) = \frac{(I-I_{gt})^2}{I_{gt}^2+0.01}$. *Mean Absolute Percentage Error (MAPE)* is defined as $mape(I, I_{gt}) = \frac{|I-I_{gt}|}{|I_{gt}|+0.01}$
.

consecutive) points. Figure 4.1 shows our results, where the shape is a rotated quad that contains 61 points (pixels). One can recognize individual shapes being "splatted" as samples when the number of samples is small. We use "points" and "pixels" interchangeably in this thesis whenever feasible. If the cost of generating each shape sample is less than that of generating all the point samples within the shape, shape samples effectively increase the number of samples.

Our main technical contribution is a general framework to achieve efficient shape splatting. Our framework allows us to splat this shape sample of 61 points in Figure 4.1 at the cost only of 4 point samples on average. Shape splatting is also more efficient than point splatting with an equal number of point samples taken (i.e., 4 times more for point samples) as Figure 4.1 demonstrates.

4.2 Shape Splatting

We first explain our method for splatting 1D shapes on an 1D image, and then we will discuss generalization to 2D shapes in Sec. 4.2.4. Listing 4.1 provides a pseudocode of our method. Consider an image I of N pixels. We focus on scalar pixel values in the following, and colors are handled similarly to what is commonly done in MC rendering.

Let us consider a more general problem than the example in Figure 4.1 where the histogram is proportional to an integral $I_i = \int g(i, \mathbf{x}) d\mathbf{x}$ of a higher dimensional function $g(i, \mathbf{x})$. The example in Figure 4.1 corresponds to a case where the integrals I are directly given as an image. This generalized problem covers rendering with the path integral formulation Sec. 2.3 where the pixel value I_i is determined by integrating a product of the pixel filter $h(i, \mathbf{x})$ and the contributions $f(\mathbf{x})$ of light transport paths $\mathbf{x}$ as

$$I_i = \int_{\mathcal{P}} g(i, \mathbf{x}) d\mathbf{x} = \int_{\mathcal{P}_i} \underbrace{h(i, \mathbf{x}) f(\mathbf{x})}_{g(i, \mathbf{x})} d\mathbf{x}. \tag{4.1}$$

The integration domain changes to $\mathcal{P}_i$ as a subset of the path space $\int_{\mathcal{P}}$ where $h(i, \mathbf{x}) > 0$. Let us consider $h(i, \mathbf{x}) = 1$ within the pixel i in the following so that we can simplify as $g(i, \mathbf{x}) = f(\mathbf{x})$ over $\mathcal{P}_i$. The vector of pixel values I is thus defined as

$$I = \begin{bmatrix} \vdots \\ I_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \int_{\mathcal{P}} g(i, \mathbf{x}) dx \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \int_{\mathcal{P}_i} f(\mathbf{x}) dx \\ \vdots \end{bmatrix}. \tag{4.2}$$

```
# Input: pixel index i
# Output: An unbiased estimate of contribution
#         of the shape centered at i
def UnbiasedShapeSplatting(i):
    # 1. Sample the primary sample x from p(·)
    x, S = SamplePathAtPixel(i), ShapeAt(i)
    S = reorder pixels in S by a hilbert curve Sec. 4.2.4
    k, pmf_k = RandInt(1, |S|-1)

    # 2. Sample secondary pixels and path samples
    #    via shift mapping
    js = sort(
        {i} + sample k distinct pixels from S-{i})
    ys = [T_{i→j}(x) for j in js]
    # Compute MCMC weights that keeps p(·) invariant
    #  Eq. (4.6)
    ws = [MCMCWeights(x, y, |∂_x T_{i→j}(x)|) for y in ys]

    # 3. Debiasing. Sec. 4.2.3
    def C(x): return f(x) / p(x)
    D(1) = Ones(|S|) * C(x)
    D(k+1) = Interpolate(C(ys) * ws)
    res = D(1)
    # Eq. (4.9)
    for j′ in js:
        if j′ == i:
            continue
        js′ = js - {j′}
        ys′ = [T_{i→j}(x) for j in js′]
        ws′ = [MCMCWeights(x, y, |∂_x T_{i→j}(x)|) for y in ys′]
        D(k) = Interpolate(C(ys′) * ws′)
        res += (D(k+1) - D(k)) / (pmf_k * k) # Eq. (4.10)
    return res
```

Listing 4.1: Overview of our method. Note we demonstrate a naive implementation of debiasing here for simplicity. In practice, we cache the evaluation of $f(\cdot)$ and compute all differences between interpolations in a single pass.

To estimate I, we first sample a pixel index i according a discrete distribution $p_i(\cdot)$ and then sample a path $\mathbf{x} \sim p(\mathbf{x}|i)$. We refer to such $\mathbf{x}$ as *point samples*. While this formulation may sound incompatible with a typical scenario where we just loop over all the pixels and sample paths within each pixel (e.g., path tracing), it is equivalent to sample i from uniform distribution but with perfectly stratified sampling. The distribution of pixel index i and path $\mathbf{x}$ form a joint distribution $(i, \mathbf{x}) \sim p(\cdot)$. Since each path corresponds to a single pixel index i, we simply refer to $\mathbf{x} \sim p(\cdot)$ as sampling both the index and path. Once a path $\mathbf{x}$ is sampled at i, we would like to splat its contribution to other pixels such that the sum of all such shape splatting estimates I in an unbiased manner.

Shapes We define a shape $\mathcal{S}$ as an *ordered* set of pixel indices $\mathcal{S} = \{i_1, ..., i_{|\mathcal{S}|}\}(i_1 < ... < i_{|\mathcal{S}|})$ where $|\mathcal{S}|$ is the number of points within the shape (e.g., 61 in Figure 4.1). We index each shape as $\mathcal{S}^i$ to enable a different shape per pixel. We denote $C(\mathcal{S}^i)_j$ as the contribution to the pixel j by the shape $\mathcal{S}^i$. We also refer to the pixel i as the *center pixel* of $\mathcal{S}^i$. A contribution to any other pixel that is not in the shape is zero.

The exact definition of $C(\mathcal{S}^i)_j$depends on how shape splatting is performed. The only requirement is that the sum of $C(\mathcal{S}^i)_j$ of all pixel i unbiasedly estimates the image I

$$I_j = \sum_{i=1}^{N} C(\mathcal{S}^i)_j. \tag{4.3}$$

In each shape $\mathcal{S}^i$, we call the corresponding point sample that goes though the pixel i as the *primary* sample, and all the other point samples as the *secondary* samples.

Shape as Multiple Point Samples A naive approach to sample a shape is to sample all the (primary and secondary) points within the shape by performing point sampling repeatedly. Each pixel value I_j is estimated by taking an average of all the point samples from all the shapes that fall within the pixel j. The cost of sampling one shape in this naive approach is *equivalent* to sampling all the points within the shape, so this approach would *not* bring any benefit over point sampling with an equivalent number of samples. It nevertheless is useful as a baseline for our algorithm.

4.2.1 Amortized Shape Splatting via Mutations

Instead of sampling each point within a shape $\mathcal{S}^i$ from scratch, one can perform path reuse of the primary sample (which goes through i) to generate all the secondary samples

within the shape. We focus on the contributions coming from a given shape $\mathcal{S}^i$ to all the other pixels (points) $\{i_1, ..., i_{|\mathcal{S}|}\}$ within it, which is equivalent to consider *splatting* of the contributions of each shape as opposed to *gathering* [16]. They are mathematically equivalent, but we found that splatting partially reused samples naturally maps to *mutation* in MCMC rendering.

Mutation involves generating a secondary sample $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{|\mathcal{S}|}}\}$ given the primary sample $\mathbf{x}_i$ (e.g, from a path tracer) and a distribution of mutated samples is called a proposal distribution. Since we want to splat equal contribution to other pixels to maximize splatting efficiency, we choose a simple proposal distribution $T_i(j)$ that mutates pixel indices uniformly within the support of $\mathcal{S}^i$ at equal probabilities:

$$T_i(j) = \begin{cases} 1/|\mathcal{S}^i| & (j \in \mathcal{S}^i) \\ 0 & (j \notin \mathcal{S}^i). \end{cases} \tag{4.4}$$

Because $T_i(j)$ is a simple binary function, we can mutate the primary samples $\mathbf{x}_i$ to the all the other secondary samples *deterministically* by selecting each j in $\mathcal{S}^i$ exactly once over $|\mathcal{S}^i|$ proposals, which is equivalent to perfectly stratified sampling of $T_i(j)$. While there are many different approaches to mutate or reuse path samples, we use shift mapping techniques developed for gradient-domain rendering [37, 33, 5]. Due to the amortized cost per sample in mutation (or shift mapping), the cost of generating all the points within a shape is less than generating an equal number of point samples.

In general, secondary samples in the pixel j mutated to the pixel i will be distributed differently than samples within the pixel i. We need to take this difference into account to define a valid estimator of I_i. When all the pixels have the same shape $\mathcal{S}$ and the Jacobian of shift mapping from i to j is $\left|\partial_{\mathbf{x}} T_{i\to j}\right|$, the probability density of generating a sample from the shape at the pixel i to the pixel j is $p(\mathbf{x}|i)/\left|\partial_{\mathbf{x}} T_{i\to j}\right|$ which in general would be different from the probability density $p(\mathbf{x}|j)$ at the pixel j itself.

While this problem setting fits well with multiple importance sampling [59], the number of techniques here is $O(|\mathcal{S}^i|)$ and we found that the evaluation cost of the balance heuristic is impractical because each shape will cost $O(|\mathcal{S}^i|^2)$ for having $|\mathcal{S}^i|$ points and the balance heuristic for each point takes $O(|\mathcal{S}^i|)$. We propose an alternative based on a common practice of splatting proposals [60]. While we also experimented with a more practical alternative of pairwise MIS [9] (see Sec. 4.2.8), we found that our MCMC approach generates more accurate results, especially when combined with our debiasing estimator.

4.2.2 MCMC-based Weighting for Shape Splatting

In Sec. 2.3.2, we discussed the basics of MCMC for rendering applications. We weight the contributions of all the points in an arbitrary shape sample using the aforementioned technique of splatting proposals by the expected MCMC decision outcome. For mutations implemented as invertible shift mappings, the ratio $\frac{\mathcal{T}(y\to x)}{\mathcal{T}(i\to j)}$ turns out to be exactly $\left|\partial_{\mathbf{x}} T_{i\to j}\right|$. In the context of our work, the primary light paths are generated from baseline point sampling method with density $p(\mathbf{x})$. We design the Markov chain to *preserve* this density exactly, by setting $p(\mathbf{x})$ as its stationary distribution. The acceptance probability, thus the weight, can be computed as:

$$a(\mathbf{x}_i \to \mathbf{x}_j) = min\left(\frac{p(\mathbf{x}_j)|\mathcal{S}^i|}{p(\mathbf{x}_i)|\mathcal{S}^j|}\left|\partial_{\mathbf{x}} T_{i\to j}\right|, 1\right), \tag{4.5}$$

with an additional $\frac{|\mathcal{S}^i|}{|\mathcal{S}^j|}$ factor to account for secondary pixels being sampled uniformly from shapes with different sizes. The contribution of each pixel in the shape can be estimated by computing the expected outcome of the accept decision accordingly:

$$w_{i\to j} = \begin{cases} \frac{1}{|\mathcal{S}^i|} a(\mathbf{x}_i \to \mathbf{x}_j) & (i \neq j) \\ \frac{1}{|\mathcal{S}^i|}(1 + \sum_{k\neq i} 1 - w_k) & (i = j). \end{cases} \tag{4.6}$$

By mutating $\mathbf{x}_i$ to all pixels in $\mathcal{S}^i$ and weighting according to Eq. (4.6), we obtain a *pointwise* estimator $P(\mathcal{S}^i)$ of $C(\mathcal{S}^i)$.

$$\langle C(\mathcal{S}^i)_j \rangle = P(\mathcal{S}^i)_j = w_{i\to j}\frac{f(\mathbf{x}_j)}{p(\mathbf{x}_j)} \quad \text{where } \mathbf{x}_j = T_{i\to j}(\mathbf{x}_i) \tag{4.7}$$

Differences from MCMC Unlike MCMC, we already have samples that are distributed according $p(\cdot)$ at each pixel. By setting the target as $\pi(\cdot) = p(\cdot)$, MCMC mutation keeps $p(\cdot)$ *invariant* because samples are already at the stationary distribution. The outcome of the acceptance of the proposal in our method is thus irrelevant because one can always generate a new sample (state) x that is already distributed according to $p(\cdot)$. We also have multiple proposals y from the same x because x corresponds to the primary sample and y correspond to all the secondary samples that are generated by perfectly stratified sampling of the proposal in Equation (4.4). When the baseline point sampling method is also MCMC (i.e. apply shape splatting on top of MCMC rendering), continue running the chain with accepted states could be beneficial. However, in practice it is more complicated as shift

mappings are usually done in path space whereas MCMC is done in PSS and would require invertible path sampling techniques [10, 45, 46] to transform samples between spaces. Since utilizing shift mappings in difference spaces is not the focus of our work, we also omit the MCMC acceptance in this case.

4.2.3 Efficient Shape Estimator via Debiasing

Our pointwise estimator $P(\mathcal{S}^i)$ Eq. (4.7) can reduce the cost of shape splatting without introducing any additional error. However, the overall splatting cost is nevertheless still $O(|\mathcal{S}^i|)$ since we need to evaluate the contribution of all the samples within the shape. To make our method more efficient, we would like to estimate $C(\mathcal{S}^i)$ as a whole at a *sublinear* cost to $|\mathcal{S}^i|$. We propose to employ *telescoping sum debiasing estimators* [43] to achieve this goal.

Let us focus on a single shape $\mathcal{S}^i$ to drop the superscript i for the rest of the discussion. Since all pixels $j \notin \mathcal{S}$ have zero contribution to the shape, we refer to only the pixels in the shape $j \in \mathcal{S}$ in the following. We also refer to the contribution $C(\mathcal{S})$ of $\mathcal{S}$ as an $\mathbb{R}^{|\mathcal{S}|}$ vector. One can reduce the cost of evaluating $C(\mathcal{S})$ by not sampling secondary samples but by replacing their contributions with the primary sample $P(\mathcal{S})_i$. The outcome of this is an biased estimator, which we refer to as $D(1)$ since it requires only 1 point sample. For a debiasing estimator, we first need to define a sequence of estimators $D(m)$ such that the bias $\mathbf{B}[D(m)] \to 0$ as m approaches a limit [43]. The limit in our case is $|\mathcal{S}|$ which samples all the points within the shape and thus $D(|\mathcal{S}|) = P(\mathcal{S})$.We now need to define $D(m)$ as a series of increasingly finer approximations of the unbiased estimator $P(\mathcal{S})$ such that the bias decreases with increasing m. Let $1 \leq k \leq |\mathcal{S}| - 1$ be a random integer with probability p_k. We sample an ordered subset $\mathcal{K} \subset \mathcal{S}, i \in \mathcal{K}$ of $k+1$ pixels, including the center and the other k pixels. A pointwise estimator $P(\mathcal{K})$ for this subset $\mathcal{K}$ is defined as $P(\mathcal{K})_j = P(\mathcal{S})_j$ for $j \in \mathcal{K}$ and 0 otherwise. Note that this estimator costs only $O(|\mathcal{K}|)$ because there are only $|\mathcal{K}|$ points to be evaluated. To approximate $P(\mathcal{S})$ with $P(\mathcal{K})$, we employ an interpolation operator $A_{\mathcal{S}} : R^k \to R^{|\mathcal{S}|}$ that fills in missing pixels in $P(\mathcal{K})$, thus we a have $A_{\mathcal{S}}(P(\mathcal{K})) = P(\mathcal{S})$ when $\mathcal{K} = \mathcal{S}$. We express $P(\mathcal{S})$ as a telescoping sum [43]

$$P(\mathcal{S}) = D(1) + \sum_{k=1}^{|\mathcal{S}|-1} \Delta D(k) \tag{4.8}$$

where $\Delta D(k) = D(k+1) - D(k)$ and $D(k) = A_{\mathcal{S}}(\mathcal{K})$. The resulting debiasing estimator is

$$\langle C(\mathcal{S}) \rangle = D(1) + \frac{\Delta D(k)}{p_k}. \tag{4.9}$$

Similar to the approach by [43], we correlate estimates for $D(k)$ and $D(k+1)$ by using the same set of pixels. We uniformly sample k *distinct* secondary pixels to obtain a set of $k+1$ pixels $\mathcal{K}$. To reuse samples as much as possible, we loop over all k different ways of removing one *secondary* pixel from $\mathcal{K}$ and estimate $\langle D(k+1) - D(k) \rangle$ for each of the removed pixels as

$$\langle D(k+1) - D(k) \rangle = \frac{1}{k} \sum_{j \in \mathcal{K}, j \neq c} A_{\mathcal{S}}(P(\mathcal{K})) - A_{\mathcal{S}}(P(\mathcal{K} - \{j\})). \tag{4.10}$$

The remaining problem is to find a suitable p_k. Choosing p_k to minimize the $\mathbf{V}[\langle C(\mathcal{S}) \rangle]$ as was done by Misso et al. [43] is not very helpful in our case since we rather want to minimize the work-variance $\mathbf{V}[\langle C(\mathcal{S})_k \rangle]\mathbf{C}[\langle C(\mathcal{S})_k \rangle]$ to account for both the variance and the cost. Minimizing the work-variance directly is challenging as it requires the knowledge of $C(\mathcal{S})$ itself. We, however, empirically found that having $p_k \propto k^{-2}$ produces reasonably low variance through numerical experiments. The average number of point evaluation $\sum_{k=1}^{|\mathcal{S}|} k p_k$ is proportional to partial sum of harmonic series that is known to grow only *logarithmically* to k, achieving a sublinear cost for each estimate of $P(\mathcal{S})$.

$$p_k = \frac{k^{-2}}{\sum_{i=1}^{|\mathcal{S}|} i^{-2}} \tag{4.11}$$

$$\lim_{|\mathcal{S}| \to \infty} p_k = \frac{k^{-2}}{\pi^2/6} \tag{4.12}$$

$$\sum_{k=1}^{|\mathcal{S}|} k p_k = \frac{k^{-1}}{\sum_{i=1}^{|\mathcal{S}|} i^{-2}} \tag{4.13}$$

$$\approx \frac{6}{\pi^2} \sum_{k=1}^{|\mathcal{S}|} k^{-1} \tag{4.14}$$

$$\approx \mathcal{O}(log(|\mathcal{S}|)) \tag{4.15}$$

Efficient Interpolation Operator

While there are many different 1D interpolation operators, we found that simple linear interpolation works well. We experimented with high order polynomial interpolation but

found that they usually increase variance as the interpolated contributions are not bounded by the minimum or maximum of $P(\mathcal{K})$. We thus approximate $P(\mathcal{S})$ by linearly interpolating weighted path contributions evaluated at two neighboring secondary pixels, and compute Eq. (4.6) using the interpolated values. For pixels j beyond the endpoints, we define $(A_{\mathcal{S}})_j$ by using the contribution of nearest endpoint.

A naive implementation of Eq. (4.10) requires evaluating the interpolation operator k times, resulting in a overall $\mathcal{O}(k|\mathcal{S}|)$ complexity. We can speed it up to $\mathcal{O}(|\mathcal{S}|)$ by taking the advantage of that the contribution of each j is only affected by the two neighboring pixels. For each pixel j, we first interpolate it by its neighbors, then we consider the alternative interpolation results by removing one of its neighbors and compute the difference. It only requires keeping track of two nearest neighbors in both directions. The total cost of computing Eq. (4.10) in this manner is $O(|\mathcal{S}|)$.

4.2.4 Generalization to 2D Shapes

Most concepts in previous sections such as shapes, MCMC weights, and debiasing estimators, generalize to 2D without modification. The only exception is the interpolation operator $A_{\mathcal{S}}$. We have to choose a scheme that allows evaluating Eq. (4.10) in linear time to prevent the computation cost exploding at large shapes. We explored many methods for interpolating irregular spatial data such as inverse distance weighting, Voronoi diagrams and k-Nearest Neighbors. However, these methods either lacks linear running time for interpolating all $|\mathcal{S}|$ pixels or have to rebuild data structures $|\mathcal{K}|$ times for computing Eq. (4.10), making the overhead of interpolation outweighs the benefit of shape splatting. We instead take an alternative approach to reduce 2D shapes back to 1D by running a space filling curve throughout $\mathcal{S}$ and order the pixels according to the curve. The secondary pixels are subsequently sampled and sorted in this order. We perform the fast 1D linear interpolation in the previous section on reduced 1D shape. Our experiment shows that our interpolation schemes work well enough to produce visually indistinguishable result compared to more sophisticated method while preserving the linear running time of evaluating Eq. (4.10).

4.2.5 Spatially Varying Shape

Our method can support spatially varying shapes across pixel to further reduces variance in Eq. (4.9). Since the debiasing estimator works best when the difference between base estimator $D(1)$ and point estimator $P(\mathcal{S})$ is small, we would like to ideally include only

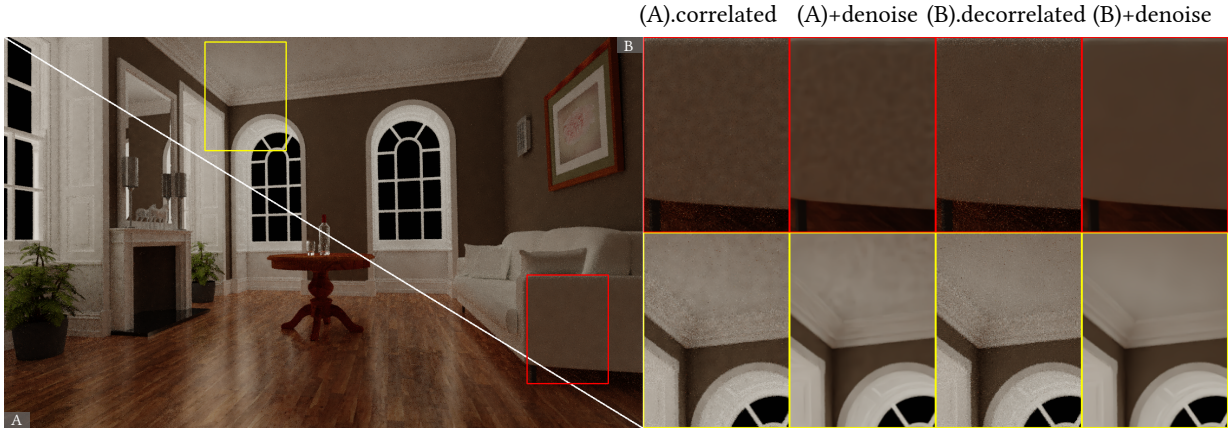


Figure 4.2: We insert strides and randomly rotating the shape to greatly reduce low frequency artifacts and correlation between neighbor pixels. The decorrelated image can be further denoised by a machine learning denoiser without introducing outstanding artifacts.

those pixels that have similar contributions. We estimate pixel similarity based on auxiliary features such as albedo and geometry normals, similar to feature-preserving denoiser kernels [13, 48]. To construct such shapes, we first assign a spatially uniform *base shape* $\mathcal{S}_B$ to every pixel in the image. We redefine $\mathcal{S}^i$ at each pixel i by selecting pixels $j \in \mathcal{S}^i_B$ such that

$$n_i \cdot n_j \geq \tau_n \wedge ||c_i - c_j||_\infty \leq \tau_c, \tag{4.16}$$

where n_i and c_i are the geometry normal and albedo at i. We empirically set the thresholds for the differences τ_n and τ_c as $\tau_n = \sqrt{2}/2, \tau_c = 0.1$. Note that the shape Eq. (4.16) is deliberately constructed to be symmetric; if $j \in \mathcal{S}^i$, we also have $i \in \mathcal{S}^j$. It is because our shape splatting formulates secondary samples as MCMC mutations, and symmetric shapes ensure that all the secondary samples receive non-zero weights.

4.2.6 Decorrelating Shape Splats

Shape splatting results in lower numerical error by increasing the effective number of samples per pixel. However, it often introduces lower frequency noise due to inter-pixel correlation as also seen in other path-reusing methods. We propose a simple technique to address this issue. We first inserts strides between pixels in the shape, shifting the noise to higher frequencies. We additionally randomly rotate the shapes in each sample, further

decorrelating shape splats. This approach generally makes the correlation less perceptible in the image space, but shifting mapping to such randomized points increases per-pixel error by $\approx 10 \sim 15\%$, with large strides resulting in higher error but less correlation. One benefit is that a decorrelated image can usually be denoised effectively by a neural network denoiser such as OpenImageDenoise [63] in Fig. 4.2. Due to the flexibility of shape splatting and that we have direct control over the splatting location, it might be possible to combine our framework with blue-noise dithered sampling [15] to achieve a more desirable error distribution over the image.

4.2.7 Optional Mixing with Point Splats

Our debiasing estimator is not free from occasional artifacts or even negative-valued pixels when the base estimator differs significantly from the point estimator, due to outliers in the path contribution that cannot be easily detected via auxiliary features in Sec. 4.2.5. For example, when the primary sample is a high energy outlier, its overly high contribution is likely to spread to neighbor pixels to produce a group of outlier pixels, especially if there are not enough secondary samples to correct the bias. The artifact can also be produced if one of the secondary sample is an outlier.

We empirically found this issue is primarily relevant to shape splatting combined with vanilla path tracers, although shape splatting with other methods also benefit from point mixing. We alleviate this issue by combining the already-available point samples at each center pixel and secondary samples as they form two unbiased estimators that are free of such artifacts. Since these three estimators are correlated and uses samples draw from the *same* density, we cannot use MIS weights to weight the estimators. Instead, we compute per-pixel variance estimates for each estimator and weight them according to their inverse variance. Unfortunately, it introduces small amount of bias manifests as energy loss around high variance regions due to the variance estimates being correlated with the estimators. Nevertheless, we found the trade-off to be worthwhile as it is generally effective in removing artifacts. Investigating an unbiased mixing scheme could be an interesting future work.

4.2.8 Alternative Weighting via Pairwise MIS

We include a discussion about shape weighting via pairwise MIS [9, 38] for completeness. Pairwise MIS approximates balanced heuristic by only considering a pair of path. We first discuss the gathering form of pairwise MIS that is used in ReSTIR, then we would introduce the scattering form that is more relevant to shape splatting.

Gathering Form

Consider path reusing around pixel i. For each pixel j in an M pixel neighborhood, a path $\mathbf{y}_j$ sampled at j and shift mapped to pixel i, $\mathbf{x}_j = T_{j\to i}(\mathbf{y}_j)$. The sample $\mathbf{x}_i$ is referred to as the *canonical sample*. We use the shorthand $p_{\leftarrow j}(\mathbf{x}) = p(T_{i\to j}(\mathbf{x}))\left|\partial_{\mathbf{x}} T_{i\to j}\right|$ to denote the path density by first sampling a path at j and shift map to i. Consider only $p_{\leftarrow i}$ and $p_{\leftarrow j}$ as two sampling techniques, the balanced heuristics weight for the canonical sample is $w_i(\mathbf{x}) = \dfrac{p_{\leftarrow i}}{p_{\leftarrow i} + p_{\leftarrow j}}$, whereas $w_j(\mathbf{x}) = \dfrac{p_{\leftarrow j}}{p_{\leftarrow i} + p_{\leftarrow j}}$. Naively summing the weights for all pixel j that shift mapping to i would produce a valid MIS weight, however it would result in the canonical sample to accumulate a very large weight since it appears $M-1$ more times.

$$w_j(\mathbf{x}) = \frac{p_{\leftarrow j}}{p_{\leftarrow i} + p_{\leftarrow j}}, j \neq i \tag{4.17}$$

$$w_i(\mathbf{x}) = \sum_{i\neq k} \frac{p_{\leftarrow i}}{p_{\leftarrow i} + p_{\leftarrow k}} \tag{4.18}$$

Bitterli [9] mitigate this issue by downweightng the canonical sample by $M-1$, resulting in the following pairwise MIS weight which produce uniform weight when all $p_{\leftarrow j}$ are equal.

$$w_j(\mathbf{x}) = \frac{1}{M-1}\frac{(M-1)p_{\leftarrow j}}{p_{\leftarrow i} + (M-1)p_{\leftarrow j}}, j \neq i \tag{4.19}$$

$$w_i(\mathbf{x}) = \frac{1}{M-1}\sum_{i\neq k} \frac{p_{\leftarrow i}}{p_{\leftarrow i} + (M-1)p_{\leftarrow k}} \tag{4.20}$$

In practice, a slight modified "defensive" variant of the above weight is used to increase robustness of sampling:

$$w_j(\mathbf{x}) = \frac{1}{M}\frac{(M-1)p_{\leftarrow j}}{p_{\leftarrow i} + (M-1)p_{\leftarrow j}}, j \neq i \tag{4.21}$$

$$w_i(\mathbf{x}) = \frac{1}{M} + \frac{1}{M}\sum_{i\neq k} \frac{p_{\leftarrow i}}{p_{\leftarrow i} + (M-1)p_{\leftarrow k}}, \tag{4.22}$$

by giving the canonical sample slightly more weight.

Scattering Form

A fundamental difference in path reusing between our method and ReSTIR is that our method employs the scattering view instead of gathering. While both interpretations are

equivalent, there are some key differences in the implementation and thus we provide the scattering form used in shape splatting for completeness. In the scattering form, since we only have access to the canonical sample $\mathbf{x}_i$ at the current primary pixel, we only compute part of "complete" pairwise MIS weight during splatting. The remaining part is computed when samples from other pixels are splatted into i. As a result, the contributions is weighted by $\frac{1}{|\mathcal{S}^j|}$ or $\frac{1}{|\mathcal{S}^i|}$ depending on whether it is the primary pixel, in contrast to the constant $\frac{1}{M}$ weight in gathering form.

$$w_{i\to j} = \begin{cases} \dfrac{1}{|\mathcal{S}^j|} \dfrac{(|\mathcal{S}^j|-1)p(\mathbf{x}_j)/\left|\partial_{\mathbf{x}} T_{i\to j}\right|}{p(\mathbf{x}_i) + (|\mathcal{S}^j|-1)p(\mathbf{x}_j)/\left|\partial_{\mathbf{x}} T_{i\to j}\right|} & (i \neq j) \\ \dfrac{1}{|\mathcal{S}^i|}\left(1 + \displaystyle\sum_{k\neq i} \dfrac{p(\mathbf{x}_i)}{p(\mathbf{x}_i) + (|\mathcal{S}^i|-1)p(\mathbf{x}_k)\left|\partial_{\mathbf{x}} T_{i\to k}\right|}\right) & (i = j). \end{cases} \tag{4.23}$$

The main issue of pairwise MIS we found is that the weights do not vary smoothly across the shape compared to MCMC weights, resulting a higher variance in our debiasing shape estimator.

Chapter 5

Results

5.1 Implementation

We evaluate our method in a CPU/GPU rendering framework powered by LuisaCompute [61]. We implemented our method on top of a mega-kernel path tracer, unidirectional PSSMLT [31], and offline ReSTIR [38]. Our method can be implemented easily on top of existing CPU/GPU path tracers without major modification to the underlying baseline method, especially if the rendering system already implemented hybrid shift mapping, which is also required for ReSTIR.

We also propose an efficient implementation of our method on a GPU. Since each pixel uses a potentially different number of debiasing samples, naively implementing our method results in poor performance due to thread divergence. Our current solution is to implement shape splatting in a wavefront manner while keeping the mega-kernel path tracer: we launch separate kernels to sample primary paths, the secondary paths, and compute telescoping sum to guarantees that all threads have equal work loads. Our program spends around 50% of time tracing primary paths, 40% on secondary paths, and 10% on computing the telescoping sum and splatting contribution.

We implemented hybrid shift mapping [38] due to its simplicity and effectiveness. We evaluate the Jacobian and sample distribution $p(\cdot)$ under primary sample space parameterization. The optimal shape size depends on the image resolution as well as shift mapping efficiency. Our method works well for shape sizes ranging from $20 \sim 80$ pixels. Using shapes larger than 80 pixels are possible but subjected to increased memory overhead during splatting. Finding the optimal shape for a given scene and image resolution is left for future work.

5.2 Numerical Results

We measure the performance of our shape splatting on top of PT and offline ReSTIR PT on an NVIDIA RTX 3070Ti GPU, where we additionally compared with Bekaert-style path reusing (BPR). For ReSTIR PT, we sample 32 initial candidates and perform 3 rounds of spatial reuse with 6 neighbors each round in a 10 pixel radius. We offered two variants for our method combined with ReSTIR: the RIS PT+Ours variant replace all 3 spatial reuse rounds with 8 rounds of shape splatting using randomized 61 pixel shapes; the ReSTIR PT+Ours variant replace the last spatial reuse in ReSTIR with 8 rounds of shape splatting. Shape splatting with PSSMLT are measured on an Intel i9-13900K 24 core processor. We tested our method on a variety of scenes in equal time settings. Fig. 5.5, Fig. 5.6 and Fig. 5.7 show some closeups.

Since offline ReSTIR PT is mainly for computing indirect lighting, we compare all methods on *indirect lighting only*. However, our method is also able to handle direct lighting efficiently. We show error plots of our method against their point splatting baselines in MAPE metrics in Fig. 5.3 and Fig. 5.4.

Results show that our method consistently outperforms point splatting in all test scenes under equal time. Our debiasing and secondary sample generation by shift mapping allows us to reduce an actual cost of shape splatting equivalent to *only two point samples* on average. With the help of spatially varying kernels, our shape splatting can adapt to textured materials and sharp geometry edges. Fig. 5.3 demonstrates how our shape splatting can accelerate a vanilla path tracer and existing path reusing methods such as ReSTIR. The improvements on top of the path reusing methods show that our shape splatting is not simply yet another path reuse method. For example, even RIS PT+Ours is already almost as effective as ReSTIR on many scenes. RIS/ReSTIR PT+Ours is also free from color noises that usually present in ReSTIR PT (Fig. 5.5). Our decorrelation approach is generally effective at reducing artifacts except for very challenging scenes or when the base estimator itself exhibits strong inter-pixels correlation. (e.g. DIFFICULT LIVING ROOM scene in Fig. 5.6)

5.3 Ablation Study

We conduct an ablation study to investigate the effectiveness of various components of our method in Fig. 5.1. We focus on analyzing shape splatting with PT in this section.

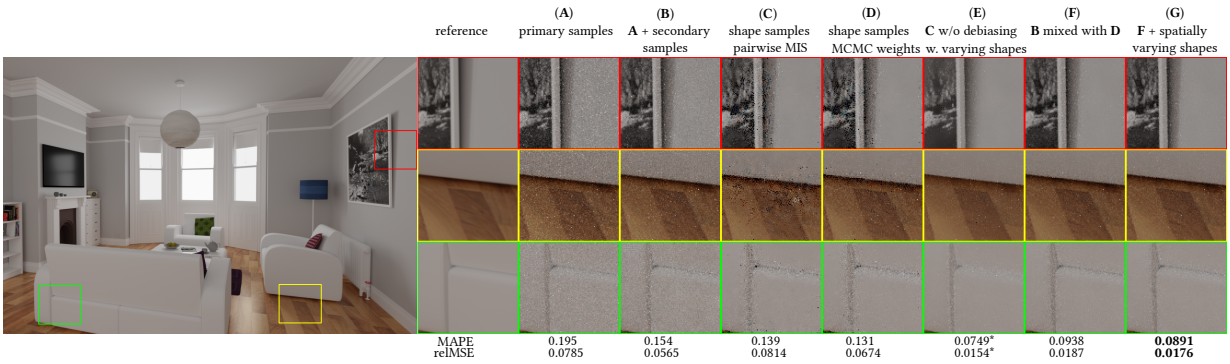


Figure 5.1: We demonstrate the performance of different components of our method. Note while **E** has lower error, it is inconsistent and has blurring artifact.

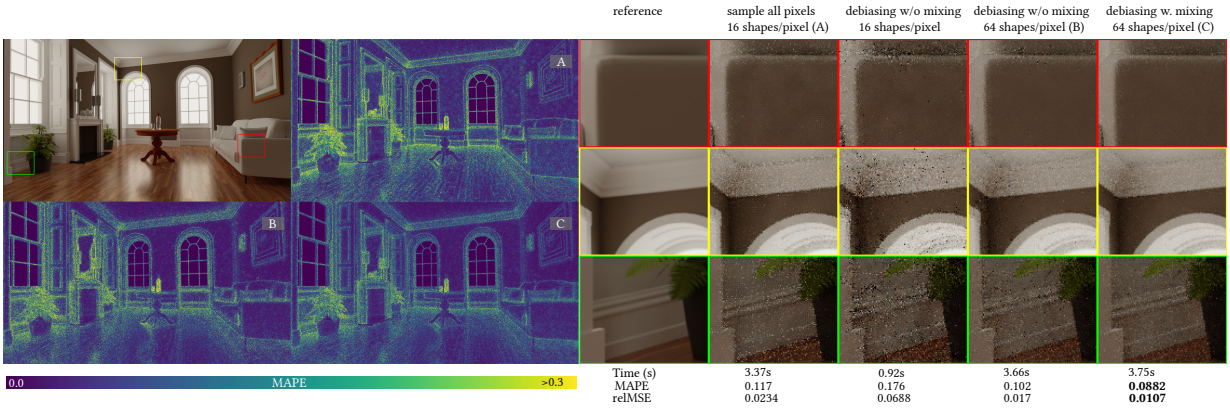


Figure 5.2: Our debiasing estimator outperforms naive shape splatting, mixing with point samples improves efficiency further by removing outliers.

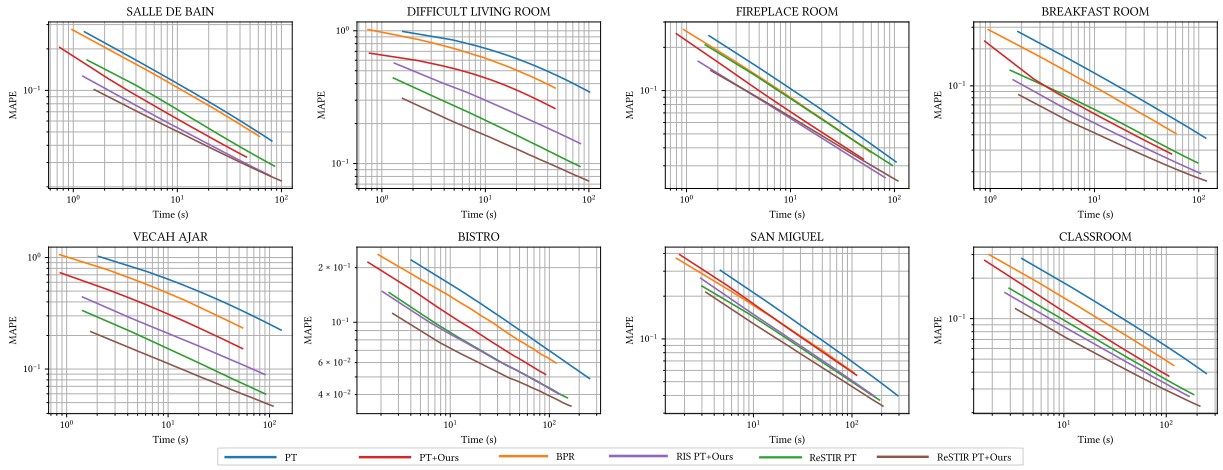


Figure 5.3: Error plots of shape splatting vs point splatting baselines on GPU.

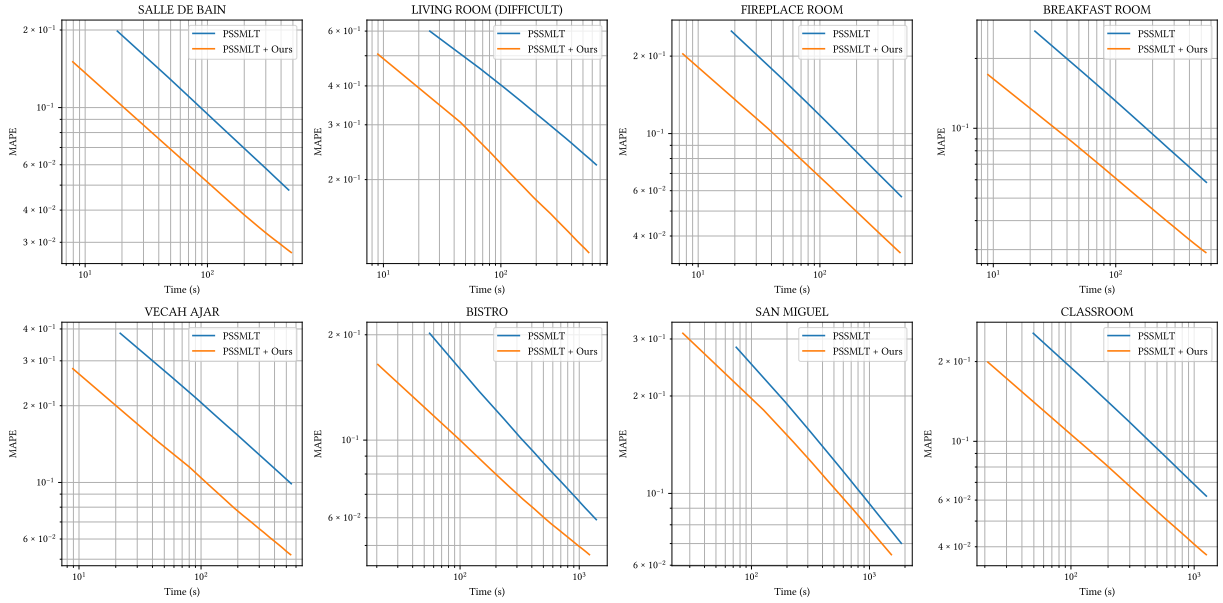


Figure 5.4: Error plots of shape vs point splatting on top of PSSMLT

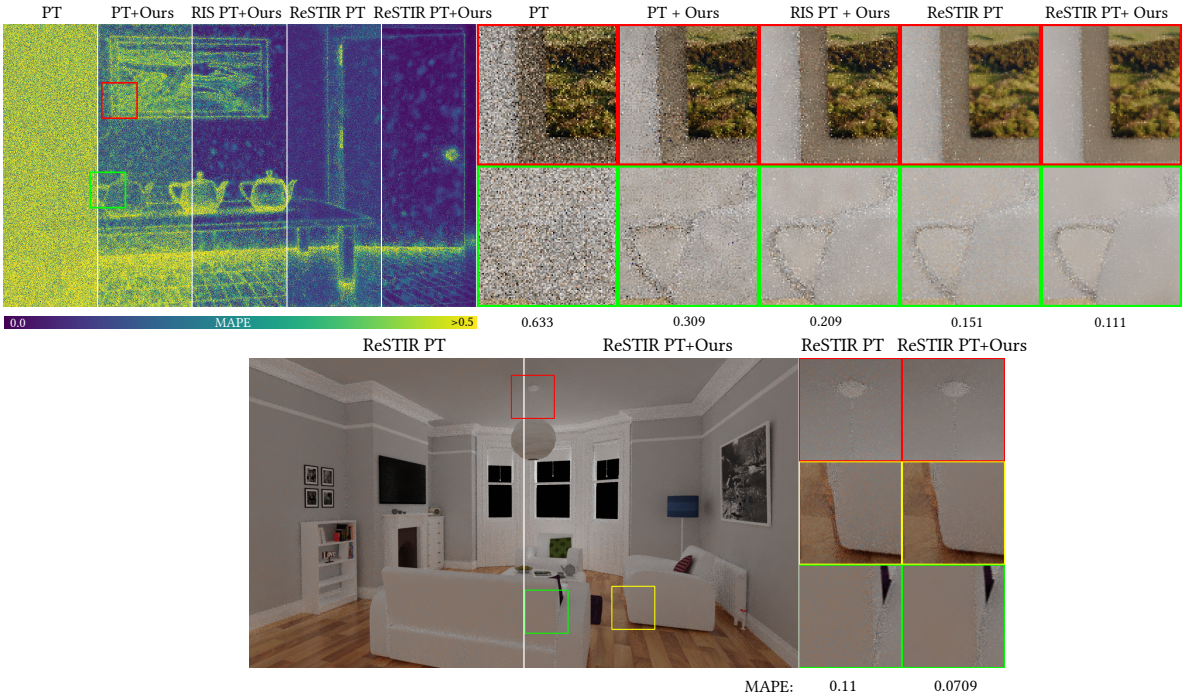


Figure 5.5: Top: On a challenging scene, our method consistently out performs its point splatting baseline. Bottom: Our method suppresses color noise when integrated with ReSTIR.

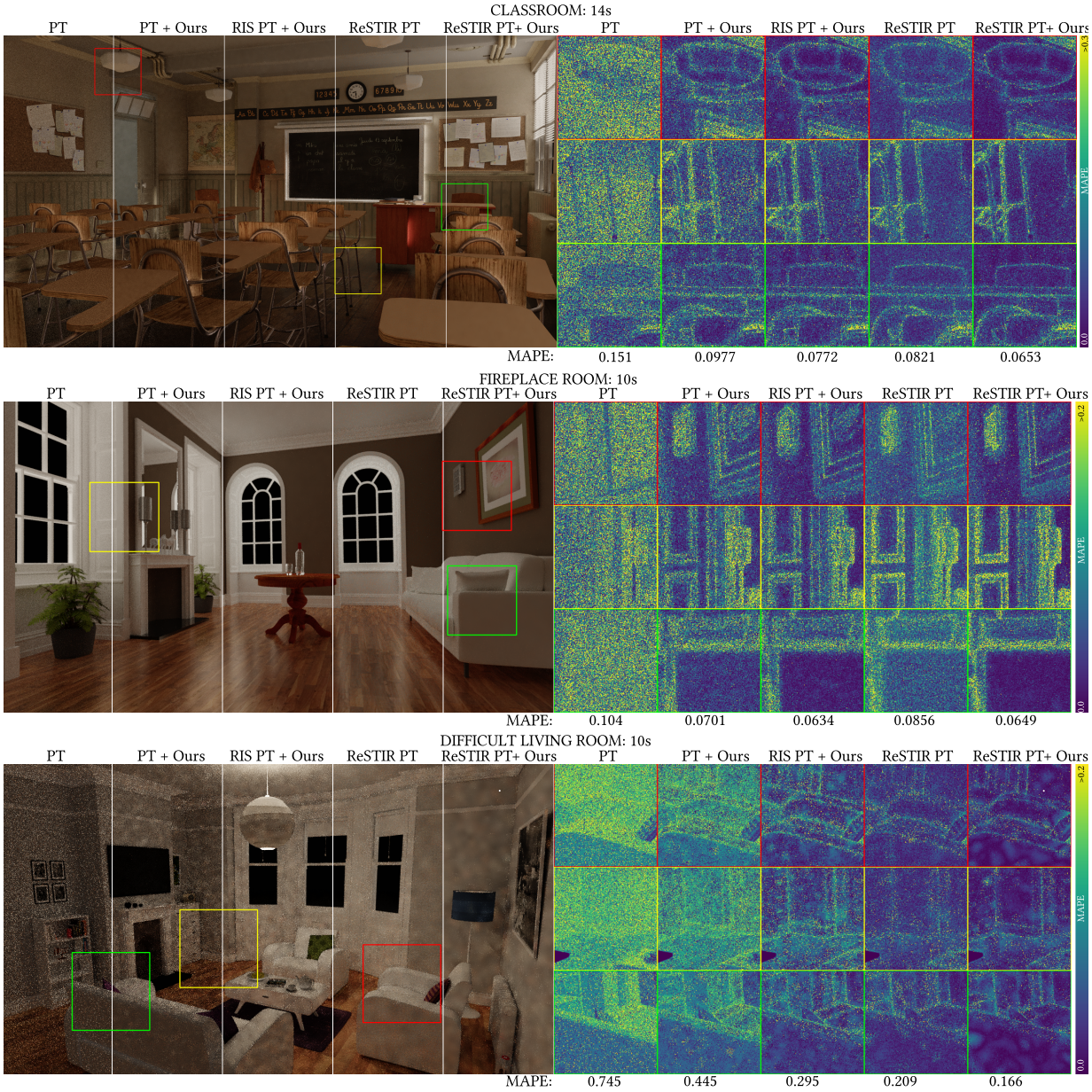


Figure 5.6: We integrated our method on top of PT and ReSTIR PT on GPU. Our method consistently outperforms point splatting in a variety of scenes.

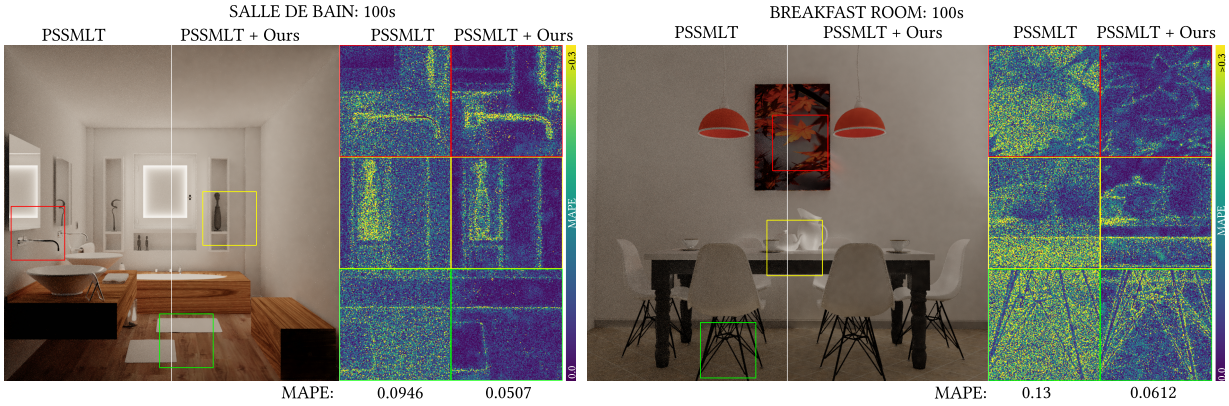


Figure 5.7: We integrated our method on top of PSSMLT on CPU. Our method consistently outperforms point splatting in a variety of scenes.

Mixing with point samples

Fig. 5.1 col. **B** is a result of using only primary and secondary *point* samples during shape splatting. It is quite noisy due to its point splatting nature. Col **D**. instead uses only shape samples. While shape splatting reduces noise, it produces outliers around texture discontinuities and geometry edges. Mixing points and shapes (col **F**) using weighted inverse variance removes outliers effectively while preserving the low variance of the shape estimator. Fig. 5.8 visualizes mixing weights on a few scenes. Shape splatting receives larger weights in most cases. In regions where high frequency geometry dominates, our method offer less improvement over point splatting as the contribution of shape splatting is negated by the variance from debiasing.

Effect of debiasing

To demonstrate that debiasing is indeed necessary, in col.**E** Fig. 5.1, we only compute the telescoping sum up to $k = 3$ in Eq. (4.9) so that the bias is not completely removed. While it has lower error at low sample count, it is neither unbiased nor consistent. Notice the obvious blurring of texture and geometry, even when we already use spatially varying shapes described in Sec. 4.2.5 to group similar pixels.

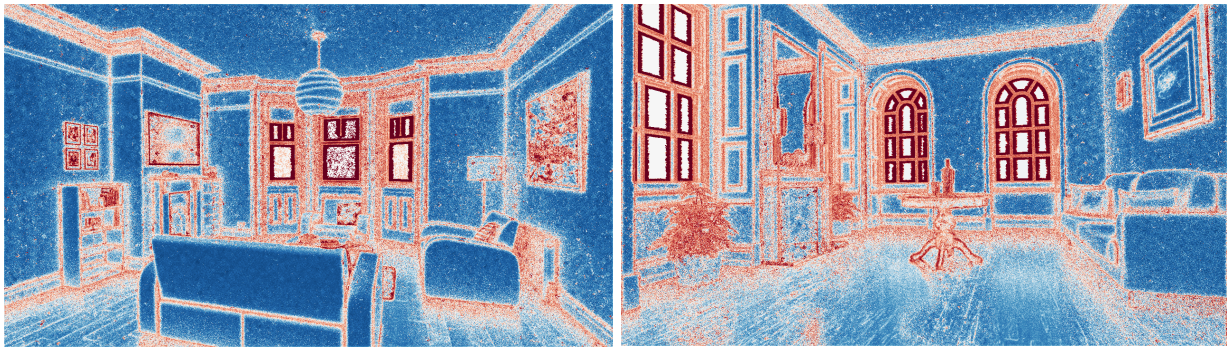


Figure 5.8: Visualizing the weights when mixing with point samples. Blue corresponds to a higher weight for shape splatting than points. Our method works best when the geometry and lighting is smooth, but it can also handle non-smooth shading such as image textures.

Comparison between pairwise MIS and MCMC weights

To evaluate MCMC weights in Eq. (4.6), we compared them against pairwise MIS [9] in col. **C** of Fig. 5.1. We found that MIS weights tend to produce higher variance and more outliers than our MCMC weights.

Comparison of PMFs for Debiasing

We have tested different choices of the PMF in Eq. (4.9). If we ignore the cost of the estimator and chose the PMF according to variance only, $p_k = k^{-1}$ has lowest variance among them. However, once we consider the overall efficiency, our choice of $p_k = k^{-2}$ performs the best.

Debiased and naive shape splatting

Our shape estimator introduces additional noise due to debiasing. This raises the question of whether the advantages of splatting shapes outweigh this noise. In Fig. 5.10, we compared against naive shape splatting where all pixels in the shape is estimated by point sampling from scratch. Our debiasing estimator outperforms the naive approach even without mixing. Once we mix with point samples, our debiasing estimators significantly outperforms naive shape splatting. To provide further insights, we provide closeups in Fig. 5.2. Under equal shape sample, our debiasing estimator performs equally well as

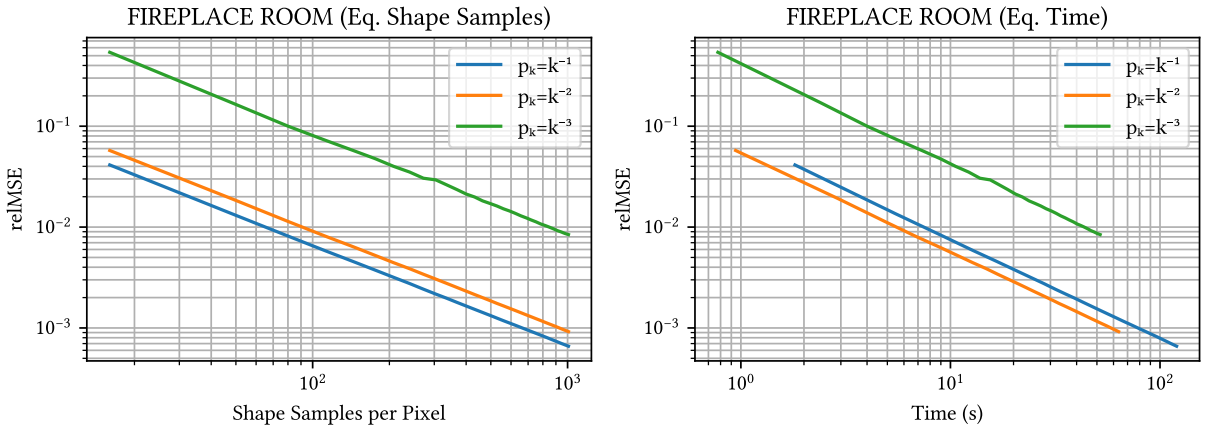


Figure 5.9: Compared different PMF for debiasing. While $p_k = k^{-1}$ has better equal sample (left) performance, under equal time (right) its larger sampling cost makes it less competitive than our choice of $p_k = k^{-2}$.

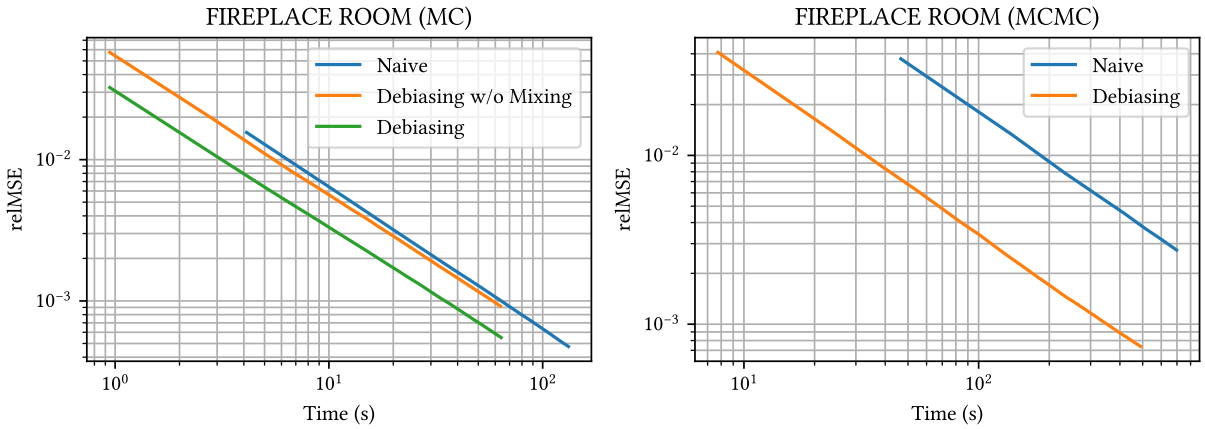


Figure 5.10: We compared our debaising estimator with naive shape splatting under equal time. In MCMC Rendering (right), our debiasing estimator outperforms naive shape splatting by a large margin. In MC rendering with path tracer, while debiasing without mixing with point samples produces higher relMSE due to the present of outliers, mixing with point samples effectively suppresses outliers and has lower variance than naive shape splatting.

naive splatting in smooth regions but with significantly lower computational cost, while naive splatting exhibits lower variance around geometry discontinuities. In equal time comparisons our debiasing estimator already outperforms in smooth region due to reduced correlation artifacts due to higher sample count, except for the presence of outliers. Mixing with point samples effectively eliminates these outliers and results in a lower mean image error.

Chapter 6

Conclusion

We proposed a framework for efficient shape splatting in image space by leveraging MCMC mutations and debiasing. While our framework can improve various point-sample counterparts already, it has a few aspects that can be further studied.

Firstly, shape splatting may not work efficiently if the underlying rendering algorithm has a large variance (i.e., firefly noise). In this case, a sampled shape might splat outlier samples to other pixels in the shape, resulting in a visual artifact. Mixing with point samples helps but may result in energy loss due to the biased nature of this mixing. Utilizing neural methods [2, 3, 21] for mixing point splats and shape splats could be helpful.

Secondly, our interpolation scheme only interpolates weighted path contribution, but it potentially enables a wide range of interpolation schemes. For example, one could decompose path contribution into BSDF and irradiance terms and interpolate them separately. Investigating a better interpolation scheme could improve performance in the presence of high frequency geometries and textures.

Lastly, applications of our framework to any other problems involving many correlated integrals would be interesting to explore. For example, integrating our framework with Monte Carlo estimators of partial differential equations [49, 54, 47] is likely fruitful because one would usually want to run such an estimator for consecutive points in space just like pixels in images.

References

[1] Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. Stochastic rasterization using time-continuous triangles. In Mark Segal and Timo Aila, editors, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware 2007, San Diego, California, USA, August 4-5, 2007*, pages 7–16. Eurographics Association, 2007.

[2] Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. Deep combiner for independent and correlated pixel estimates. *ACM Trans. Graph.*, 39(6):242:1–242:12, 2020.

[3] Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. Self-supervised post-correction for monte carlo denoising. In Munkhtsetseg Nandigjav, Niloy J. Mitra, and Aaron Hertzmann, editors, *SIGGRAPH '22: Special Interest Group on Computer Graphics and Interactive Techniques Conference, Vancouver, BC, Canada, August 7 - 11, 2022*, pages 18:1–18:8. ACM, 2022.

[4] Rasmus Barringer, Carl Johan Gribel, and Tomas Akenine-Möller. High-quality curve rendering using line sampled visibility. *ACM Trans. Graph.*, 31(6):162:1–162:10, 2012.

[5] Pablo Bauszat, Victor Petitjean, and Elmar Eisemann. Gradient-domain path reusing. *ACM Trans. Graph.*, 36(6):229:1–229:9, 2017.

[6] Philippe Bekaert, Mateu Sbert, and John H. Halton. Accelerating path tracing by re-using paths. In Simon Gibson and Paul E. Debevec, editors, *Proceedings of the 13th Eurographics Workshop on Rendering Techniques, Pisa, Italy, June 26-28, 2002*, volume 28 of *ACM International Conference Proceeding Series*, pages 125–134. Eurographics Association, 2002.

[7] Espen Bernton, Shihao Yang, Yang Chen, Neil Shephard, and Jun S. Liu. Locally weighted markov chain monte carlo. *arXiv: Computation*, 2015.

[8] Niels Billen and Philip Dutré. Line sampling for direct illumination. *Comput. Graph. Forum*, 35(4):45–55, 2016.

[9] Benedikt Bitterli. *Correlations and Reuse for Fast and Accurate Physically Based Light Transport*. PhD thesis, Dartmouth College, 2021.

[10] Benedikt Bitterli, Wenzel Jakob, Jan Novák, and Wojciech Jarosz. Reversible jump metropolis light transport using inverse mappings. *ACM Trans. Graph.*, 37(1):1, 2018.

[11] Benedikt Bitterli and Wojciech Jarosz. Beyond points and beams: higher-dimensional photon samples for volumetric light transport. *ACM Trans. Graph.*, 36(4):112:1–112:12, 2017.

[12] David Cline and Parris Egbert. A practical introduction to metropolis light transport. *Brigham Young University*, 2005.

[13] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In Justin Hensley, Philipp Slusallek, David K. McAllister, and Christiaan P. Gribble, editors, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics 2010, Saarbrücken, Germany, June 25-27, 2010*, pages 67–75. Eurographics Association, 2010.

[14] Xi Deng, Shaojie Jiao, Benedikt Bitterli, and Wojciech Jarosz. Photon surfaces for robust, unbiased volumetric density estimation. *ACM Trans. Graph.*, 38(4):46:1–46:12, 2019.

[15] Iliyan Georgiev and Marcos Fajardo. Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, pages 1–1. 2016.

[16] Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. Sample-based monte carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[17] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[18] Carl Johan Gribel, Rasmus Barringer, and Tomas Akenine-Möller. High-quality spatio-temporal rendering using semi-analytical visibility. *ACM Trans. Graph.*, 30(4):54, 2011.

[19] Carl Johan Gribel, Michael C. Doggett, and Tomas Akenine-Möller. Analytical motion blur rasterization with compression. In Justin Hensley, Philipp Slusallek, David K. McAllister, and Christiaan P. Gribble, editors, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics 2010, Saarbrücken, Germany, June 25-27, 2010*, pages 163–172. Eurographics Association, 2010.

[20] Adrien Gruson, Binh-Son Hua, Nicolas Vibert, Derek Nowrouzezahrai, and Toshiya Hachisuka. Gradient-domain volumetric photon density estimation. *ACM Trans. Graph.*, 37(4):82, 2018.

[21] Jeongmin Gu, José Antonio Iglesias Guitián, and Bochang Moon. Neural james-stein combiner for unbiased and biased renderings. *ACM Trans. Graph.*, 41(6):262:1–262:14, 2022.

[22] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):130, 2008.

[23] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[24] Wenzel Jakob and Steve Marschner. Manifold exploration: a markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Trans. Graph.*, 31(4):58:1–58:13, 2012.

[25] Wojciech Jarosz, Derek Nowrouzezahrai, Iman Sadeghi, and Henrik Wann Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Trans. Graph.*, 30(1):5:1–5:19, 2011.

[26] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike J. Sloan, and Matthias Zwicker. Progressive photon beams. *ACM Trans. Graph.*, 30(6):181, 2011.

[27] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. *Comput. Graph. Forum*, 27(2):557–566, 2008.

[28] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In Steve Cunningham, Walt Bransford, and Michael F. Cohen, editors, *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, pages 311–320. ACM, 1998.

[29] Thouis R. Jones and Ronald N. Perry. Antialiasing with line samples. In Bernard Peroche and Holly E. Rushmeier, editors, *Proceedings of the Eurographics Workshop on Rendering Techniques 2000, Brno, Czech Republic, June 26-28, 2000*, Eurographics, pages 197–206. Springer, 2000.

[30] James T. Kajiya. The rendering equation. In David C. Evans and Russell J. Athay, editors, *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986, Dallas, Texas, USA, August 18-22, 1986*, pages 143–150. ACM, 1986.

[31] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the metropolis light transport algorithm. *Comput. Graph. Forum*, 21(3):531–540, 2002.

[32] Markus Kettunen, Eugene d'Eon, Jacopo Pantaleoni, and Jan Novák. An unbiased ray-marching transmittance estimator. *ACM Trans. Graph.*, 40(4):137:1–137:20, 2021.

[33] Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain path tracing. *ACM Trans. Graph.*, 34(4):123:1–123:13, 2015.

[34] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[35] Jaroslav Křivánek, Iliyan Georgiev, Toshiya Hachisuka, Petr Vévoda, Martin Sik, Derek Nowrouzezahrai, and Wojciech Jarosz. Unifying points, beams, and paths in volumetric light transport simulation. *ACM Trans. Graph.*, 33(4):103:1–103:13, 2014.

[36] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[37] Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. Gradient-domain metropolis light transport. *ACM Trans. Graph.*, 32(4):95:1–95:12, 2013.

[38] Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. Generalized resampled importance sampling: foundations of restir. *ACM Trans. Graph.*, 41(4):75:1–75:23, 2022.

[39] Yang Liu, Shaojie Jiao, and Wojciech Jarosz. Temporally sliced photon primitives for time-of-flight rendering. *Comput. Graph. Forum*, 41(4):29–40, 2022.

[40] Marco Manzi, Markus Kettunen, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. Gradient-domain bidirectional path tracing. In Jaakko Lehtinen and Derek Nowrouzezahrai, editors, *26th Eurographics Symposium on Rendering, Rendering - Experimental Ideas & Implementations, EGSR 2015, EI&I Track, Darmstadt, Germany, June 23-26, 2015*, pages 65–74. Eurographics Association, 2015.

[41] Marco Manzi, Fabrice Rousselle, Markus Kettunen, Jaakko Lehtinen, and Matthias Zwicker. Improved sampling for gradient-domain metropolis light transport. *ACM Trans. Graph.*, 33(6):178:1–178:12, 2014.

[42] Luca Martino, David Luengo, and Joaquín Míguez. *Independent random sampling methods*. Springer, 2018.

[43] Zackary Misso, Benedikt Bitterli, Iliyan Georgiev, and Wojciech Jarosz. Unbiased and consistent rendering using biased estimators. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4), July 2022.

[44] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Virtual ray lights for rendering scenes with participating media. *ACM Trans. Graph.*, 31(4):60:1–60:11, 2012.

[45] Hisanari Otsu, Anton S. Kaplanyan, Johannes Hanika, Carsten Dachsbacher, and Toshiya Hachisuka. Fusing state spaces for markov chain monte carlo rendering. *ACM Trans. Graph.*, 36(4):74:1–74:10, 2017.

[46] Jacopo Pantaleoni. Charted metropolis light transport. *ACM Trans. Graph.*, 36(4):75:1–75:14, 2017.

[47] Damien Rioux-Lavoie, Ryusuke Sugimoto, Tümay Özdemir, Naoharu H. Shimada, Christopher Batty, Derek Nowrouzezahrai, and Toshiya Hachisuka. A monte carlo method for fluid simulation. *ACM Trans. Graph.*, 41(6):240:1–240:16, 2022.

[48] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature and color information. *Comput. Graph. Forum*, 32(7):121–130, 2013.

[49] Rohan Sawhney and Keenan Crane. Monte carlo geometry processing: a grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), aug 2020.

[50] Rohan Sawhney, Daqi Lin, Markus Kettunen, Benedikt Bitterli, Ravi Ramamoorthi, Chris Wyman, and Matt Pharr. Decorrelating restir samplers via mcmc mutations. *ACM Trans. Graph.*, oct 2023. Just Accepted.

[51] Benjamin Segovia, Jean-Claude Iehl, and Bernard Péroche. Coherent metropolis light transport with multiple-try mutations. *LIRIS UMR*, 5205, 2007.

[52] Peter Shirley and Chris Wyman. Generating stratified random lines in a square. *Journal of Computer Graphics Techniques (JCGT)*, 6(2):49–54, June 2017.

[53] Gurprit Singh, Bailey Miller, and Wojciech Jarosz. Variance and convergence analysis of monte carlo line and segment sampling. *Comput. Graph. Forum*, 36(4):79–89, 2017.

[54] Ryusuke Sugimoto, Terry Chen, Yiti Jiang, Christopher Batty, and Toshiya Hachisuka. A practical walk-on-boundary method for boundary value problems. *ACM Trans. Graph.*, 42(4), jul 2023.

[55] Weilun Sun, Xin Sun, Nathan A. Carr, Derek Nowrouzezahrai, and Ravi Ramamoorthi. Gradient-domain vertex connection and merging. In Matthias Zwicker and Pedro V. Sander, editors, *28th Eurographics Symposium on Rendering, Rendering - Experimental Ideas & Implementations, EGSR 2017, EI&I Track, Helsinki, Finland, 19-21 June 2017*, pages 83–92. Eurographics Association, 2017.

[56] Xin Sun, Kun Zhou, Jie Guo, Guofu Xie, Jingui Pan, Wencheng Wang, and Baining Guo. Line segment sampling with blue-noise properties. *ACM Trans. Graph.*, 32(4):127:1–127:14, 2013.

[57] Stanley Tzeng, Anjul Patney, Andrew A. Davidson, Mohamed S. Ebeida, Scott A. Mitchell, and John D. Owens. High-quality parallel depth-of-field using line samples. In Carsten Dachsbacher, Jacob Munkberg, and Jacopo Pantaleoni, editors, *Proceedings of the EUROGRAPHICS Conference on High Performance Graphics 2012, Paris, France, June 25-27, 2012*, pages 23–31. Eurographics Association, 2012.

[58] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. PhD thesis, Stanford University, USA, 1997.

[59] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In Susan G. Mair and Robert Cook, editors, *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*, pages 419–428. ACM, 1995.

[60] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal, editors, *Proceedings of the 24th Annual*

Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997, pages 65–76. ACM, 1997.

[61] Shaokun Zheng, Zhiqian Zhou, Xin Chen, Difei Yan, Chuyan Zhang, Yuefeng Geng, Yan Gu, and Kun Xu. Luisarender: A high-performance rendering framework with layered and unified interfaces on stream architectures. *ACM Trans. Graph.*, 41(6):232:1–232:19, 2022.

[62] Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. Vectorization for fast, analytic, and differentiable visibility. *ACM Trans. Graph.*, 40(3):27:1–27:21, 2021.

[63] Attila T Áfra. Openimagedenoise, 2019.