Online Neural Path Guiding with Normalized Anisotropic Spherical Gaussians

JIAWEI HUANG, Chuzhou University, Chuzhou, China and Void Dimensions, China
AKITO IIZUKA, Tohoku University, Sendai, Japan
HAJIME TANAKA, Tohoku University, Sendai, Japan
TAKU KOMURA, The University of Hong Kong, Hong Kong and Tohoku University, Sendai, Japan
YOSHIFUMI KITAMURA, Tohoku University, Sendai, Japan

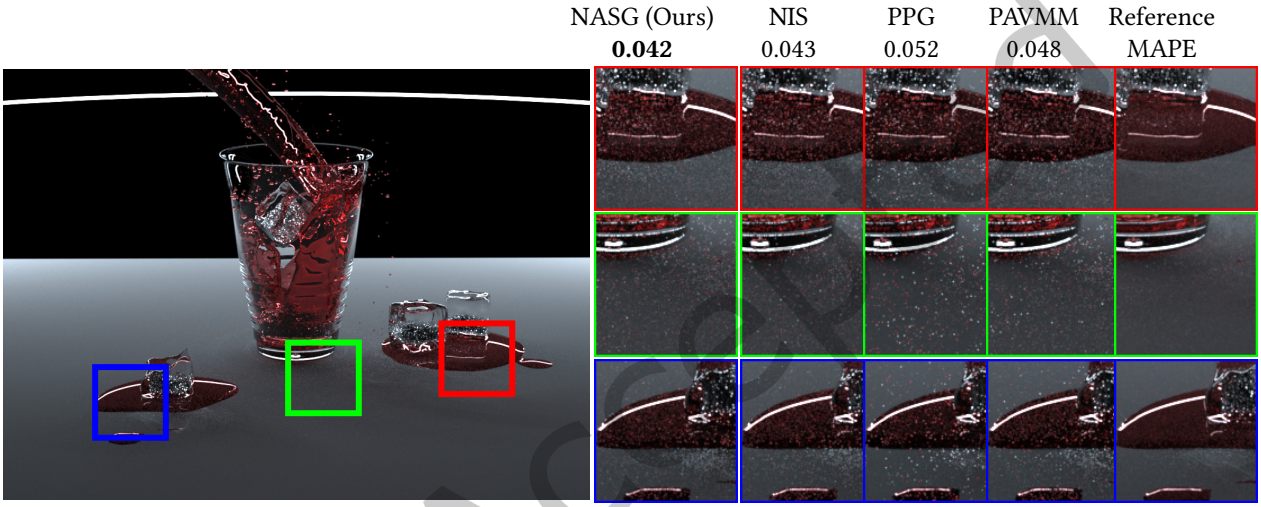

Fig. 1. AJAR scene rendered using proposed online path guiding framework and previous path guiding techniques, including Neural Importance Sampling (NIS) [Müller et al. 2019], Practical Path Guiding (PPG) [Müller 2019], and the Robust Fitting of Parallax-Aware Mixtures for Path Guiding (PAVMM) [Ruppert et al. 2020], with the same 512 sample budget. Our framework uses a single multilayer perceptron (MLP) to learn the continuous distribution of the full scattered radiance product, represented as normalized anisotropic spherical Gaussian mixtures, to achieve lower sampling variance.

Importance sampling techniques significantly reduce variance in physically-based rendering. In this paper we propose a novel online framework to learn the spatial-varying distribution of the full product of the rendering equation, with a single small neural network using stochastic ray samples. The learned distributions can be used to efficiently sample the full product of incident light. To accomplish this, we introduce a novel closed-form density model, called the Normalized Anisotropic Spherical Gaussian mixture, that can model a complex light field with a small number of parameters and that can be directly sampled. Our framework progressively renders and learns the distribution, without requiring any warm-up phases. With the

Authors' addresses: Jiawei Huang, Chuzhou University, Chuzhou, China and Void Dimensions, China; Akito Iizuka, Tohoku University, Sendai, Japan; Hajime Tanaka, Tohoku University, Sendai, Japan; Taku Komura, The University of Hong Kong, Hong Kong and Tohoku University, Sendai, Japan; Yoshifumi Kitamura, Tohoku University, Sendai, Japan.

ACM 0730-0301/2024/2-ART
https://doi.org/10.1145/3649310

compact and expressive representation of our density model, our framework can be implemented entirely on the GPU, allowing it to produce high-quality images with limited computational resources. The results show that our framework outperforms existing neural path guiding approaches and achieves comparable or even better performance than state-of-the-art online statistical path guiding techniques.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: Wireless sensor networks, media access control, multi-channel, radio interference, time synchronization

1 INTRODUCTION

Unbiased physically-based rendering (PBR) is achieved by launching a light transport simulation to solve the rendering equation with Monte Carlo methods. Over the past few decades, unidirectional path tracing has become the dominant method of PBR in the film and design industries due to its flexibility and simplicity. To reduce the variance of the Monte Carlo integral efficiently, many importance sampling methods have been proposed (*e.g.*, [Hart et al. 2020; Ureña et al. 2013; Veach 1998]). However, currently, only some components in the rendering equation (typically, the BSDF term, or direct lighting) can be importance-sampled. When indirect lighting is dominant, the sampling efficiency becomes poor.

Path guiding is a promising genre of importance sampling approaches to overcome this challenge [Müller et al. 2017; Vorba et al. 2014]. A path guiding method usually learns distributions over the 3D scene that fit the rendering equation more closely, either during the rendering process (online) or with precomputation (offline). Then, the path tracer can sample the scattering directions using the learned distribution to reduce the variance. Previous approaches [Müller et al. 2017; Vorba et al. 2014] have partitioned the 3D space and approximated the incident radiance of each zone using statistical methods (typically, histogramming or expectation-maximization) instead of modeling the whole product of the rendering equation for each shading point; consequently, they could only learn one distribution for each spatial partition and usually suffered from the parallax issue.

Recently, neural networks are starting to be used for learning spatially varying, per-shading-point product distribution of the rendering equation [Müller et al. 2020; Zhu et al. 2021a,b]. Ideally, an explicit model that directly fits the continuous product distribution over 3D space is desired; however, this has been difficult due to the lack of (1) a representation that can reconstruct both high- and low-frequency features with a small number of parameters and (2) a closed-form density model that can provide an analytical solution for integrating the distribution for normalization. As a result, existing models either (1) learn an implicit model that only provides a mapping between a uniform distribution and the shading-point product distribution, but with expensive computation [Müller et al. 2020]; (2) learn a coarse, low-resolution distribution with limited accuracy [Zhu et al. 2021a]; or (3) require costly offline precomputation [Zhu et al. 2021b].

In this paper, we propose a novel, neural-network-based path guiding framework that learns an explicit spatial-varying distribution of scattered radiance over the surfaces in a 3D scene online. Using the shading point's 3D position and auxiliary information as input, our network estimates the distribution of full scattered radiance product, which can be used for efficient importance sampling of light transport. Our key insight is a novel closed-form density representation, namely the Normalized Anisotropic Spherical Gaussian mixture (NASG). The NASG is a 5-parameter anisotropic distribution model that comes with an easy-to-compute integral and a closed-form sampling algorithm, making it more feasible for Monte Carlo rendering compared to Kent distribution. We also show that normalization is an important factor for successful learning of scattered radiance distribution with sparse online training samples. The simplicity and expressiveness of NASG, along with its analytical normalization formula, lead to successful online learning of the complex radiance distribution via a tiny multilayer perceptron (MLP).

Our framework is robust enough to handle a variety of lighting setups, ranging from normal indirectly illuminated scenes to caustics. We also propose the corresponding training workflow, including specialized loss function and online acquisition of training samples. Because our network can be easily stored and executed in parallel on a GPU, we are able to integrate it with a wavefront path tracer. In this way, we can implement a high-performance, neural-guided, unidirectional path tracer on a single GPU, making neural guiding affordable for conventional personal computers. Our framework outperforms existing neural path guiding approaches both in terms of sampling efficiency and raw performance, and it provides comparable or even better performance than state-of-the-art statistical methods.

The contributions of this paper can be summarized as

- Normalized Anisotropic Spherical Gaussian mixture, a novel density model that can effectively represent spatial-varying distribution for path tracing, and
- A novel, lightweight, online neural path guiding framework that can be integrated with either CPU- or GPU-based conventional production path tracers.

2 RELATED WORK

2.1 Physically-based Rendering and Path Guiding

The radiance of a shading point can be calculated using the rendering equation [Kajiya 1986]:

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}) + \int_{S^2} f_s(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) |\cos\theta_i| d\omega_i, \tag{1}$$

which consists of the emitted radiance $L_e(\mathbf{x})$ at the point and the integral over the sphere S^2, which aggregates the contributions of the incident radiance $L_i(\mathbf{x}, \omega_i)$ from all directions ω_i. For each ω_i, the product of incident radiance, the bidirectional scattering distribution function (BSDF) $f_s(\mathbf{x}, \omega_i, \omega_o)$, and the geometry term $|\cos\theta_i|$ signify its contribution to the outgoing radiance. In 3D scenes, due to light bouncing off surfaces, the incoming radiance L_i can be considered as originating from another point on a different surface. This implies that L_i also follows the integral form described by the rendering equation Eq. (1) and needs to be evaluated at the point from which the scattered light originates. This makes the problem highly complex, and no analytical solution can be formed in general.

Path tracers solve the rendering equation via the Monte Carlo method, which draws random direction samples ω_i', and the average of samples is the estimation of the integral:

$$L(\mathbf{x}, \omega_o) \approx L_e(\mathbf{x}) + \frac{1}{N} \sum_k^N \frac{f_s(\mathbf{x}, \omega_{ik}', \omega_o) L_{ik}(\mathbf{x}, \omega_{ik}') |\cos\theta_{ik}|}{p(\omega_{ik}')}, \tag{2}$$

where $p(\omega_{ik}')$ is the probability density function (PDF) from which the integrator samples at point $\mathbf{x}$. Due to the complexity of light transport over the 3D space, the variance of samples is high, and many importance sampling methods have been proposed to reduce this variance (e.g., [Conty Estevez and Kulla 2018; Hanika et al. 2015; Veach 1998]). These methods usually focus on importance sampling of single components of the rendering equation. Here, a universal importance sampling technique for the whole product of the rendering equation can help further improve sampling efficiency. Path guiding, which we describe next, is a genre of methods developed along this direction.

Path guiding methods aim to find sufficient approximation of the incident radiance distribution, which serves to achieve better importance sampling that follows the global illumination distribution rather than local BSDF distribution. Jensen [1995] and Lafortune et al. [1995] first proposed to fit the incident light distribution for more efficient indirect lighting sampling. Vorba et al. [2014] fit a Gaussian mixture to model the distribution estimated by an explicit photon tracing pass. Müller et al. [2019; 2017] proposed the "Practical Path Guiding (PPG)"

algorithm to model the distribution using an SD-tree approach. Based on these techniques, full product guiding methods have also been proposed. Herholz et al. [2016] calculated an approximation of full product on top of an earlier work [Vorba et al. 2014], and Diolatzis et al. [2020] calculated an approximation on top of another study [Müller et al. 2017] to achieve product guiding. Both of these methods operate at the cost of higher overhead as the approximated product needs to be calculated multiple times from the corresponding learned incident radiance distribution. The above techniques share the same idea of partitioning the space and progressively learning discrete distributions. Each discrete distribution is shared among points within the spatial partition. A major issue of this approach is parallax error: the above techniques fail to accurately learn the distribution for close-distance incident radiance where the incoming light quickly changes within the same spatial partition. Ruppert et al. [2020] proposed a parallax-aware robust fitting method to address this issue with a discrete approach. We compared their approach with our network-based approach in § 7.3. Recent research shows that good path guiding requires a proper blending weight between BSDF and product-driven distributions [Müller et al. 2020], and the learned distribution should be variance-aware since the samples are not zero-variance [Rath et al. 2020]. These findings can help to achieve a more robust path guiding framework.

Recently, neural networks have been used to model scattered radiance distribution; both online and offline learning methods have been proposed. Zhu et al. [2021b] used neural networks to estimate a quad-tree representation of incident radiance distribution using nearest photons as input. Currius et al. [2020] used convolutional neural networks (CNN) to estimate incident radiance represented by spherical Gaussians; this work is for real-time rendering, but the estimated radiance distribution could be used for path guiding. Zhu et al. [2021a] applied an offline-trained neural network to efficiently sample complex scenes with lamps. This technique requires training a U-Net for more than 10 hours for just a single light source, while the estimated distribution is only a 16×16 2D map, which is not sufficient for representing general indirect lighting distributions. A much higher resolution is required for robustly guiding over the entire 3D scene; for instance, PPG's quad-tree approach can represent a resolution of $2^{16} \times 2^{16}$). These methods are categorized as offline learning, since they require training a network offline with massive training samples using ground truth distributions.

Our framework, however, is categorized as online learning, which learns the distribution on the fly, without a ground truth distribution as reference. Previously, Müller et al. [2020] adopted normalizing flow [Kobyzev et al. 2021] to model the full product of incident radiance distribution. However, with an implicit density model like normalizing flow, each sample/density evaluation requires a full forward pass of multiple neural networks, which introduces heavy computation costs. In a modern path tracer with multiple importance sampling (typically, BSDF sampling and next event estimation (NEE)), this means full forward pass needs to be executed for each surface sample at least two times. Moreover, the training process requires dense usage of differentiable transforms, which makes the training slower than that for regular neural networks. Indeed, Müller et al. [2020] used two GPUs specifically for the normalizing flow's neural network computation along with the CPU-based path tracing implementation; nevertheless, the sampling speed was still only 1/4 of PPG. Our work, in contrast, proposes the use of an explicitly parameterized density model in closed-form that can be learned using a small MLP. The neural network is used to generate the closed-form distribution model, rather than actual samples; therefore, we are able to freely generate samples or evaluate the density after a single forward pass. With careful implementation, we show that our method can reach a similar sampling rate to that of PPG, while the result has lower variance. In research that is parallel with our work, Dong et al. [2023] used a small neural network to estimate per-shading-point distribution, which is similar to our approach. However, we also propose the use of NASG, a novel anisotropic model, in place of classic von Mises-Fisher distribution. NASG helps to further improve the fitting accuracy and guiding efficiency. We highlight the benefits through experiments in § 7.

2.2 GPU Path Tracing

GPU path tracing has gained significant attention in recent years due to its high performance, supported by the GPU's ability to concurrently render many pixels. While many recent research works have focused on constructing real-time path tracers [Bitterli et al. 2020; Chaitanya et al. 2017; Lin et al. 2022; Müller et al. 2021; Ouyang et al. 2021]), GPU-based production renderers leverage GPU to render fully converged, noise-free results faster [Maxon 2023; OTOY 2023]. Compared to CPU architecture, GPU requires a sophisticated programming design for better concurrency, which is needed for overcoming the large memory latency of GPU. One of the general idea of this is ray re-ordering [Lü et al. 2017; Meister et al. 2020]. Laine et al. [2013] proposed a new architecture that splits full path tracing computation into stages (small kernels) and processes threads in the same stage to maximize performance. Recently, Zheng et al. [2022] proposed a more sophisticated framework for GPU path tracing, which leverages run-time compilation for high performance and flexible implementation. While several GPU-based renderer products are available and it has been proven that path guiding can improve sampling efficiency, few products have provided GPU-based path guiding implementation. This is because existing path guiding approaches are designed for CPUs, and specialization for GPUs only emerged recently at the research level [Dittebrandt et al. 2020; Pantaleoni 2020]. Our work provides a low-cost continuous path guiding option for GPU-based path tracers.

2.3 Density Models

Parametric density models have been extensively explored in statistics. Exponential distribution is a representative family, among which Gaussian mixture is the most widely used density model. Vorba et al. [2014] used a 2D Gaussian mixture to model incident radiance distribution; however, since the domain of a 2D Gaussian is the entire 2D plane, when mapping it to unit sphere, it is necessary to discard samples outside the domain, which affects computation efficiency. Dodik et al. [2022] further proposed using 5D Gaussians to model incident radiance distribution over the space. By using a tangent space formulation, it greatly reduced the number of discarded samples.

In the context of physically-based rendering, spherical models have been widely leveraged. The spherical Gaussian (SG) is widely used for radiance representation and density modelling (e.g., [Wang et al. 2009]). Actually, a normalized SG is equivalent to the von Mises-Fisher (vMF) distribution in 3D. It has several desirable properties, such as computational efficiency and analytical tractability for integrals. However, its expressiveness is limited due to its isotropic nature, which restricts the shape of the distribution on the sphere. The Kent distribution [1982] presents a possible remedy by generalizing the 3D vMF model to a five-parameter anisotopic spherical exponential model, providing higher expressiveness. However, its application within our framework is impeded by three significant limitations: high computational cost, the absence of a direct sampling method, and issues with numerical precision, particularly when the parameters for controlling concentration are high. These limitations are elaborated upon in § 4.1.

Other recent works proposed spherical models specifically for graphics. Xu et al. [2013] proposed an anisotropic spherical Gaussian model to achieve a larger variety of shapes; however, no analytic solution was found for its integral, which is problematic for normalization.

Heitz et al. [2016] proposed another anisotripic model with a closed-form expression called linearly transformed cosines (LTC). However, the integral of LTC requires computation of the inverse matrix, and one component requires in total 12 scalars to parameterize a component.

Another family of density models widely adopted in the rendering context is the polynomial family, among which spherical harmonics is the most commonly used model. Polynomial models have been extensively used in graphics [Moon et al. 2016; Sloan et al. 2002]. They can be easily mapped to unit spheres to represent spherical distributions. However, polynomial models are generally limited to capturing low-frequency distributions. To

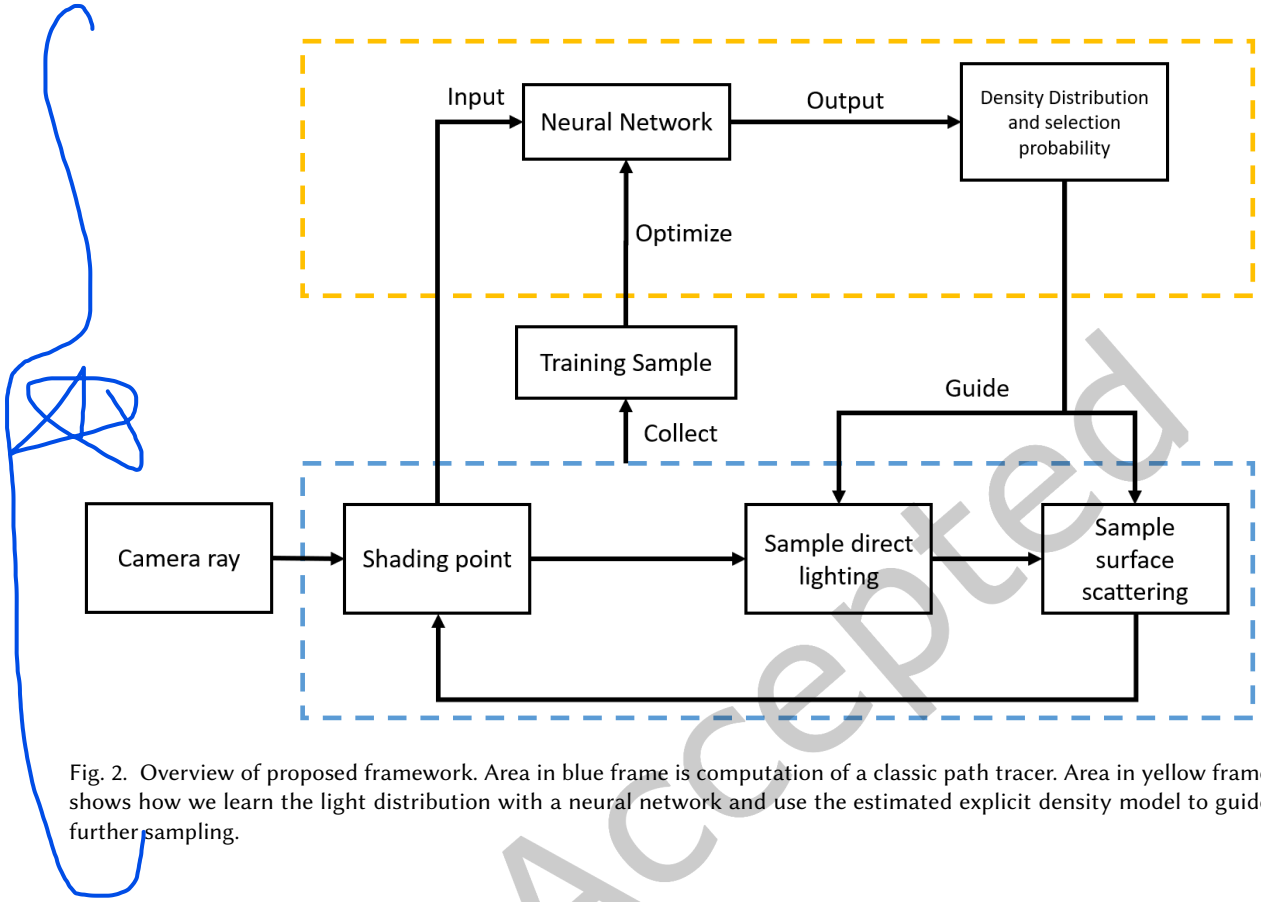

Fig. 2. Overview of proposed framework. Area in blue frame is computation of a classic path tracer. Area in yellow frame shows how we learn the light distribution with a neural network and use the estimated explicit density model to guide further sampling.

represent high-frequency distributions, a very high degree is required, leading to a significant increase in the number of parameters. Furthermore, there is currently no efficient approach to directly sample a polynomial distribution, although there does exist some relatively expensive importance sampling schemes (e.g., [Jarosz et al. 2009]). These were the main challenges in our pilot study that made us give up the idea of adopting them for our path guiding scheme.

Learning-based density models have been drawing more attention recently [Gilboa et al. 2021; Müller et al. 2020]. The model based on normalizing flow can successfully learn complex distributions [Müller et al. 2020] but suffers from the implicitness and heavy computation mentioned earlier. Marginalizable Density Model Approximation (MDMA) [Gilboa et al. 2021] has been proposed as a closed-form learning-based density model, which is essentially a linear combination of multiple sets of 1D distributions. MDMA showed that closed-form normalization is an essential factor in learning density models, and inspired by their work, we propose such a model for our application.

3 OVERVIEW

The goal of our work is to model the distribution of the product of incident light over the unit sphere for every unique shading point $\mathbf{x}$ so that the product can be fully importance-sampled (see § 4 for details of our density model). In other words, we map the shading point's 3D position and auxiliary data to a parameterized spherical distribution.

As shown in Fig. 2, our framework learns the distribution online, in a progressive manner. In every rendering iteration, the image is rendered at one sample per pixel, and a subset of the pixel samples are collected to train an MLP (see § 6 for details of collecting strategy). After the image is rendered and training samples are collected, we train the MLP using an optimizer implemented with Libtorch (C++ frontend of Pytorch [2017]) as described in § 5. When rendering an image, every time a ray hits a shading point $\mathbf{x}$, our framework uses a learned MLP (whose weights are updated every M iterations as described in § 6) to estimate the product distribution, and it draws the sample from either the BSDF distribution or the learned one, based on the learned selection probability.

4 NORMALIZED ANISOTROPIC SPHERICAL GAUSSIAN MIXTURE

In this section, we describe our proposed density model we call the Normalized Anisotropic Spherical Gaussian mixture (NASG), which allows us to efficiently learn the light distribution at each shading point. We first review the existing density models that inspired our model in § 4.1. We then describe our model itself in § 4.2.

4.1 Background

One of the main requirements for progressive learning of a distribution with neural networks is that models be easily normalizable, implying that a model must have a closed-form integral.

Marginalizable Density Model Approximation. Gilboa et al. [2021] proposed Marginable Density Model Approximation (MDMA), which can be effectively optimized via deep learning. A bivariate dimensional distribution can be modeled as

$$D(x, y) = \sum_{i,j} A_{ij} \phi_{1i}(x) \phi_{2j}(y), \tag{3}$$

where ϕ_{1i} and ϕ_{2j} are 1D normalized distributions and A_{ij} are normalized coefficients that sum to 1.

In practice, the 1D distributions in MDMA can be piece-wise linear or piece-wise quadratic spline models; however, with its limited number of components, MDMA can learn only a coarse distribution. Therefore, it always gives poor results for distributions with high-frequency spots. Despite the limited accuracy, MDMA works well in learning distributions from sparse noisy samples, compared with non-normalizable models (*e.g.,* polynomial model, spherical harmonics). During training, samples with high energy can lower the contribution of other areas, and eventually the training converges. This feature is crucial in learning distributions from sparse training samples.

Normalized Gaussian Mixtures. Normalized Gaussian mixtures can address the accuracy limitations of MDMA:

$$D(\mathbf{x}) = \sum_{i}^{N} A_i \frac{G_i(\mathbf{x}; \theta_i)}{K_i}, \tag{4}$$

where $G_i(\mathbf{x}; \theta_i)$ are Gaussian distributions parameterized by θ_i, K_i are normalizing factors, and A_i are normalized weights of Gaussians such that $\sum_i A_i = 1$. Gaussian mixtures are highly expressive and also easily normalizable: we look into two common Gaussian distributions that can represent the spherical distributions needed for modeling the lighting: spherical Gaussian and anisotropic spherical Gaussian.

Spherical Gaussian. Spherical Gaussian (SG) is a variant of the univariate Gaussian function defined in the spherical domain:

$$G(\mathbf{v}; \boldsymbol{\mu}, \lambda) = \exp(\lambda(\boldsymbol{\mu} \cdot \mathbf{v} - 1)), \tag{5}$$

where $\mathbf{v}$ is a unit vector specifying the direction, $\boldsymbol{\mu}$ is the lobe axis, and λ is a parameter that controls the "sharpness" of the distribution. The normalizing term of an SG can be computed by

$$K = \int_{S^2} G(\mathbf{v}; \boldsymbol{\mu}, \lambda) d\omega = \frac{2\pi}{\lambda}(1 - e^{-2\lambda}). \tag{6}$$

Normalized SG is equivalent to the vMF distribution in 3D. Since SG is defined in the spherical domain and is suitable for representing all-frequency light distribution at each sample point, mixture models based on SG have been applied in representing the light map used for precomputed radiance transfer (e.g., [Currius et al. 2020; Wang et al. 2009]). However, SG is an isotropic univariate model, and thus less expressive; for example, it cannot represent anisotropic distributions.

Anisotropic Spherical Gaussian. To model anisotropic distributions, Xu et al. [2013] proposed Anisotropic Spherical Gaussian (ASG) to model the light map. ASG can be written as

$$G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], [a, b], c) = c \cdot S(\mathbf{v}; \boldsymbol{z}) \cdot e^{(-a(\mathbf{v}\cdot\boldsymbol{x})^2 - b(\mathbf{v}\cdot\boldsymbol{y})^2)}, \tag{7}$$

where $\boldsymbol{z}$, $\boldsymbol{x}$, and $\boldsymbol{y}$ are the lobe, tangent, and bi-tangent axes, respectively, and $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ form an orthonormal frame; a and b are the bandwidths for the $\boldsymbol{x}$- and $\boldsymbol{y}$-axes, respectively, and c is the lobe amplitude. Xu et al. [2013] successfully modeled complex lighting conditions and rendered anisotropic metal dishes using ASG. On the other hand, ASG does not have a closed-form solution for the integral, which inhibits its usage for our purpose.

Kent Distribution. Another candidate is the Kent distribution [Kent 1982], which is also an anisotropic spherical distribution:

$$K(\mathbf{v}; \kappa, \beta) = \frac{1}{C(\kappa, \beta)} \exp\left(\kappa(\mathbf{v}\cdot\mathbf{z}) + \beta((\mathbf{v}\cdot\mathbf{x})^2 - (\mathbf{v}\cdot\mathbf{y})^2)\right), \tag{8}$$

where κ, β are parameters that control anisotropic concentration with $0 < 2\beta < \kappa$, and $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ form an orthonormal frame in the same way as ASG. The Kent distribution has a closed-form integral, i.e., its normalizing constant:

$$C(\kappa, \beta) = 2\pi \sum_{i=0}^{\infty} \frac{\Gamma(i + 0.5)}{\Gamma(i + 1)} \beta^{2i} \left(\frac{\kappa}{2}\right)^{-2i-\frac{1}{2}} I_{2i+\frac{1}{2}}(\kappa), \tag{9}$$

where $\Gamma(\cdot)$ is the gamma function, and $I_n(\kappa)$ is the modified Bessel function of the first kind. However, since this integral entails the evaluation of a nested infinite series (since $I_n(\kappa)$ is also an infinite series), sufficient precision necessitates the computation of a significant number of terms, thus imposing a substantial computational overhead. In the context of computer science, the precision of the Kent distribution is further constrained by the limitations on floating-point number precision. Specifically, when the value of κ is large (e.g., $\kappa > 20$), both the non-normalized density and the normalizing constant approach the representational boundary of floating-point numbers, resulting in a lack of precision. This leads to either a biased result or disrupted learning (where an NaN loss is generated and corrupts the learning process). Within the realm of Monte Carlo rendering, an additional limitation emerges, since Kent distribution lacks a direct sampling method, which necessitates the use of a more computationally expensive rejection method.

4.2 Normalized Anisotropic Spherical Gaussian

Inspired by the models in § 4.1, we believe a spherical anisotropic exponential distribution can help to achieve more accurate results in our framework. While the Kent distribution exists as a potential candidate, its inherent limitations - namely, the necessity of approximating its integral, the precision issue, and the absence of a direct sampling algorithm - prompt us to develop our own distribution, which we call the Normalized Anisotropic Spherical Gaussian:

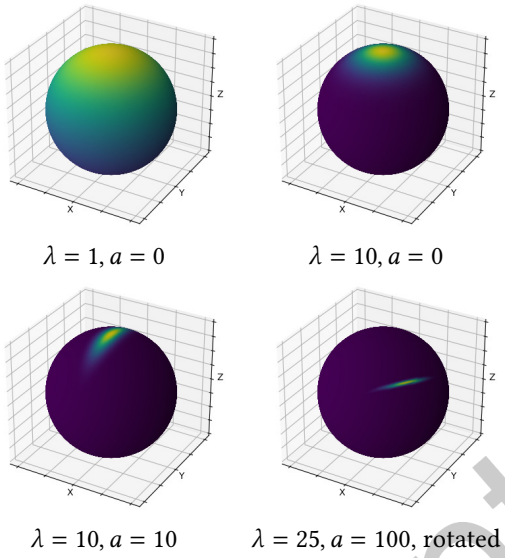

Fig. 3. Visualization of NASG component G with different parameters. Note that G agrees with spherical Gaussian when $a = 0$.

$$G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a) = \begin{cases} \exp\left(2\lambda\left(\frac{\mathbf{v}\cdot\boldsymbol{z}+1}{2}\right)^{1+\frac{a(\mathbf{v}\cdot\boldsymbol{x})^2}{1-(\mathbf{v}\cdot\boldsymbol{z})^2}} - 2\lambda\right)\left(\frac{\mathbf{v}\cdot\boldsymbol{z}+1}{2}\right)^{\frac{a(\mathbf{v}\cdot\boldsymbol{x})^2}{1-(\mathbf{v}\cdot\boldsymbol{z})^2}} & \text{if } \mathbf{v} \neq \pm\boldsymbol{z} \\ 1 & \text{if } \mathbf{v} = \boldsymbol{z} \\ 0 & \text{if } \mathbf{v} = -\boldsymbol{z}, \end{cases} \tag{10}$$

where $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ forms an orthogonal frame as above, λ is sharpness, and a controls the eccentricity. Fig. 3 shows examples of the NASG component with different parameters.

The derivation of NASG proceeds as follows. First, we employ the standard spherical coordinate expression of $\mathbf{v} \neq \pm\boldsymbol{z}$ in terms of the polar angle θ and the azimuthal angle ϕ (with respect to the orthonormal frame $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$) to obtain the equation

$$\left(\frac{\mathbf{v}\cdot\boldsymbol{z}+1}{2}\right)^{\frac{a(\mathbf{v}\cdot\boldsymbol{x})^2}{1-(\mathbf{v}\cdot\boldsymbol{z})^2}} = \left(\frac{\cos\theta+1}{2}\right)^{a\cos^2\phi}, \tag{11}$$

which introduces the anisotropic nature of the distribution through the exponent $a\cos^2\phi$. Next, we effect a change of variables in the integral of SG (cf. Eq. (6)) so that the Jacobian in Eq. (11) emerges as part of the corresponding Jacobian. In this way, we arrive at NASG, which satisfies all requirements as a model for learning the distribution needed for unbiased rendering.

Different from ASG [Xu et al. 2013], NASG has an **analytical closed-form solution** for its integral (see § B), which makes it easy to normalize:

$$K = \int_{S^2} G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a)\, d\omega = \frac{2\pi\left(1 - e^{-2\lambda}\right)}{\lambda\sqrt{1+a}}. \tag{12}$$

NASG is also **expressive**, since it can model anisotropic distributions, as well as **numerically stable** to represent high- and low-frequency distributions accurately [1]. It is also **compact** because it is parameterized by only 7 scalars, making it highly efficient in GPU-based computation due to its low bandwidth requirement. Additionally, the sampling algorithm of NASG is remarkably **efficient** (see § C). These characteristics make NASG a highly feasible and practical option for our framework.

5 ONLINE LEARNING OF DENSITY MODEL

We now describe our neural network structure for learning the parameters of our NASG model, as well as the training scheme.

5.1 Network Architecture

We intend our neural network to be as simple as possible, considering the performance requirements for practical use. In this work, we use a 4-layer MLP, where each layer has 128 units *without* bias.

Network Inputs. The input to the model consists of the location of the shading point $\mathbf{x}$, outgoing ray direction ω_o, and surface normal $\mathbf{n}$. The location of the shading point is first encoded into a 57 dimensional vector by positional encoding. Thus, the input size is 63 (57 + 3 + 3), but is padded to 64 for hardware acceleration purposes[2]. We adopt the simple one-blob encoding [Müller et al. 2019] instead of complex encoding mechanics, such as *Multiresolution Hash Encoding* [Müller et al. 2022], because there was very little improvement in our pilot study to justify the overhead such mechanics introduce. Details of the encoding scheme can be found in Tab. 1.

Table 1. Encoding scheme of network inputs

Parameter	Symbol	Encoding
Position	$p \in \mathbb{R}^3$	$ob(p)$
Outgoing ray direction	$\omega_o \in [-1, 1]^3$	ω_o
Surface normal	$\mathbf{n} \in [-1, 1]^3$	$\mathbf{n}$

Parameterization of NASG. To improve the efficiency of learning, we reduce the number of parameters used to represent the NASG model. First, we introduce θ, ϕ, τ, the Euler angles representing the orientation of the orthogonal basis with respect to the global axes. Accordingly, $\mathbf{z}$ and $\mathbf{x}$ of the orthogonal basis can be represented by

$$\mathbf{z} = \begin{pmatrix} \cos\phi \sin\theta \\ \sin\phi \sin\theta \\ \cos\theta \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} \cos\theta \cos\phi \cos\tau - \sin\phi \sin\tau \\ \cos\theta \sin\phi \cos\tau + \cos\phi \sin\tau \\ -\sin\theta \cos\tau \end{pmatrix}. \tag{13}$$

Thus, we represent NASG with the following seven parameters: $cos\theta$, $sin\phi$, $cos\phi$, $sin\tau$, $cos\tau$, and λ, a.

Network Outputs. The output of our neural network is a d dimensional vector $\boldsymbol{o}$, where each component corresponds to the NASG parameter at the shading point $\mathbf{x}$ and a selection probability c. Assuming the number of mixture components is N, $d = 5 \times N + 2 \times N + N + 1 = 8N + 1$ (see Tab. 2). These parameters are further decoded to represent the actual parameters as summarized in Tab. 2. The mixture weights of NASG $\mathbf{A}$ are normalized with a softmax function.

[1] NASG is not continuous at $\mathbf{v} = -\boldsymbol{z}$ when $a > 0$ in this form. Indeed, it approaches 0 if we let $\mathbf{v}$ tend to $-\boldsymbol{z}$ along any meridian, except for the ones passing through $\pm\boldsymbol{y}$ (i.e., $\phi = \pi/2$ and $\phi = 3\pi/2$), in which case NASG approaches $\exp(-2\lambda) > 0$. However, this discontinuity does not affect our application, and it can be resolved easily by introducing an auxiliary parameter (see § A for details)

[2] By setting padded values to 1 we achieve an alternative to the bias of hidden layers.

Table 2. Decoding scheme for NASGM with N components

Parameter	Symbol	Decoding
$\cos\theta$, $\sin\phi$, $\cos\phi$, $\sin\tau$, $\cos\tau$	$s_0 \in R^5 \times N$	$sigmoid(s_0) \times 2 - 1$
λ, a	$s_1 \in R^2 \times N$	e^{s_1}
Component weights	$\mathbf{A} \in R^N$	$softmax(A)$
Selection probability	$c \in R$	$sigmoid(c)$

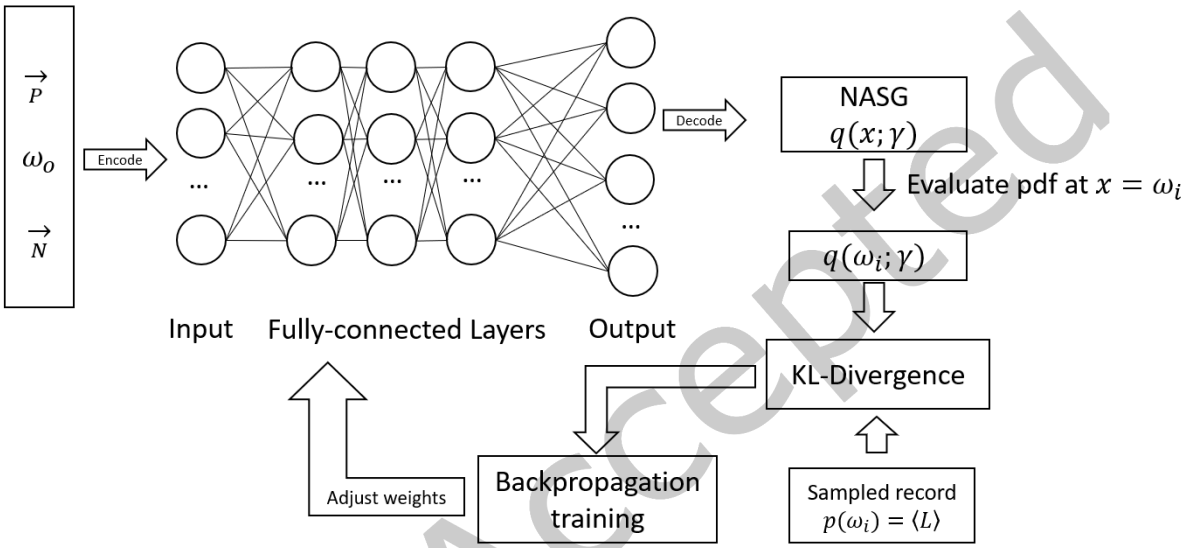

Fig. 4. Training process of proposed network. We heavily apply automatic differentiation to adjust network weights based on estimated distribution and rendering estimation $\langle L \rangle$ from the sampled scattering direction ω_i.

5.2 Training

We now describe our loss function and the training process. As shown in Fig. 4, we utilize automatic differentiation to train the network effectively with sparse online rendering samples. Since the training data are noisy online samples, we need a loss function that is more robust than regular L1 or L2 losses.

Kullback-Leibler Divergence. The design of a loss function for distribution learning in PBR context has been studied in several previous works [Müller et al. 2019, 2020; Zhu et al. 2021a,b]. By using $q(\omega_i;\gamma)$ to denote the NASG distribution (with NASG parameters γ) estimated by the neural network, to achieve importance sampling of the rendering equation, the optimal distribution should be proportional to the product:

$$q(\omega_i;\gamma) \propto f_s(\mathbf{x},\omega_i,\omega_o)L_i(\mathbf{x},\omega_i)|\cos\theta_i|. \tag{14}$$

Therefore, we use $p(\omega_i) = Ff_s(\mathbf{x},\omega_i,\omega_o)L_i(\mathbf{x},\omega_i)|\cos\theta_i|$ as our target distribution, where F is a normalizing term whose value is unknown. We can still train the system without knowing F as shown below.

In our framework, we use the Kullback-Leibler Divergence (KL-Divergence) to represent the likelihood between our estimated distribution $q(\omega_i;\gamma)$ and the target one:

$$D_{KL}(p(\omega_i)||q(\omega_i;\gamma)) = \int_{S^2} p(\omega_i)(\log[p(\omega_i)] - \log[q(\omega_i;\gamma)])d\omega_i. \tag{15}$$

Since $p(\omega_i)$ is γ-independent, we get

$$\nabla_\gamma D_{KL}(p(\omega_i)||q(\omega_i;\gamma)) = -\int_{S^2} p(\omega_i)\nabla_\gamma \log[q(\omega_i;\gamma)]d\omega_i. \tag{16}$$

Then, the integral can be replaced with our one-sample estimation:

$$\nabla_\gamma D_{KL}(p(\omega_i)||q(\omega_i;\gamma)) = -\mathbb{E}\left[\frac{p(\omega_i)}{\hat{q}(\omega_i;\gamma)}\nabla_\gamma \log[q(\omega_i;\gamma)]\right], \tag{17}$$

where $\hat{q}(\omega_i;\gamma)$ denotes the PDF of the distribution the sample obeys during rendering (see below). Although the right-hand side still involves the global scaling factor F, it is canceled in moment-based optimizers such as Adam [Kingma and Ba 2014]; accordingly, we can train the system even though there is an unknown scaling factor in $p(\omega_i)$ [Müller et al. 2019].

Selection Probability. Following [Müller et al. 2020], we do not directly use a distribution estimated by our neural network. Instead, we regard our scattering sampling process as an MIS process that blends the learned distribution and the BSDF distribution $p_{f_s}(\omega_i)$. As we also learn a selection probability c, as described in § 5.1, the MIS PDF at shading point $\mathbf{x}$ is $\hat{q}(\omega_i;\gamma) = cq(\omega_i;\gamma) + (1-c)p_{f_s}(\omega_i)$. However, optimizing $\hat{q}(\omega_i;\gamma)$ naively leads to falling into local optima with degenerate selection probability; therefore, our final loss function blends $\hat{q}(\omega_i;\gamma)$ with $q(\omega_i;\gamma)$:

$$loss = eD_{KL}(p(\omega_i)||\hat{q}(\omega_i;\gamma)) + (1-e)D_{KL}(p(\omega_i)||q(\omega_i;\gamma)), \tag{18}$$

where e is a fixed fraction that we set to 0.2.

6 IMPLEMENTATION DETAILS

We integrate our framework into Clight, our in-house wavefront GPU path tracer for production [Void Dimensions 2024]. In this section we discuss implementation details as well as design considerations.

Progressive Learning and Sampling. Our framework collects samples to train the network online, and we use the network to generate a distribution that is then blended with the BSDF sampling distribution using the learned selection probability c. We further multiply an extra coefficient b to compute an actual blending weight $c' = bc$, where b is initially set to 0 and gradually increased to 1, such that the framework initially relies on the BSDF distribution but gradually switches to the learned distribution $\hat{q}(\omega_i;\gamma)$. We use a fixed-step strategy to ensure that our guided sampling uses a sufficient number of samples for learning the distribution: for every M images rendered, we increase b by $\frac{1}{B}$. In our implementation, we set $M = 4$ and $B = 64$. This could require adjustments according to the rendering task, and further discussions are given in § 8.3.

For rendering images while progressively learning the distribution, previous works [Müller et al. 2017, 2019, 2020] blended the rendered image samples based on their inverse variance. Such a process introduces extra memory and computation overhead, especially for a renderer that produces arbitrary output variables (AOVs). Based on the observation that the variance of individual image samples decreases as the learning proceeds, we use a simplified method that scales the weights of accumulated results according to the training steps. This essentially gives more weight to later samples in a progressive manner. The weight scaling process stops when the training step count reaches $M \times R$. Since the rendering result at each step can be seen as an unbiased estimation of the ground truth, and the weighted average can be seen as an MIS process with different sampling techniques (in this case, different distributions), the result remains unbiased.

Network Optimization. The process of distribution learning runs on GPU after every rendering iteration. We use Libtorch, the C++ frontend of Pytorch [Paszke et al. 2017], for the implementation, along with the Adam optimizer with its learning rate set to 0.002. When rendering an image, we split the image into smaller tiles of size $l \times l$ (l is a real number, and $l \geq 1$) and collect data from one pixel per tile in each render iteration. After the iteration of rendering, we update l based on collected sample size s and maximum size S as $l = \max(1, l\sqrt{\frac{s}{S}})$; consequently, after rendering the full image plane, we can get an s close to S. Note that if s exceeds S, we only record and keep S samples for training.

After each render iteration, the collected data are directly used for progressively training the model, and the training step is determined by $T = \nu \cdot \lceil \frac{S}{t} \rceil$ so that every sample will be trained at least ν times. Larger S and ν help to improve the learning accuracy, but the training overhead introduces a significant trade-off. Here, we set the following parameters: maximum size of training samples in each iteration $S = 2^{16}$, training batch size $t = 2^{12}$, and step factor $\nu = 1$. The differences in the results obtained with different configurations are revealed by experiment in § 7.2.

Direct Lighting with NEE. Our implementation uses MIS to blend guided scattering directions with Next Event Estimation (NEE) to sample direct lighting. NEE helps to improve the quality of the images, especially at the beginning of the training stage. When recording radiance to a training sample, we use the MIS-weighted value. It should be noted that MIS is only practical because our model is an explicit model that allows us to freely sample the distribution and evaluate PDF in a certain direction, which could be computationally expensive for implicit models such as [Müller et al. 2019, 2020].

Wavefront Architecture. Neural network computation can be greatly accelerated with hardware (*e.g.,* Tensor Core). However, such hardware is usually designed for batched execution, in which a single matrix multiplication involves multiple threads. This conflicts with the classic thread-independent "Mega-kernel" path tracing implementation. Our implementation adopts "Wavefront path tracing" architecture [Laine et al. 2013] to solve this problem. In a wavefront path tracer, all of the pixels at the same stage are executed concurrently. Under this condition, we are able to "insert" our neural network calculation seamlessly right before the sampling/shading stage and minimize the computation overhead for best performance. The forward pass of our MLP does not require gradient calculation, so we implement it on the GPU with CUBLAS, and the resulting implementation is substantially faster than when using existing deep learning frameworks. Moreover, this implementation achieves a low overhead that makes our framework practical with conventional hardware, the performance of which we evaluate in § 7.2.

7 EVALUATION

We evaluated our framework in three ways. First, we integrated our framework into our in-house GPU production renderer. We evaluated its variance reduction efficiency by comparing it to that of plain path tracing. To demonstrate the benefits of our NASG and product guiding, we compared the results rendered with incident radiance guiding and other density models. Second, we evaluated its raw performance through a comparison with the results produced by path tracing in the same amount of time; here, our method was executed multiple times under different meta-parameter configurations. Finally, we implemented our framework in Mitsuba and compared the results produced with the same number of samples by several state-of-the-art methods, including Practical Path Guiding (PPG, [Müller 2019]), Robust Fitting of Parallax-Aware Mixtures for Path Guiding (PAVMM, [Ruppert et al. 2020]), and Neural Importance Sampling (NIS, [Müller et al. 2019]). We used mean absolute percentage error (MAPE) for quality measurement. MAPE is calculated by dividing the absolute error by the value of the ground truth. We added a small $\epsilon = 0.01$ to the denominator to avoid divisions by zero. When calculating the MAPE from all of the rendering results, we discarded 0.1% of the pixels with the highest error before computing the average.

7.1 Variance Reduction

We implemented our framework on top of our in-house GPU production renderer and evaluated the quality of the images by comparing them with those produced by plain path tracing using the same number of samples per pixel (SPP). We implemented both the sampling algorithm and the optimizer on the GPU, leaving the CPU to only perform scheduling tasks. We also compared different guiding schemes, i.e., guiding with incident radiance distribution, cosine-weighted radiance distribution, and full product distribution. In addition, we compared NASG with SG, MDMA, 2D Gaussians, and Kent distribution. Here, we used 8-component NASG (64 parameters). For other density models, we adjusted the number of parameters of each Gaussian to match that of NASG: SG uses 14 components (70 parameters), 2D Gaussians uses 12 components (72 parameters), Kent distribution uses 8 components (64 parameters), and MDMA uses 8 1D distributions for both dimensions (80 parameters). For Kent distribution, even when calculations are performed in the logarithmic space to mitigate the precision issues, we found it necessary to restrict the range of κ to $0 < \kappa < 20$, and accordingly constrained $0 < 2\beta < \kappa$ to prevent biased results or unstable learning.

We run our GPU path tracer on a conventional PC with a 4-core i7 9700K CPU and an Nvidia 2080ti GPU. The implementation of our framework introduces roughly 600 MB memory overhead on the GPU, mainly for buffering batched network execution. Our Pytorch optimizer takes another 600 MB of GPU memory to cache tensors and gradients. The test 3D scenes are shown in Fig. 5, and we render the scenes with NEE enabled. All of the rendered images are included in the supplemental material. Tab. 3 shows the results of the quantitative evaluation.

Fig. 5. Eight scenes used to evaluate proposed GPU implementation. From left to right: Ajar, Bathroom, Bidir, Corridor, Kitchen, Glossy Tube, Box, Pool.

Learned distribution. Our framework can easily learn a full product. In all of the test scenes, guiding with full product distribution gives lower variance. Guiding with full product helps to effectively reduce variance when rendering smooth surfaces. This is because learning full product not only improves the accuracy of the learned distribution but also helps to learn an optimal selection probability. The difference is more obvious in scenes such as Ajar and Corridor, where the floors are large surfaces with low roughness. When guiding with incident

Table 3. Errors of the results from variance reduction experiment, reported in MAPE. We rendered a variety of 3D scenes at 1024 samples per pixel (SPP), guided with our framework and different learned distributions. Scenes were rendered in different resolutions for a larger variety. We compare the learning distribution of incident radiance using NASG (NASG Li) with both the cosine-weighted incident radiance (+cosθ) and the full product distribution. We also compare the learning distribution of full product with different density models, i.e., spherical Gaussians (SG), MDMA (MDMA), 2D Gaussians (G2D), and Kent distribution (KENT). The time cost of each method is provided for completeness.

Scene		NASG Li	+cosθ	Full Product	SG	MDMA	G2D	KENT	PT
Ajar	MAPE	0.293	0.290	**0.045**	0.055	0.084	0.071	0.066	0.154
	Time (s)	250	259	264	249	325	208	1641	65
Bathroom	MAPE	0.085	0.073	**0.064**	0.066	0.081	0.078	0.098	0.140
	Time (s)	360	373	381	358	469	300	2261	170
Bidir	MAPE	0.086	0.079	**0.058**	0.066	0.133	0.090	0.137	0.319
	Time (s)	115	120	126	119	151	100	793	25
Box	MAPE	0.056	0.056	**0.051**	0.061	0.081	0.076	0.066	0.118
	Time (s)	83	86	87	83	108	69	545	17
Corridor	MAPE	0.148	0.142	**0.114**	0.117	0.150	0.132	0.146	0.243
	Time (s)	209	211	220	212	272	178	1268	83
Kitchen	MAPE	0.102	0.092	**0.042**	0.043	0.051	0.072	0.046	0.078
	Time (s)	303	308	309	302	395	253	1980	126
Glossy Tube	MAPE	0.010	0.011	**0.009**	0.015	0.010	0.015	0.010	0.018
	Time (s)	134	136	136	133	174	111	877	61
Pool	MAPE	0.056	0.041	**0.041**	0.051	0.209	0.230	0.287	0.321
	Time (s)	300	310	318	299	391	250	1969	156

radiance distribution, the lack of BSDF distribution approximation results in many more outliers, as shown in Fig. 6 (a).

Distribution Models. Compared to traditional distribution models such as spherical Gaussians, our NASG model is more expressive and achieves lower variance in most scenes, and this is especially the case for scenes with anisotropic lighting conditions, such as Ajar, Ring, and Glossy Tube. In Fig. 6 (b) we show an example of a scene where the dominant indirect lighting is anisotropic. Although Kent distribution is also a spherical anisotropic distribution, we have observed two limitations: First, the computationally expensive nature of Kent distribution makes it 5× slower than NASG. Second, due to the constraints imposed to mitigate the precision issue, Kent distribution can only yield a coarse approximation when the target distribution contains high-energy spots. As illustrated in Fig. 6 (a), the caustics emanating from the smooth floor are not accurately reconstructed when using Kent distribution. In addition the rendered images, several learned distributions with different models are compared in Fig. 7.

7.2 Performance

An importance sampling method is more practical when it achieves a lower level of error with the same time budget. We conducted an experiment to compare the equal-time performance of our framework with different

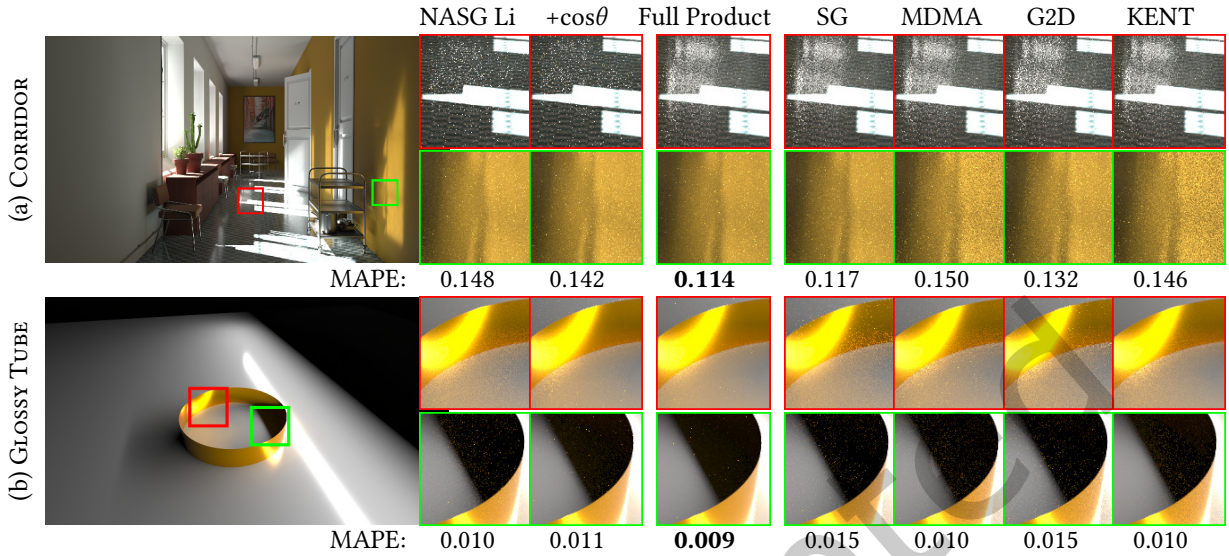

Fig. 6. A subset of rendering results in variance reduction comparison, rendered at 1024 samples per pixel (SPP). (a) Compared with incident radiance guiding, our full product guiding significantly improves results on specular surface: our "Full Product" result shows clear reflection of the metallic shelf, while "Li" or cosine weighted guiding only produces noise. MDMA and Kent distribution fail to learn an accurate distribution for narrow incident light, leading to higher variance. (b) Compared to traditional distributions such as SG, our NASG is more expressive and can learn a more accurate distribution for anisotropic lighting conditions, leading to lower variance.

configurations, as well as with plain path tracing, using the same set of test scenes as in our previous experiments. There are several meta-parameters that can influence performance and accuracy, including the maximum sample size (S), number of hidden units (HU), and number of components (C). In our default configuration, we set $S = 2^{16}$, HU = 128, and C = 8. We show the results under different configurations in Tab. 4. In general, the default configuration achieves a good balance between learning accuracy and sampling speed: it achieves the lowest MAPE in four scenes out of eight. Even in the scenes where it does not perform the best, its MAPE is close to the best results. With 32 components (C=32), the result of Box has the lowest MAPE, while the sample count is only 58% of our default configuration. Actually in a simple scene such as Box, where performance is not a bottleneck and all methods achieve significant number of samples, a more precise distribution can be learned with more components, resulting in a lower error. However, in complex scenes where higher variance is introduced by multiple light sources or complex materials, sufficient samples are required for convergence for Monte Carlo rendering. With C=32, sample count is low under limited computation budget, resulting in larger error.

Neural network overhead. In the experiment we measured the time cost of three parts of computation: forward execution for every shading point, training of the neural network, and path tracing. While changing through scenes, we found that typically 15% of the time is used for forward execution, 35% for training, and 50% for path tracing. The training overhead is almost constant among different image resolutions, which makes our framework relatively costly when rendering small images. For example, in the Bidir scene, which uses a small 512×512 resolution, the ratio of neural network overhead appears to be larger. From our experience, increasing the training steps (i.e., more training batches in each iteration) leads to higher accuracy in the learned distribution,

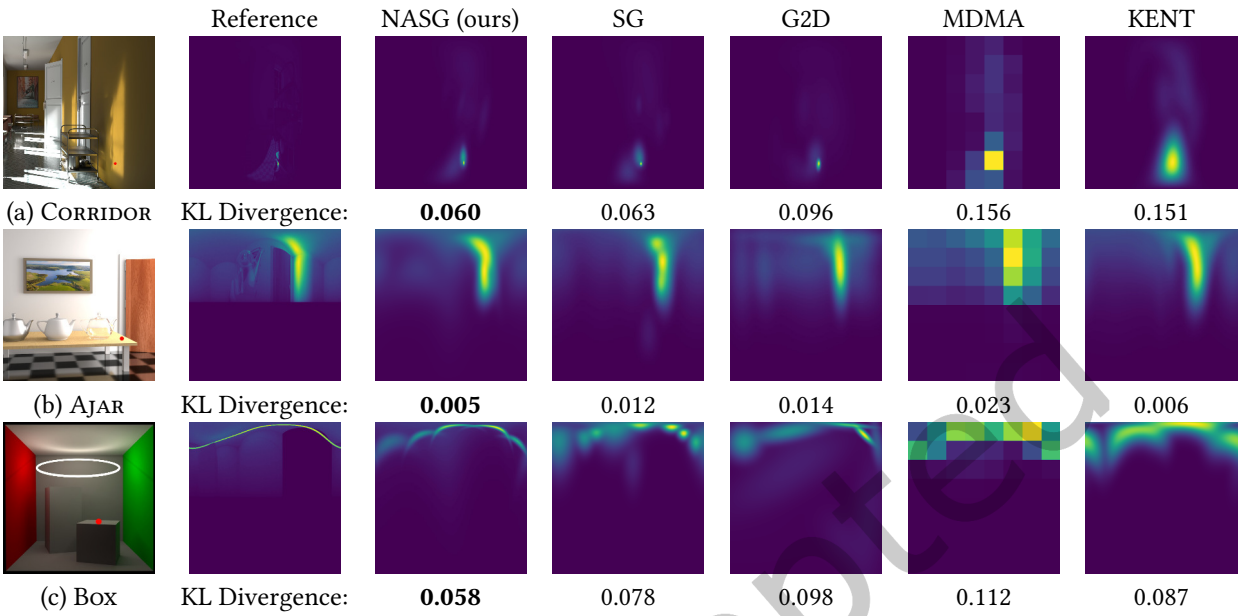

Fig. 7. Visualization images of learned distributions using different models, including 14-component spherical Gaussians (SG), 12-component 2D Gaussians (G2D), MDMA with 8 components for both dimensions, 8-component Kent distribution (KENT, due to the precision issue, we apply a constraint $0 < \kappa < 20$), and our 8-component NASG. Each row shows the learned distributions at the red-dotted locations. The reference is generated by rendering a panorama image at given locations, to which a cosine weight is applied. We measured the difference from the reference using KL-divergence (lower is better). (a) Our NASG achieves slightly more accurate distribution in an ordinary indoor indirect lighting scene. (b) Ajar scene features a vertical indirect lighting source, and our NASG can fit the shape well with fewer components. (c) The thin curve distribution in the Box scene is difficult to fit with isotropic models: the approximated curve is much thicker than the reference. In contrast, our NASG effectively preserves the thin nature of the curve, achieving lower KL-divergence. The Kent distribution fails to learn sharp distributions due the limited range of κ. However, raising the limit leads to a biased result or disrupted learning.

but with significant overhead. We conjecture that using meta-learning to accelerate the training process would be a promising direction, and leave this as a future work.

Network scale. Intuitively, a larger network, with either more weight parameters or more components, could achieve higher accuracy. However, as neural network computation is the primary overhead in our framework and complex scenes require a significant sample count to converge, it is still worthwhile to trade off accuracy for computation overhead given our limited time budget.

7.3 State-of-the-Art Comparison

To compare our framework with various state-of-the-art path guiding techniques, we implemented it on the Mitsuba renderer. In our Mitsuba implementation, we set $S = 2^{18}$ and $\nu = 4$ to better show the potential of our framework. The test scenes are shown in Fig. 8, and we rendered all of the images in this experiment with two Intel Xeon E5-2680v2 20-thread CPUs (40 threads in total). We compared the results of our framework with the authors' implementations of PPG [Müller 2019] and PAVMM [Ruppert et al. 2020], along with our

Table 4. Equal-time comparison results, using a fixed 300-second budget. We compare the rendering results of plain path tracing (PT) with our framework in its default configuration, as well as several different configurations: 4 components (C=4), 16 components (C=16), 32 components (C=32), 64 hidden units per network layer (HU=64), 256 hidden units per network layer (HU=256), and a larger sample size (S=2^{18}).

Scene	Metric	Default	C=4	C=16	C=32	HU=64	HU=256	S=2^{18}	PT
Ajar	SPP	1419	1449	1241	768	1590	1200	882	4800
	MAPE	**0.050**	0.050	0.050	0.052	0.054	0.054	0.059	0.075
Bathroom	SPP	800	816	756	559	870	650	595	1800
	MAPE	**0.082**	0.082	0.086	0.095	0.085	0.097	0.091	0.105
Bidir	SPP	2430	2471	2274	1372	2586	2231	1320	12300
	MAPE	0.053	0.051	0.053	0.054	0.056	**0.050**	0.055	0.114
Box	SPP	3318	3386	3480	1912	3600	2950	1225	18500
	MAPE	0.027	0.035	0.027	**0.024**	0.034	0.026	0.045	0.028
Corridor	SPP	1350	1377	1248	733	1455	980	753	3680
	MAPE	**0.106**	0.110	0.127	0.122	0.116	0.122	0.159	0.175
Kitchen	SPP	920	961	873	579	1000	706	715	2380
	MAPE	0.051	0.052	0.051	0.060	**0.050**	0.058	0.054	0.052
Pool	SPP	971	992	890	633	1100	880	730	1960
	MAPE	**0.042**	0.043	0.055	0.052	0.056	0.066	0.058	0.305
Glossy Tube	SPP	2120	2141	2035	1431	2643	1957	1605	5215
	MAPE	0.008	0.009	0.008	0.010	**0.008**	0.009	0.009	0.010

implementation of NIS [Müller et al. 2019]. For all techniques, we enabled NEE. To render with PPG, we disabled inverse-variance-based weighting and selection probability optimizaton, while enabling tree splatting using default configuration. For PAVMM, we used BSDF product sampling when available and cosine weight sampling otherwise (the authors' implementation requires specific scene structure, which conflicts with scene setups), and we set *maxSamplesPerLeafNode* to 16k, with all other options left as default. We implemented NIS following the original paper; based on our framework, we replaced the sampling-density evaluation component with 2-layer quadratic linear normalizing flow, and we added another MLP to learn selection probability. We used the encoded input of our NASG framework as extra features for the transform layer, and the feature dimensions are encoded with one-blob encoding (we used 64 bins). The training sample size and training step's configuration are aligned with our framework. The major difference of our implementation from that in the original paper is that we only use lightweight MLPs, at the same scale used for our framework, rather than ResNets. The source code of our framework and our NIS implementation are given in the supplemental material.

The rendering speed of of a CPU-based neural path guiding implementation is strongly affected by the actual implementation, so we mainly compared equal-sample results rather than raw performance.

7.3.1 Variance reduction of guiding method. To objectively evaluate the efficiency of different guiding algorithms, we first disabled orthogonal features and rendered scenes with our framework and other methods. To ensure a fair comparison, we used the same sampling parameters in all of the methods. The selection probability is set to the fixed fraction 0.5 for all methods. All of the methods have a 128-sample training budget and 384 samples for rendering (512 samples in total). We achieved this by clearing the rendering accumulation after the 128th sample, and we accumulated the results for the rest of the samples.

Fig. 9 reports statistical results and Fig. 10 shows the convergence behavior during the experiment. From the results, we can observe that each previous work has both pros and cons; consequently, one technique may perform very well in one kind of scene but poor in others (details discussed below). Our framework, on the other hand, while achieving overall low error also provides a more balanced performance in all test scenes.

Comparison with PPG. PPG uses quad-tree for directional representation, which can closely model distributions with multiple high-energy points. Hence, in the Torus scene it achieves low variance, while PAVMM struggles due to the limited components used. However, in the Maze scene, the parallax issue makes the PPG result very noisy, as shown in Fig. 9 (c). Parallax error occurs because PPG's spatial partition algorithm has difficulty learning from narrow and close incident radiance. Our framework learns the continuous spatial-varying distribution and does not suffer from such issues.

Comparison with PAVMM. PAVMM performs generally better than PPG. With its parallax-aware fitting method, it learns distributions in the Maze scene most accurately. However, PAVMM produces a rather noisy result in the Ajar scene, as shown in Fig. 9 (d). This is due to two reasons: First, the major illumination is a line-shaped lit surface on the wall at the right side of the scene (lit by light source outside the door via a small crack), which is anisotropic and difficult for vMF to fit. Second, the fitting algorithm of PAVMM encounters a challenge in modeling the indirect lighting from the specular checkbox floor. In contrast, our proposed framework exhibits greater robustness by leveraging NASG's ability to accurately capture the anisotropic distribution with a parsimonious set of components, thus achieving low variance.

Comparison with NIS. With the same configuration of training samples and steps, NIS in general achieves similar variance, except in the Maze scene (as shown in Fig. 9 (c), in which the learning of the 2D distribution suffers from discontinuity at the edge of the azimuth axis). Compared to the results reported in the original NIS paper [Müller et al. 2019], our NIS implementation has a slightly higher variance. We attribute this to two main factors: (a) Our implementation employs a lightweight 2-layer MLP-based transform as opposed to the original, relatively more complex, 4-layer ResNet-based version and (b) we limit our training to a subset of samples at each iteration. However, even with this relatively lightweight implementation, NIS takes roughly 2× longer than our framework: in Tab. 5 we compare the rendering time of NIS and our NASG framework. The extra overhead is due to the usage of coupled networks, and the normalizing flow needs to be evaluated two times at each non-delta shading point (one for scattering sampling and the other for NEE). The intrinsic characteristics of normalizing flow as an implicit density model, characterized by the utilization of multi-layer transformations, imply that a larger network architecture may be imperative to effectively capture complex distributions with accuracy. Moreover, it is plausible that a greater number of training samples or iterations might be essential to ensure convergence. These adjustments are likely requisite for attaining the results reported in the original paper. However, they also inherently bring a significant increase in computational overhead. In other words, with limited computation resources, our framework can learn more accurate distributions.

7.3.2 System comparison. Our neural path guiding framework integrates two features to further reduce sampling variance, namely, selection probability optimization and progressive sample re-weighting. PPG [Müller 2019] proposed similar features yet achieved them with different approaches. To compare the sampling efficiency of fully integrated systems, we render the above scenes with all features *enabled*. For PPG [Müller 2019] we additionally set the sample budget to 511 in order to benefit from inverse-variance-weighted blending. While PAVMM [Ruppert et al. 2020] does not implement similar features, similar improvement could be expected if the features were implemented with it.

As shown in Tab. 6, by enabling selection probability optimization and sample re-weighting, the MAPE of the results produced by both methods decreases in a generally varying range. PPG achieves a more obvious boost, which we attribute to its lack of product guiding: selection probability optimization thus has a stronger impact

Table 5. Rendering time (minutes) of proposed NASG framework and NIS implementation in Mitsuba. NIS uses a normalizing flow to model distribution, which takes roughly twice as much time to calculate, compared to our explicit NASG model.

Scene	NASG (ours)	NIS
Ajar	188.8m	520.4m
Bathroom	56.2m	166.8m
Bidir	35.6mm	89.6m
Kitchen	50.1m	163.5m
Maze	66.4m	84.42m
Box	12.4m	42.7m
Staircase	46.3m	217.2m
Torus	27.3m	109.2m
Wine	32.2m	128.4m

Table 6. MAPE of rendering results, with and without enabling selection probability optimization and sample re-weighting schemes, in PPG and proposed framework. A general improvement can be observed in most cases for both methods.

	PPG			NASG (ours)		
	Guiding	+ Features	Improvement (%)	Guiding	+ Features	Improvement (%)
Ajar	0.075	0.078	-2.83	0.052	0.050	4.77
Bathroom	0.066	0.060	8.17	0.058	0.060	-3.13
Bidir	0.066	0.061	7.84	0.047	0.042	10.73
Box	0.149	0.109	26.55	0.096	0.094	2.51
Kitchen	0.124	0.111	10.88	0.157	0.150	4.76
Maze	0.140	0.077	45.19	0.059	0.050	15.58
Staircase	0.064	0.055	14.04	0.050	0.048	4.16
Torus	0.028	0.025	12.19	0.024	0.024	-0.15
Wine	0.052	0.049	6.43	0.042	0.040	5.41

on reducing variance. While both methods are further improved by these features, it is also notable that our framework can optimize a selection probability almost for free, making our framework an attractive option when integrating it into a high-performance rendering system.

8 DISCUSSION

8.1 Performance in GPU Path Tracing

A GPU-based brute-force path tracer can be 20× faster than a CPU-based one. If an importance sampling method provides low per-sample variance yet high overhead, it cannot defeat a brute-force path tracer on GPU. As a network-based guiding method, the major overhead is the network itself, therefore, we must adopt a sampling strategy that can be implemented in a small network. This motivates us to find a solution for learning an explicit distribution, rather than an implicit model such as one based on a normalizing-flow. Despite its powerful computation ability, a GPU's large memory bandwidth can only be leveraged by carefully designed high-concurrency algorithms. This raises the necessity of reducing code branching as much as possible, which conflicts with the nature of path tracing. Otherwise, it suffers from a much more obvious memory latency than CPU, which causes many CPU algorithms to perform relatively slowly on the GPU (*e.g.,* PPG's quad-tree

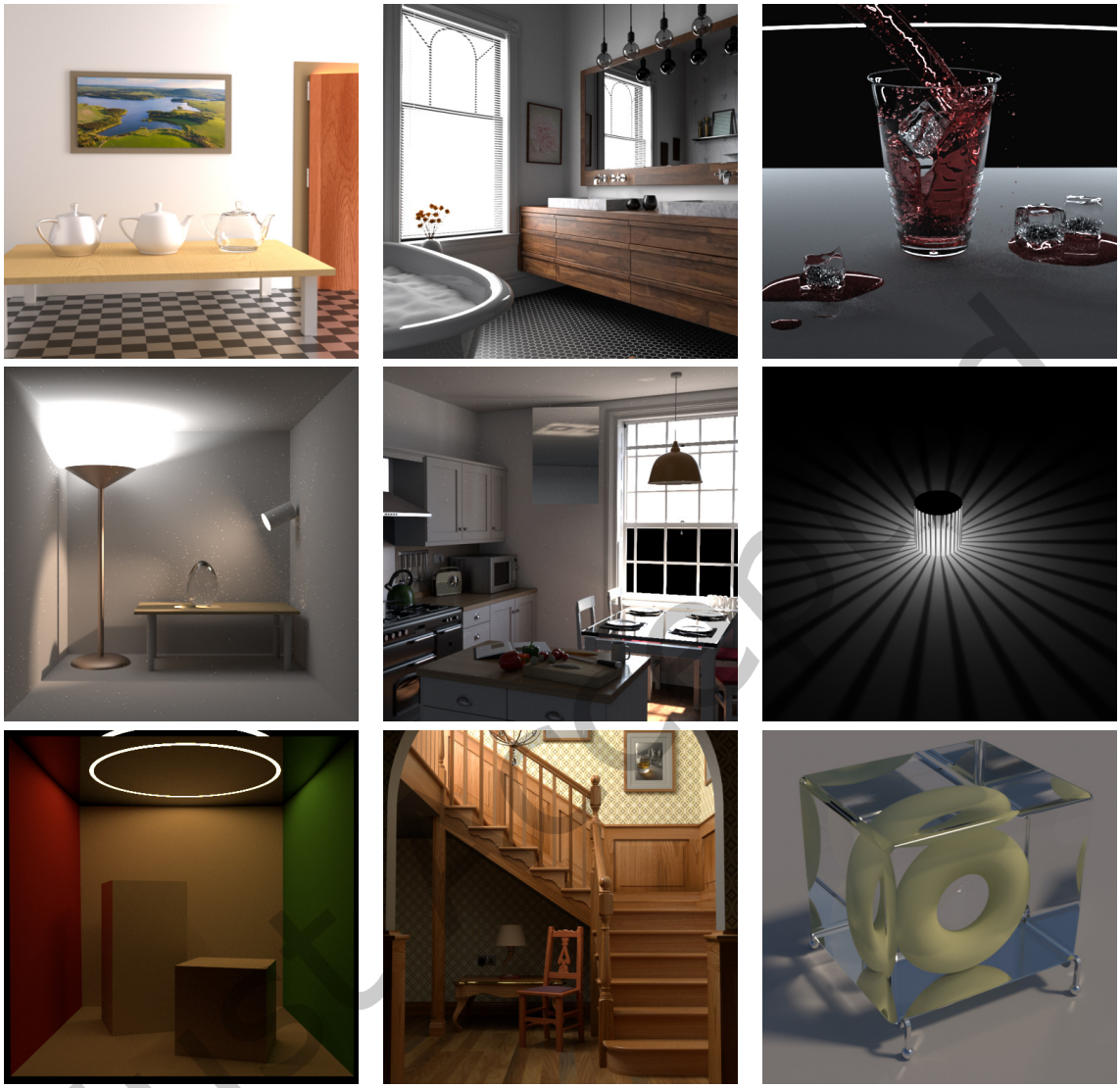

Fig. 8. Nine scenes used to evaluate proposed Mitsuba implementation with state-of-the-art techniques. From left to right: Ajar, Bathroom, Wine, Bidir, Kitchen, Maze, Box, Staircase, Torus. Several scenes apply different lighting setups for more indirect and anisotropic illumination.

approach requires multiple random accesses to GPU memory, thus causing long stalls). Recently, with hardware improvement, we see some acceleration hardware that is specifically designed for neural networks, which our framework could potentially benefit from.

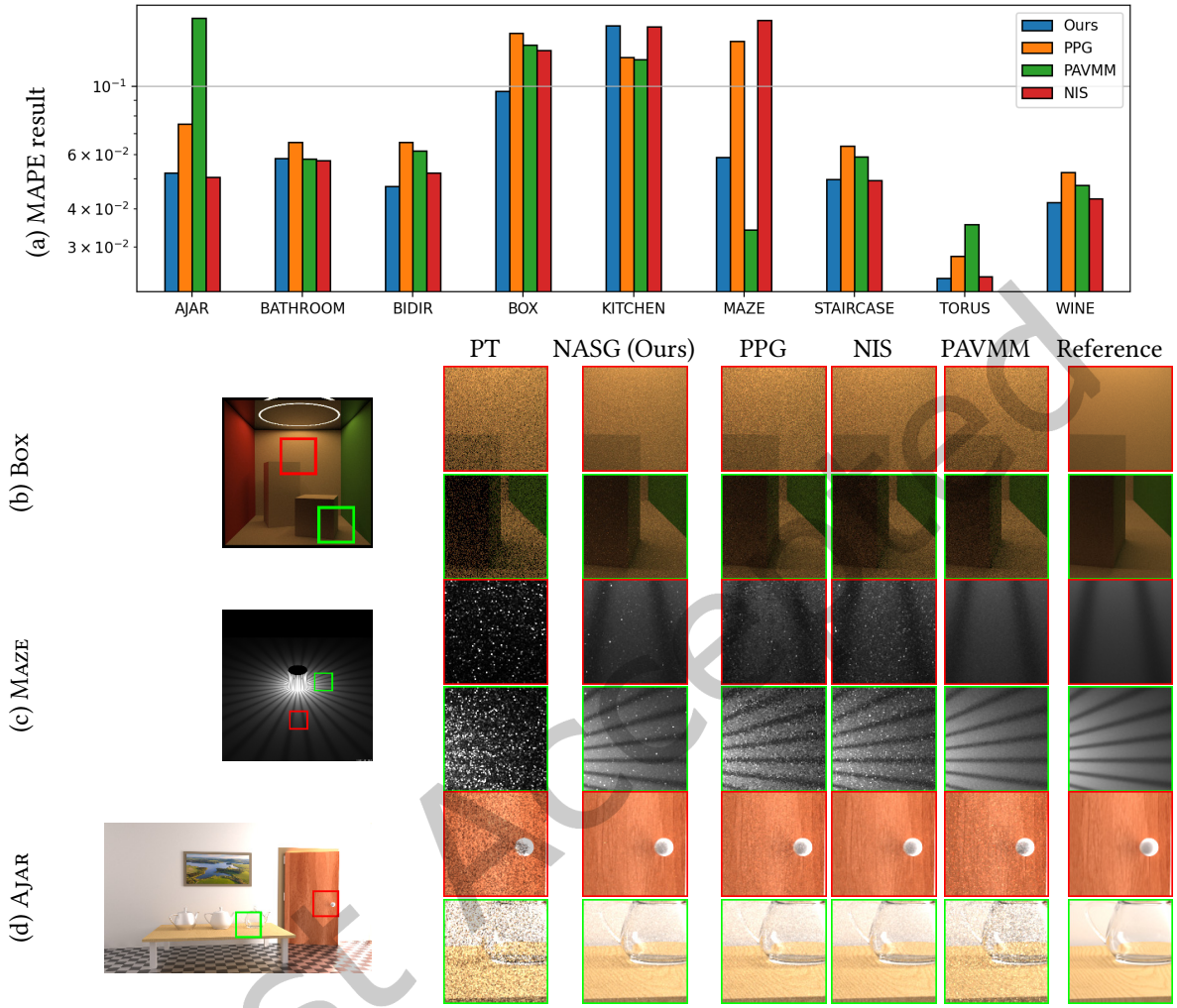

Fig. 9. Results of state-of-the-art comparison, visualized in logarithmic scale (lower is better). We render a variety of 3D scenes with 512 SPP, guided with different techniques including our framework, PPG [Müller 2019], PAVMM [Ruppert et al. 2020], and NIS [Müller et al. 2019]. Each technique uses the authors' implementations without modification except NIS. A subset of rendering results that demonstrates: (b) The Box scene, which features a circle luminary lighting the scene via a glass ceiling. The lighting is indirect and anisotropic, which is challenging for isotropic models such as vMF used in PAVMM. Our NASG helps to fit the distribution accurately and achieve lower variance. (c) The Maze scene with strong indirect lighting from a cage, which is challenging for PPG due to the parallax issue, while PAVMM achieves a clean result with its parallax-aware fitting algorithm, closely followed by ours. However, in (d) the Ajar scene, PAVMM has difficulty learning and fitting accurate distribution due to anisotropic indirect lighting and the near-specular floor, while our framework guides robustly. Although every method has its advantages in special circumstances, our framework performs well in general, making it an attractive option as a path-guiding technique.

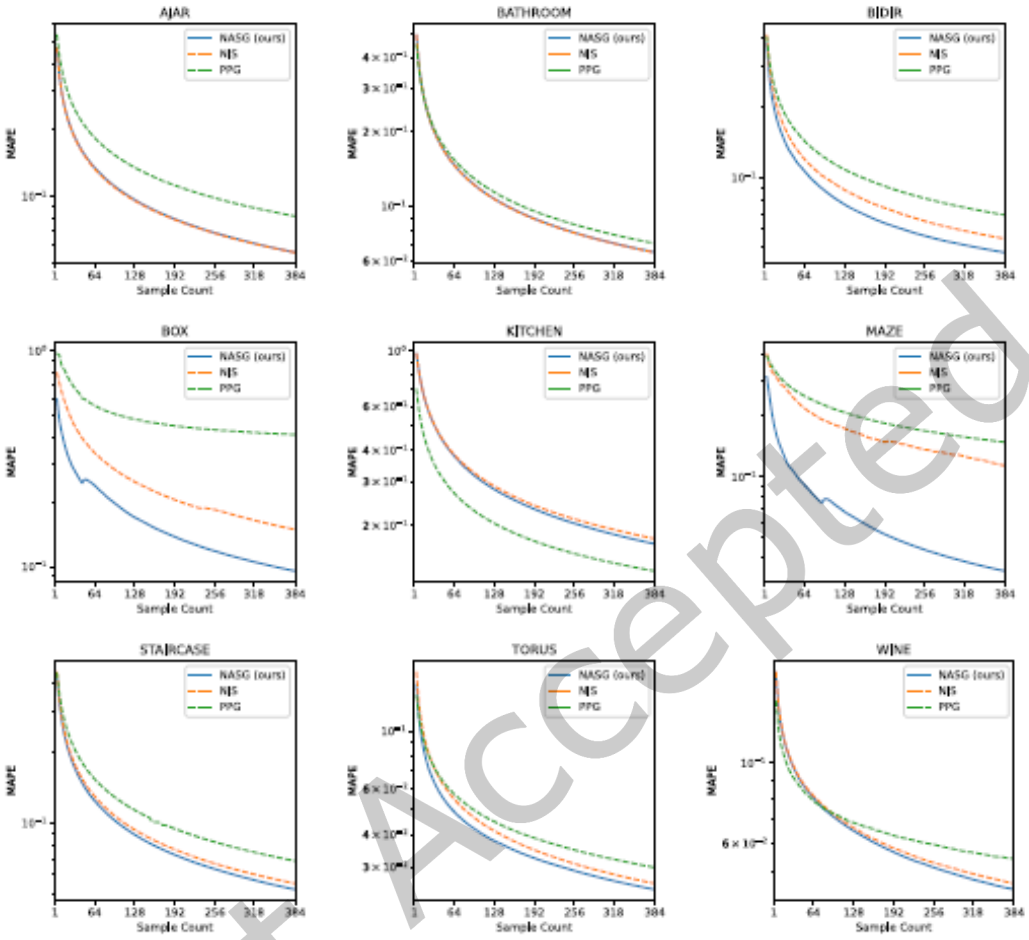

Fig. 10. Convergence graphs of all experiment scenes. PAVMM result is not reported due to a different mechanism of rendering. All of the techniques show similar convergence behavior due to the characteristics of importance sampling, while in general, our framework learns a more accurate distribution to achieve faster convergence. In Box and Maze, we observe jitter on NASG's convergence. This can be explained as an attempt to get out of the local minima by the optimizer, with the relatively large learning rate. In the Wine experiment, neural path guiding methods start from higher variance but quickly surpass PPG due to the progressive learning scheme.

8.2 Learning rate

In our GPU implementation, we used a relatively large learning rate (0.002). This decision was strategically aimed at enabling our model to swiftly navigate the parameter space, thus facilitating rapid convergence. This feature is particularly crucial in leveraging early samples to their fullest potential in our context. However, a large learning rate limits our model's ability to reach the deepest minima. That being said, at the latter stage of training, the resulting "jitters" remain close to the ground truth, ensuring that our learned distributions are valid approximations. We leave the improvement of optimization strategy as future work.

8.3 Limitations and Future Work

Although in general our configuration for training works well, sometimes we encounter cases where we need to use a smaller learning rate to avoid degenerated distributions. Such degenerated distributions have been observed only a few times when rendering caustics. They could be due to the complexity of the distribution, where only a small amount of light hits the point from a narrow direction. When the early samples based on BSDF fail to produce a representative initial distribution, the system could fail to sample important directions, which could eventually lead to degeneration. Adopting an adaptive learning strategy could possibly improve robustness - we leave this as a future work.

Another interesting extension of our framework could be path guiding in scenes with participating media. This could be achieved in theory (similar to what was done previously [Herholz et al. 2019]), yet the ability to learn 3D distributions remains an area to be further investigated.

9 CONCLUSION

In this paper we proposed an effective online neural path guiding framework for unbiased physically-based rendering. We tackled the major challenges of neural-based online path guiding methods by proposing a novel closed-form density model, NASG. The simplicity and expressiveness of NASG allow it to be efficiently trained online via a tiny, MLP-based neural network with spare ray samples.

Through experiments, we show that under the same sample budget, our framework outperforms existing neural path guiding techniques and achieves comparable results to state-of-the-art statistical path guiding techniques. With proper implementation on GPU, our framework helps to improve the raw performance of a GPU path tracer via neural path guiding. This was achieved because our framework can effectively learn the spatial-varying distribution and guide a unidirectional path tracer with low overhead, allowing the path tracer to produce high-quality images with limited computational resources. Our work also shows that learning-based importance sampling has great potential in practical rendering tasks. We hope our work can help pave the path for research communities in industry and academia to adopt neural technologies for path tracing.

ACKNOWLEDGEMENTS

This work was supported in part by JSPS KAKENHI Grant Number JP20K03551.

A CONTINUITY OF NASG

The complete form of a NASG component is given by:

$$
\begin{aligned}
&G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a, \epsilon, \kappa) \\
&\quad = \begin{cases} \kappa \cdot \exp\left(2\lambda \left(\frac{\mathbf{v}\cdot\boldsymbol{z}+1}{2}\right)^{1+\epsilon+\frac{a(\mathbf{v}\cdot\boldsymbol{x})^2}{1-(\mathbf{v}\cdot\boldsymbol{z})^2}} - 2\lambda\right)\left(\frac{\mathbf{v}\cdot\boldsymbol{z}+1}{2}\right)^{\epsilon+\frac{a(\mathbf{v}\cdot\boldsymbol{x})^2}{1-(\mathbf{v}\cdot\boldsymbol{z})^2}} & \text{if } \mathbf{v} \neq \pm\boldsymbol{z} \\ \kappa & \text{if } \mathbf{v} = \boldsymbol{z} \\ 0 & \text{if } \mathbf{v} = -\boldsymbol{z} \end{cases}
\end{aligned}
\tag{19}
$$

where κ is the lobe amplitude and ϵ is an auxiliary parameter introduced so that G is continuous whenever $\epsilon > 0$. In practice, it is possible to set $\epsilon = 0$, although this breaks continuity at $\mathbf{v} = -\boldsymbol{z}$ unless $a = 0$. For the rest of the content, we set $\kappa = 1$ and omit it from the notation.

B DERIVATION OF THE NORMALIZING TERM

Consider an NASG component defined as $G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a, \epsilon)$. Since G is rotation invariant, we assume that $\boldsymbol{x} = (1, 0, 0)$, $\boldsymbol{y} = (0, 1, 0)$, and $\boldsymbol{z} = (0, 0, 1)$. Then the surface integral of the function G over S^2 is given by

$$\begin{aligned}&\int_{S^2} G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a, \epsilon) d\omega \\ &= \int_0^{2\pi} d\phi \int_0^{\pi} \exp\left(2\lambda \left(\frac{\cos\theta + 1}{2}\right)^{1+\epsilon+a\cos^2\phi} - 2\lambda\right) \\ &\qquad\qquad \left(\frac{\cos\theta + 1}{2}\right)^{\epsilon+a\cos^2\phi} \sin\theta d\theta.\end{aligned} \tag{20}$$

We compute the inner integral over θ first. Consider the following change of variable (for fixed ϕ):

$$\begin{aligned}&\frac{\cos\theta + 1}{2} = \left(\frac{\cos\tau + 1}{2}\right)^{\frac{1}{1+\epsilon+a\cos^2\phi}} \\ \Leftrightarrow\ &\cos\theta = 2\left(\frac{\cos\tau + 1}{2}\right)^{\frac{1}{1+\epsilon+a\cos^2\phi}} - 1,\end{aligned} \tag{21}$$

where $0 \leqslant \tau \leqslant \pi$. Then

$$-\sin\theta d\theta = \frac{2}{1+\epsilon+a\cos^2\phi}\left(\frac{\cos\tau + 1}{2}\right)^{\frac{1}{1+\epsilon+a\cos^2\phi}-1}\left(-\frac{\sin\tau}{2}\right)d\tau, \tag{22}$$

so

$$\sin\theta d\theta = \frac{1}{1+\epsilon+a\cos^2\phi}\left(\frac{\cos\tau + 1}{2}\right)^{-\frac{\epsilon+a\cos^2\phi}{1+\epsilon+a\cos^2\phi}} \sin\tau d\tau, \tag{23}$$

and the inner integral with respect to θ becomes

$$\begin{aligned}&\frac{1}{1+\epsilon+a\cos^2\phi}\int_0^{\pi} \exp(\lambda\cos\tau - \lambda)\sin\tau d\tau \\ &= \frac{1}{1+\epsilon+a\cos^2\phi}\left[-\frac{\exp(\lambda\cos\tau - \lambda)}{\lambda}\right]_0^{\pi} \\ &= \frac{1 - e^{-2\lambda}}{\lambda(1+\epsilon+a\cos^2\phi)}.\end{aligned} \tag{24}$$

Therefore, it follows that

$$\begin{aligned}&\int_{S^2} G(\mathbf{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a, \epsilon) d\omega \\ &= \frac{1 - e^{-2\lambda}}{\lambda}\int_0^{2\pi} \frac{d\phi}{1+\epsilon+a\cos^2\phi} \\ &= \frac{2(1 - e^{-2\lambda})}{\lambda}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{d\phi}{1+\epsilon+a\cos^2\phi}.\end{aligned} \tag{25}$$

Now, consider the following change of variable:

$$\tan\phi = \sqrt{\frac{1+\epsilon+a}{1+\epsilon}}\tan\rho. \tag{26}$$

Then

$$\frac{d\phi}{\cos^2\phi} = \sqrt{\frac{1+\epsilon+a}{1+\epsilon}}\frac{d\rho}{\cos^2\rho}, \tag{27}$$

where

$$\begin{aligned}\cos^2\phi &= \frac{1}{1+\tan^2\phi}\\ &= \frac{1}{1+\frac{1+\epsilon+a}{1+\epsilon}\tan^2\rho}\\ &= \frac{1+\epsilon}{\frac{1+\epsilon+a}{\cos^2\rho}-a}\\ &= \frac{(1+\epsilon)\cos^2\rho}{1+\epsilon+a-a\cos^2\rho},\end{aligned} \tag{28}$$

so

$$d\phi = \frac{\sqrt{(1+\epsilon)(1+\epsilon+a)}}{1+\epsilon+a-a\cos^2\rho}d\rho. \tag{29}$$

Consequently, it follows that

$$\begin{aligned}&\int_{S^2} G(\boldsymbol{v};[\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}],\lambda,a,\epsilon)d\omega\\ &= \frac{2(1-e^{-2\lambda})}{\lambda}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}}\frac{1}{1+\epsilon+\frac{(1+\epsilon)a\cos^2\rho}{1+\epsilon+a-a\cos^2\rho}}\cdot\frac{\sqrt{(1+\epsilon)(1+\epsilon+a)}}{1+\epsilon+a-a\cos^2\rho}d\rho\\ &= \frac{2(1-e^{-2\lambda})}{\lambda}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}}\frac{d\rho}{\sqrt{(1+\epsilon)(1+\epsilon+a)}}\\ &= \frac{2\pi(1-e^{-2\lambda})}{\lambda\sqrt{(1+\epsilon)(1+\epsilon+a)}}.\end{aligned} \tag{30}$$

This expression reduces to Eq. (12) when $\epsilon = 0$.

C SAMPLING NASG

Here, we discuss how to sample from the distribution on S^2 obtained by normalizing G (cf. Eq. (30)). To this end, we essentially reverse the discussions in the previous section. Define two functions

$$\begin{aligned}&\Phi_E : [e^{-2\lambda},1]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right]\to[0,\pi]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right],\\ &\Phi_W : [e^{-2\lambda},1]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right]\to[0,\pi]\times\left[\frac{\pi}{2},\frac{3\pi}{2}\right]\end{aligned} \tag{31}$$

by

$$\begin{aligned}&\Phi_E(s,\rho)\\ &= \left(\arccos\left(2\left(\frac{\log s}{2\lambda}+1\right)^{\frac{1+\epsilon+a-a\cos^2\rho}{(1+\epsilon)(1+\epsilon+a)}}-1\right),\arctan\left(\sqrt{\frac{1+\epsilon+a}{1+\epsilon}}\tan\rho\right)\right),\\ &\Phi_W(s,\rho) = \Phi_E(s,\rho)+(0,\pi),\end{aligned} \tag{32}$$

where $(s,\rho)\in[e^{-2\lambda},1]\times[-\frac{\pi}{2},\frac{\pi}{2}]$ (here, "E" and "W" stand for east and west, respectively).

In practice, we just need to sample two uniform values $(\xi_0, \xi_1) \in [0,1]^2$ and linearly map them to $[e^{-2\lambda}, 1] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ to obtain (s, ρ) used in the above equation. We introduce another uniform random number $\xi_2 \in [0,1]$. When $\xi_2 > 0.5$, we sample the eastern hemisphere and $\Phi_E(s,\rho)$ is the sampled direction's (θ, ϕ). When $\xi_2 \leqslant 0.5$, we instead use $\Phi_W(s,\rho)$.

Let us now argue that the above sampling method serves our purpose. Observe that both Φ_E and Φ_W are bijections. Their inverses are given by

$$\begin{aligned}
&\Phi_E^{-1}(\theta,\phi) \\
&= \left(\exp\left(2\lambda\left(\frac{\cos\theta+1}{2}\right)^{1+\epsilon+a\cos^2\phi} - 2\lambda\right), \arctan\left(\sqrt{\frac{1+\epsilon}{1+\epsilon+a}}\tan\phi\right)\right), \\
&\Phi_W^{-1}(\theta,\phi) = \Phi_E^{-1}(\theta,\phi-\pi),
\end{aligned} \tag{33}$$

where $(\theta,\phi) \in [0,\pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ for the former, and $(\theta,\phi) \in [0,\pi] \times [\frac{\pi}{2}, \frac{3\pi}{2}]$ for the latter. The Jacobian for Φ_E^{-1} is computed as

$$\begin{aligned}
&J_{\Phi_E^{-1}}(\theta,\phi) \\
&= \exp\left(2\lambda\left(\frac{\cos\theta+1}{2}\right)^{1+\epsilon+a\cos^2\phi} - 2\lambda\right) \\
&\times 2\lambda(1+\epsilon+a\cos^2\phi)\left(\frac{\cos\theta+1}{2}\right)^{\epsilon+a\cos^2\phi}\left(-\frac{\sin\theta}{2}\right) \\
&\times \frac{1}{1+\frac{1+\epsilon}{1+\epsilon+a}\tan^2\phi}\sqrt{\frac{1+\epsilon}{1+\epsilon+a}}\frac{1}{\cos^2\phi} \\
&= \lambda\sqrt{(1+\epsilon)(1+\epsilon+a)}\exp\left(2\lambda\left(\frac{\cos\theta+1}{2}\right)^{1+\epsilon+a\cos^2\phi} - 2\lambda\right) \\
&\times \left(\frac{\cos\theta+1}{2}\right)^{\epsilon+a\cos^2\phi}(-\sin\theta).
\end{aligned} \tag{34}$$

Let X be a random variable on S^2, which we also view as a function of (θ,ϕ). Then, the expected value of X over the points $\Phi_E(s,\rho)$, where the (s,ρ) are uniformly sampled from $[e^{-2\lambda},1] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$, is given by

$$\mathbb{E}\left[X \circ \Phi_E\right] = \frac{1}{\pi(1-e^{-2\lambda})}\int_{[e^{-2\lambda},1]\times[-\frac{\pi}{2},\frac{\pi}{2}]} X \circ \Phi_E(s,\rho)\,ds\,d\rho. \tag{35}$$

By taking the change of variables $(s, \rho) = \Phi_E^{-1}(\theta, \phi)$, we have

$$\begin{aligned}
&\mathbb{E}\left[X \circ \Phi_E\right] \\
&= \frac{1}{\pi(1-e^{-2\lambda})} \int_{[0,\pi]\times[-\frac{\pi}{2},\frac{\pi}{2}]} X(\theta,\phi) \left| J_{\Phi_E^{-1}}(\theta,\phi) \right| d\theta d\phi \\
&= \frac{\lambda\sqrt{(1+\epsilon)(1+\epsilon+a)}}{\pi(1-e^{-2\lambda})} \\
&\int_{[0,\pi]\times[-\frac{\pi}{2},\frac{\pi}{2}]} X(\theta,\phi) \exp\left(2\lambda\left(\frac{\cos\theta+1}{2}\right)^{1+\epsilon+a\cos^2\phi} - 2\lambda\right) \\
&\qquad\qquad \left(\frac{\cos\theta+1}{2}\right)^{\epsilon+a\cos^2\phi} \sin\theta d\theta d\phi.
\end{aligned} \tag{36}$$

We also obtain a similar expression for $\mathbb{E}\left[X \circ \Phi_W\right]$. Since we choose each of the eastern and western hemispheres with probability $\frac{1}{2}$ according to the values of ξ_2, the overall expected value becomes

$$\begin{aligned}
&\frac{1}{2}\mathbb{E}\left[X \circ \Phi_E\right] + \frac{1}{2}\mathbb{E}\left[X \circ \Phi_W\right] \\
&= \frac{\lambda\sqrt{(1+\epsilon)(1+\epsilon+a)}}{2\pi(1-e^{-2\lambda})} \int_{S^2} X(\boldsymbol{v}) G(\boldsymbol{v}; [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \lambda, a, \epsilon) d\omega.
\end{aligned} \tag{37}$$

See Eq. (20). It follows that our sampled directions obey the distribution obtained by normalizing G, as desired.

REFERENCES

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting. *ACM Trans. Graph.* 39, 4, Article 148 (aug 2020), 17 pages. https://doi.org/10.1145/3386569.3392481

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (jul 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

Alejandro Conty Estevez and Christopher Kulla. 2018. Importance Sampling of Many Lights with Adaptive Tree Splitting. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 25 (aug 2018), 17 pages. https://doi.org/10.1145/3233305

Roc Ramon Currius, Dan Dolonius, Ulf Assarsson, and Erik Sintorn. 2020. Spherical Gaussian Light-field Textures for Fast Precomputed Global Illumination. *Computer Graphics Forum* (2020). https://doi.org/10.1111/cgf.13918

Stavros Diolatzis, Adrien Gruson, Wenzel Jakob, Derek Nowrouzezahrai, and George Drettakis. 2020. Practical Product Path Guiding Using Linearly Transformed Cosines. *In Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 39, 4 (July 2020).

Addis Dittebrandt, Johannes Hanika, and Carsten Dachsbacher. 2020. Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing. In *Eurographics Symposium on Rendering*.

Ana Dodik, Marios Papas, Cengiz Öztireli, and Thomas Müller. 2022. Path Guiding Using Spatio-Directional Mixture Models. *Computer Graphics Forum* (2022). https://doi.org/10.1111/cgf.14428

Honghao Dong, Guoping Wang, and Sheng Li. 2023. Neural Parametric Mixtures for Path Guiding. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 29, 10 pages. https://doi.org/10.1145/3588432.3591533

Dar Gilboa, Ari Pakman, and Thibault Vatter. 2021. Marginalizable Density Models. https://doi.org/10.48550/ARXIV.2106.04741

Johannes Hanika, Marc Droske, and Luca Fascione. 2015. Manifold Next Event Estimation. *Computer Graphics Forum* 34. https://doi.org/10.1111/cgf.12681

David Hart, Matt Pharr, Thomas Müller, Ward Lopes, Morgan McGuire, and Peter Shirley. 2020. Practical Product Sampling by Fitting and Composing Warps. *Computer Graphics Forum* 39, 4 (July 2020), 149–158. https://doi.org/10.1111/cgf.14060

Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. 2016. Real-Time Polygonal-Light Shading with Linearly Transformed Cosines. *ACM Trans. Graph.* 35, 4, Article 41 (jul 2016), 8 pages. https://doi.org/10.1145/2897824.2925895

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* 35, 4 (2016), 67–77. https://doi.org/10.1111/cgf.12950 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12950

Sebastian Herholz, Yangyang Zhao, Oskar Elek, Derek Nowrouzezahrai, Hendrik P. A. Lensch, and Jaroslav Křivánek. 2019. Volume Path Guiding Based on Zero-Variance Random Walk Theory. *ACM Trans. Graph.* 38, 3, Article 25 (jun 2019), 19 pages. https://doi.org/10.1145/3230635

Wojciech Jarosz, Nathan A. Carr, and Henrik Wann Jensen. 2009. Importance Sampling Spherical Harmonics. *Computer Graphics Forum* (2009). https://doi.org/10.1111/j.1467-8659.2009.01398.x

Henrik Wann Jensen. 1995. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques '95*, Patrick M. Hanrahan and Werner Purgathofer (Eds.). Springer Vienna, Vienna, 326–335.

James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (aug 1986), 143–150. https://doi.org/10.1145/15886.15902

John T. Kent. 1982. The Fisher-Bingham Distribution on the Sphere. *Journal of the Royal Statistical Society. Series B (Methodological)* 44, 1 (1982), 71–80.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. 2021. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 11 (nov 2021), 3964–3979. https://doi.org/10.1109/tpami.2020.2992934

Eric P. Lafortune and Yves D. Willems. 1995. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques '95 (Proceedings of the 6th Eurographics Workshop on Rendering)*. 11–20. https://doi.org/10.1007/978-3-7091-9430-0_2

Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. In *Proceedings of the 5th High-Performance Graphics Conference* (Anaheim, California) *(HPG '13)*. Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/2492045.2492060

Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. 2022. Generalized Resampled Importance Sampling: Foundations of ReSTIR. *ACM Trans. Graph.* 41, 4, Article 75 (jul 2022), 23 pages. https://doi.org/10.1145/3528223.3530158

Yashuai Lü, Libo Huang, Li Shen, and Zhiying Wang. 2017. Unleashing the Power of GPU for Physically-Based Rendering via Dynamic Ray Shuffling. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 560–573. https://doi.org/10.1145/3123939.3124532

Maxon. 2023. Redshift Renderer. https://www.maxon.net/en/redshift.

Daniel Meister, Jakub Boksansky, Michael Guthe, and Jiri Bittner. 2020. On Ray Reordering Techniques for Faster GPU Ray Tracing. In *Symposium on Interactive 3D Graphics and Games* (San Francisco, CA, USA) *(I3D '20)*. Association for Computing Machinery, New York, NY, USA, Article 13, 9 pages. https://doi.org/10.1145/3384382.3384534

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *ACM Trans. Graph.* 35, 4, Article 40 (jul 2016), 10 pages. https://doi.org/10.1145/2897824.2925936

Thomas Müller. 2019. Path Guiding in Production, Chapter 10. In *ACM SIGGRAPH 2019 Courses* (Los Angeles, California) *(SIGGRAPH '19)*. Association for Computing Machinery, New York, NY, USA, Article 18, 77 pages. https://doi.org/10.1145/3305366.3328091

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (jul 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Comput. Graph. Forum* 36, 4 (jul 2017), 91–100. https://doi.org/10.1111/cgf.13227

Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (oct 2019), 19 pages. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural Control Variates. *ACM Trans. Graph.* 39, 6, Article 243 (nov 2020), 19 pages. https://doi.org/10.1145/3414685.3417804

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4, Article 36 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

OTOY. 2023. Octane Renderer. https://home.otoy.com/render/octane-render/.

Yaobin Ouyang, Shiqiu Liu, Markus Kettunen, Matt Pharr, and Jacopo Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. *Computer Graphics Forum* (2021). https://doi.org/10.1111/cgf.14378

Jacopo Pantaleoni. 2020. Online path sampling control with progressive spatio-temporal filtering. *CoRR* abs/2005.07547 (2020). arXiv:2005.07547 https://arxiv.org/abs/2005.07547

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

Alexander Rath, Pascal Grittmann, Sebastian Herholz, Petr Vévoda, Philipp Slusallek, and Jaroslav Křivánek. 2020. Variance-Aware Path Guiding. *ACM Trans. Graph.* 39, 4, Article 151 (jul 2020), 12 pages. https://doi.org/10.1145/3386569.3392441

Lukas Ruppert, Sebastian Herholz, and Hendrik P. A. Lensch. 2020. Robust Fitting of Parallax-Aware Mixtures for Path Guiding. *ACM Trans. Graph.* 39, 4, Article 147 (aug 2020), 15 pages. https://doi.org/10.1145/3386569.3392421

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Trans. Graph.* 21, 3 (jul 2002), 527–536. https://doi.org/10.1145/566654.566612

Carlos Ureña, Marcos Fajardo, and Alan King. 2013. An Area-Preserving Parametrization for Spherical Rectangles. In *Proceedings of the Eurographics Symposium on Rendering* (Zaragoza, Spain) *(EGSR '13)*. Eurographics Association, Goslar, DEU, 59–66. https://doi.org/10.1111/cgf.12151

Eric Veach. 1998. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph. D. Dissertation. Stanford, CA, USA. Advisor(s) Guibas, Leonidas J. AAI9837162.

Void Dimensions. 2024. Clight Renderer. https://www.clightrender.com/.

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-Line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. Graph.* 33, 4, Article 101 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601203

Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. 2009. All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance. In *ACM SIGGRAPH Asia 2009 Papers* (Yokohama, Japan) *(SIGGRAPH Asia '09)*. Association for Computing Machinery, New York, NY, USA, Article 133, 10 pages. https://doi.org/10.1145/1661412.1618479

Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. 2013. Anisotropic Spherical Gaussians. *ACM Trans. Graph.* 32, 6, Article 209 (nov 2013), 11 pages. https://doi.org/10.1145/2508363.2508386

Shaokun Zheng, Zhiqian Zhou, Xin Chen, Difei Yan, Chuyan Zhang, Yuefeng Geng, Yan Gu, and Kun Xu. 2022. LuisaRender: A High-Performance Rendering Framework with Layered and Unified Interfaces on Stream Architectures. *ACM Trans. Graph.* 41, 6, Article 232 (nov 2022), 19 pages. https://doi.org/10.1145/3550454.3555463

Junqiu Zhu, Yaoyi Bai, Zilin Xu, Steve Bako, Edgar Velázquez-Armendáriz, Lu Wang, Pradeep Sen, Miloš Hašan, and Ling-Qi Yan. 2021a. Neural Complex Luminaires: Representation and Rendering. *ACM Trans. Graph.* 40, 4, Article 57 (jul 2021), 12 pages. https://doi.org/10.1145/3450626.3459798

Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. 2021b. Hierarchical Neural Reconstruction for Path Guiding Using Hybrid Path and Photon Samples. *ACM Trans. Graph.* 40, 4, Article 35 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459810