Neural Importance Sampling

THOMAS MÜLLER, Disney Research & ETH Zürich
BRIAN MCWILLIAMS, Disney Research
FABRICE ROUSSELLE, Disney Research
MARKUS GROSS, Disney Research & ETH Zürich
JAN NOVÁK, Disney Research

We propose to use deep neural networks for generating samples in Monte Carlo integration. Our work is based on non-linear independent components estimation (NICE), which we extend in numerous ways to improve performance and enable its application to integration problems. First, we introduce piecewise-polynomial coupling transforms that greatly increase the modeling power of individual coupling layers. Second, we propose to preprocess the inputs of neural networks using one-blob encoding, which stimulates localization of computation and improves inference. Third, we derive a gradient-descent-based optimization for the KL and the χ^2 divergence for the specific application of Monte Carlo integration with unnormalized stochastic estimates of the target distribution. Our approach enables fast and accurate inference and efficient sample generation independently of the dimensionality of the integration domain. We show its benefits on generating natural images and in two applications to light-transport simulation: first, we demonstrate learning of joint path-sampling densities in the primary sample space and importance sampling of multi-dimensional path prefixes thereof. Second, we use our technique to extract conditional directional densities driven by the product of incident illumination and the BSDF in the rendering equation, and we leverage the densities for path guiding. In all applications, our approach yields on-par or higher performance than competing techniques at equal sample count.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Ray tracing**; *Supervised learning by regression*; *Reinforcement learning*; • **Mathematics of computing** → *Sequential Monte Carlo methods*.

1 INTRODUCTION

Solving integrals is a fundamental problem of calculus that appears in many disciplines of science and engineering. Since closed-form antiderivatives exist only for elementary problems, many applications resort to numerical recipes. Monte Carlo (MC) integration is one such approach, which relies on sampling a number of points within the integration domain and averaging the integrand thereof.

Authors' addresses: Thomas Müller, Disney Research & ETH Zürich, thomas94@gmx.net; Brian McWilliams, Disney Research, bvpmcwilliams@gmail.com; Fabrice Rousselle, Disney Research, fabrice.rousselle@gmail.com; Markus Gross, Disney Research & ETH Zürich, grossm@inf.ethz.ch; Jan Novák, Disney Research, novakj4@gmail.com.

The main drawback of MC methods is the relatively low convergence rate. Many techniques have thus been developed to reduce the integration error, e.g. via importance sampling, control variates, Markov chains, integration on multiple accuracy levels, and use of quasi-random numbers.

In this work, we focus on the concept of importance sampling and propose to parameterize the sampling density using a collection of neural networks. Generative neural networks have been successfully leveraged in many fields, including signal processing, variational inference, and probabilistic modeling, but their application to MC integration—in the form of sampling densities—remains to be investigated; this is what we strive for in the present paper.

Given an integral

$$F = \int_{\mathcal{D}} f(x)\, dx, \tag{1}$$

we can introduce a probability density function (PDF) $q(x)$, which, under certain constraints, allows expressing F as the expected ratio of the integrand and the PDF:

$$F = \int_{\mathcal{D}} \frac{f(x)}{q(x)} q(x)\, dx = \mathbb{E}\left[\frac{f(X)}{q(X)}\right]. \tag{2}$$

The above expectation can be approximated using N independent, randomly chosen points $\{X_1, X_2, \cdots X_N\}; X_i \in \mathcal{D}, X_i \sim q(x)$, with the following MC estimator:

$$F \approx \langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i)}. \tag{3}$$

The variance of the estimator, besides being inversely proportional to N, heavily depends on the shape of q. If q follows normalized f closely, the variance is low. If the shapes of the two differ significantly, the variance tends to be high. In the special case when samples are drawn from a PDF proportional to $f(x)$, i.e. $p(x) \equiv f(x)/F$, we obtain a zero-variance estimator, $\langle F \rangle_N = F$, for any $N \geq 1$.

It is thus crucial to use *expressive* sampling densities that match the shape of the integrand well. Additionally, generating sample X_i must be *fast* (relative to the cost of evaluating f), and *invertible*. That is, given a sample X_i, we require an efficient and exact evaluation of its corresponding probability density $q(X_i)$—a necessity for evaluating the unbiased estimator of Equation (3) . Being expressive, fast to evaluate, and invertible are the key properties of good sampling densities, and all our design decisions can be traced back to these.

We focus on the general setting where little to no prior knowledge about f is given, but f can be observed at a sufficiently high number of points. Our goal is to extract the sampling PDF from these observations while handling complex distributions with possibly many modes and arbitrary frequencies. To that end, we approximate

the ground-truth $p(x)$ using a generative probabilistic parametric model $q(x; \theta)$ that utilizes deep neural networks.

Our work builds on approaches that are capable of compactly representing complex manifolds in high-dimensional spaces, and permit fast and exact inference, sampling, and PDF evaluation. We extend the work of Dinh et al. [2014, 2016] on learning stably invertible transformations, represented by so-called *coupling layers*, that are stacked to produce highly nonlinear mappings between an observation x and a latent variable z. Specifically, we introduce two types of *piecewise-polynomial* coupling layers—piecewise-linear and piecewise-quadratic—that greatly increase the expressive power of individual coupling layers, allowing us to employ fewer of those and thereby reduce the total cost of evaluation.

After reviewing related work on generative neural networks in Section 2, we detail the framework of *non-linear independent components estimation* (NICE) [Dinh et al. 2014, 2016] in Section 3; this forms the foundation of our approach.

In Section 4, we describe a new class of *invertible piecewise-polynomial* coupling transforms that replace affine transforms proposed in the original work. We also introduce the *one-blob*-encoding of network inputs, which stimulates localization of computation and improves inference. We illustrate the benefits on low-dimensional regression problems and test the performance when learning a (high-dimensional) distribution of natural images.

In Section 5, we apply NICE to Monte Carlo integration and propose an optimization strategy for minimizing estimation variance.

Finally in Section 6, we present the benefits of using NICE in light-transport simulations. We use NICE with our polynomial warps to guide the construction of light paths and demonstrate that, while currently being impractical due to large computational overhead, it outperforms the state of the art at equal sample counts in *path guiding* and primary-sample-space *path sampling*. We combine our path-guiding distribution with other sampling distributions using multiple importance sampling (MIS) [Veach and Guibas 1995] and use a dedicated network to learn the approximately optimal selection probabilities to further improve the MIS performance.

In summary, our contributions are

- two *piecewise-polynomial* coupling transforms (piecewise-linear and piecewise-quadratic) that improve expressiveness,
- *one-blob*-encoded network inputs—a generalization of one-hot encoding—for improving learning speed and quality,
- stochastic gradients that can be used for optimizing the KL and χ^2 divergences when only MC estimates of the unnormalized target distribution are available, and
- an application of NICE with the aforementioned tools to the problem of light-transport simulation with
- data-driven probabilities for selecting sampling techniques.

2 BACKGROUND AND RELATED WORK

Since our goal is to learn a probability distribution and—among other things—draw samples from it, our approach falls into the category of *generative modeling*. In the following, we review the most relevant generative neural networks focusing on the requirements for MC integration. Specifically, we seek a model that provides a parametric PDF $q(x; \theta)$ for approximating the ideal PDF $p(x) \equiv f(x)/F$. We

also need an optimization scheme that tolerates noisy estimates of the integrand $f(x)$. Lastly, the trained model must permit efficient sampling and evaluation of $q(x; \theta)$.

Several prior generative models were built on less stringent requirements. For example, it is often the case that only the *synthesis* of samples is required without explicitly evaluating $q(x; \theta)$ [Germain et al. 2015; Goodfellow et al. 2014; van den Oord et al. 2016a,b]. These models are thus not easily applicable to MC integration in the aforementioned manner. In the following, we focus on existing techniques that show promise in satisfying our requirements.

The Latent-Variable Model. Many existing generative models rely on auxiliary unobserved "latent" variables z with fixed, prescribed PDF $q(z)$, where each possible value of z gives rise to a unique conditional distribution $q(x|z; \theta)$ that is learnable via parameters θ. Since any particular value of x can be caused by multiple different values of z, one must resort to integration to obtain $q(x; \theta)$

$$q(x; \theta) = \int q(x|z; \theta) q(z) \, dz \,. \tag{4}$$

In this context, $q(x; \theta)$ is referred to as the "marginal likelihood" and $q(z)$ as the "prior distribution". The prior is often modeled as a simple distribution (e.g. Gaussian or uniform) and it is the task of a neural network to learn the parameters of $q(x|z; \theta)$.

Unfortunately, the above integral is often not solvable in closed form, necessitating its estimation with another MC estimator. It may be tempting to use such estimates of $q(x; \theta)$ in the denominator of our desired MC estimator

$$\langle F \rangle_N \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{\langle q(X_i) \rangle} \,,$$

but this scheme introduces undesirable bias[1].

Normalizing Flows. To avoid the limitation of having to estimate $q(x; \theta)$, a growing literature emerged that models x as a deterministic bijective mapping of z, a so-called "normalizing flow" $x = h(z; \theta)$. This has the effect of assigning only a *single* value of x to each possible value of z (and vice versa), thereby avoiding the difficult integration when computing the likelihood. Mathematically, $q(x|z; \theta)$ becomes a Dirac-delta function $\delta(x - h(z; \theta))$, resulting in

$$q(x; \theta) = \int \delta(x - h(z; \theta)) q(z) \, dz = q(z) \left| \det\left(\frac{\partial h(z; \theta)}{\partial z^T}\right) \right|^{-1} . \tag{5}$$

Here, the inverse Jacobian determinant accounts for the change in density due to h in the infinitesimal neighborhood around x.

Although Equation (5) no longer contains a difficult integral, there exist a number of additional requirements on h to make the usage of $q(x; \theta)$ in MC integration practical. Since $q(x; \theta)$ is expressed in terms of z, one must know the value of z that corresponds to x. Generally, this requires a tractable inverse $z = h^{-1}(x)$.[2] Additionally, to ensure efficiency, the evaluation of both h and h^{-1} as well as the corresponding Jacobian determinant must be *fast* relative to the cost of evaluating f.

---

[1]This can be verified using Jensen's inequality.

[2]MC estimators that only sample from $q(x; \theta)$ are an exception, because they can simply use the z that generated x. However, when combining multiple PDFs using MIS heuristics [Veach and Guibas 1995] one must be able to evaluate $q(x; \theta)$ for x that were drawn from the other distributions.

Prior Work on Normalizing Flows. In the following, we mention a number of existing techniques based on normalizing flows that satisfy some (but not necessarily all) of our requirements.

Rezende and Mohamed [2015] model the posterior distribution using normalizing flows to perform variational inference more effectively. Unfortunately, their computation graph is difficult to invert, not permitting exact evaluation of $q(x; \theta)$.

Chen et al. [2018] propose a continuous analog of normalizing flows that utilizes the *instantaneous* change-of-variable formula, which only requires computing the *trace* of the Jacobian as opposed to the determinant. This reduces computational cost in many situations. Unfortunately, the evaluation of their continuous normalizing flows relies on a numeric ODE solver, which reintroduces computational cost in other places and results in approximation error when computing $q(x; \theta)$. This approximation causes bias in our use case of MC integration and therefore disqualifies their approach.

A number of recent approaches investigate the usage of normalizing flows for auto-regressive density estimation [Huang et al. 2018; Kingma et al. 2016; Papamakarios et al. 2017]. These "autoregressive flows" offer the desired exact evaluation of $q(x; \theta)$. Unfortunately, they generally only permit *either* efficient sample generation *or* efficient evaluation of $q(x; \theta)$, which makes them prohibitively expensive for our application to MC integration.

Lastly, "non-linear independent components estimation" (NICE) [Dinh et al. 2014, 2016] is a special case of autoregressive flows that allows *both* fast sampling *and* density evaluation through the use of so-called "coupling layers", the composition of which constitutes a normalizing flow. Because of NICE's efficient sample generation and its efficient, exact density evaluation, NICE satisfies all our postulated requirements for usage in MC integration and we therefore base our work on it. Concurrently with us, Zheng and Zwicker [2018] also investigate the applicability of NICE to MC integration.

Kingma and Dhariwal [2018] extend the work of Dinh et al. [2016] with invertible 1×1 convolutions, achieving better results. The 1×1 convolutions, however, require a deep computation graph to be effective, which is oppsite to the shallow computation graph we desire for efficiency. We therefore do not adopt the 1×1 convolutions.

In the following section, we introduce NICE in detail and proceed with describing our piecewise-polynomial coupling layers that increase its modeling capacity.

3 NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

In this section, we detail the works of Dinh et al. [2014, 2016] that form the basis of our approach. The authors propose to learn a mapping between the data and the latent space as an invertible compound function $\widehat{h} = h_L \circ \cdots \circ h_2 \circ h_1$, where each h_i is a relatively simple bijective transformation (warp). The choice of the type of h is different in the two prior works and in our paper (details follow in Section 4), but the key design principle remains: h needs to be stably invertible with (computationally) tractable Jacobians. This enables exact and fast inference of latent variables as well as exact and fast probability density evaluation: given a differentiable mapping $h : \mathcal{X} \to \mathcal{Y}$ of points $x \sim p_\mathcal{X}(x)$ to points $y \in \mathcal{Y}$, we can compute the PDF $p_\mathcal{Y}(y)$ of transformed points $y = h(x)$ using the

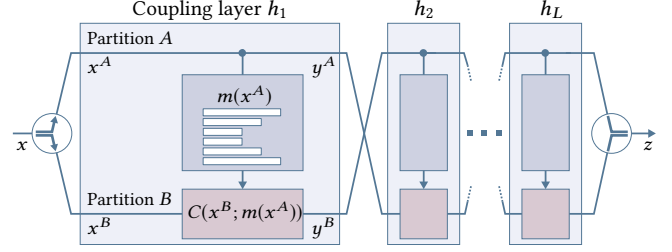Fig. 1. A coupling layer splits the input x into two partitions A and B. One partition is left untouched, whereas dimensions in the other partition are warped using a parametric coupling transform C driven by the output of a neural network m. Multiple coupling layers may need to be compounded to achieve truly expressive transforms.

change-of-variables formula:

$$p_\mathcal{Y}(y) = p_\mathcal{X}(x) \left| \det \left(\frac{\partial h(x)}{\partial x^T} \right) \right|^{-1}, \tag{6}$$

where $\frac{\partial h(x)}{\partial x^T}$ is the Jacobian of h at x.

The cost of computing the determinant grows superlinearly with the dimensionality of the Jacobian. If $\mathcal{X}$ and $\mathcal{Y}$ are high-dimensional, computing $p_\mathcal{Y}(y)$ is therefore computationally intractable. The key proposition of Dinh et al. [2014] is to focus on a specific class of mappings—referred to as *coupling layers*—that admit Jacobian matrices where determinants reduce to the product of diagonal terms.

3.1 Coupling Layers

A single coupling layer takes a D-dimensional vector and partitions its dimensions into two groups. It leaves the first group intact and uses it to parameterize the transformation of the second group.

Definition 3.1 (Coupling layer). Let $x \in \mathbb{R}^D$ be an input vector, A and B denote disjoint partitions of $[\![1, D]\!]$, and m be a function on $\mathbb{R}^{|A|}$, then the output of a coupling layer $y = (y^A, y^B) = h(x)$ is defined as

$$y^A = x^A, \tag{7}$$
$$y^B = C(x^B; m(x^A)), \tag{8}$$

where the *coupling transform* $C : \mathbb{R}^{|B|} \times m(\mathbb{R}^{|A|}) \to \mathbb{R}^{|B|}$ is a separable and invertible map.

The invertibility of the coupling transform, and the fact that partition A remains unchanged, enables a trivial inversion of the coupling layer $x = h^{-1}(y)$ as:

$$x^A = y^A, \tag{9}$$
$$x^B = C^{-1}(y^B; m(x^A)) = C^{-1}(y^B; m(y^A)). \tag{10}$$

If partition A was allowed to change arbitrarily, then the inversion (precisely the input to m in Equation (10)) would be difficult to find. The invertibility is crucial in our setting as we require both density evaluation and sample generation in Monte Carlo integration.

The second important property of C is separability. Separable C ensures that the Jacobian matrix is triangular and the determinant reduces to the product of diagonal terms; see Dinh et al. [2014] or

Appendix A for a full treatment. The computation of the determinant thus scales linearly with D and is therefore tractable even in high-dimensional problems.

3.2 Affine Coupling Transforms

Additive Coupling Transform. Dinh et al. [2014] describe a very simple coupling transform that merely translates the signal in individual dimensions of B:

$$C(x^B; t) = x^B + t, \qquad (11)$$

where the translation vector $t \in \mathbb{R}^{|B|}$ is produced by function $m(x^A)$.

Multiply-add Coupling Transform. Since additive coupling layers have unit Jacobian determinants, i.e. they preserve volume, Dinh et al. [2016] propose to add a multiplicative factor e^s:

$$C(x^B; s, t) = x^B \odot e^s + t, \qquad (12)$$

where $\odot$ represents element-wise multiplication and vectors t and $s \in \mathbb{R}^{|B|}$ are produced by m: $(s, t) = m(x^A)$. The Jacobian determinant of a multiply-add coupling layer is simply $e^{\sum s_i}$.

The coupling transforms above are relatively simple. The trick that enables learning *nonlinear* dependencies across partitions is the parametric function m. This function can be arbitrarily complex, e.g. a neural network, as we do not need its inverse to invert the coupling layer and its Jacobian does not affect the determinant of the coupling layer (cf. Appendix A). Using a sophisticated m allows extracting complex nonlinear relations between the two partitions. The coupling transform C, however, remains simple, invertible, and permits tractable computation of determinants even in high-dimensional settings.

3.3 Compounding Multiple Coupling Layers

As mentioned initially, the complete transform between the data space and the latent space is obtained by chaining a number of coupling layers. A different instance of neural network m is trained for each coupling layer. To ensure that all dimensions can be modified, the output of one layer is fed into the next layer with the roles of the two partitions swapped; see Figure 1. Compounding two coupling layers in this manner ensures that every dimension can be altered.

The number of coupling layers required to ensure that each dimension can influence every other dimension depends on the total number of dimensions. For instance, in a 2D setting (where each partition contains exactly one dimension) we need only two coupling layers. 3D problems require three layers, and for any high-dimensional configuration there must be at least four coupling layers.

In practice, however, high-dimensional problems (e.g. generating images of faces), require significantly more coupling layers since each affine transform is fairly limited. In the next section, we address this limitation by providing more expressive mappings that allow reducing the number of coupling layers and thereby the sample-generation and density-evaluation costs. This improves the performance of Monte Carlo estimators presented in Section 6.

4 PIECEWISE-POLYNOMIAL COUPLING LAYERS

In this section, we propose piecewise-polynomial invertible maps as coupling transforms instead of the limited affine warps reviewed
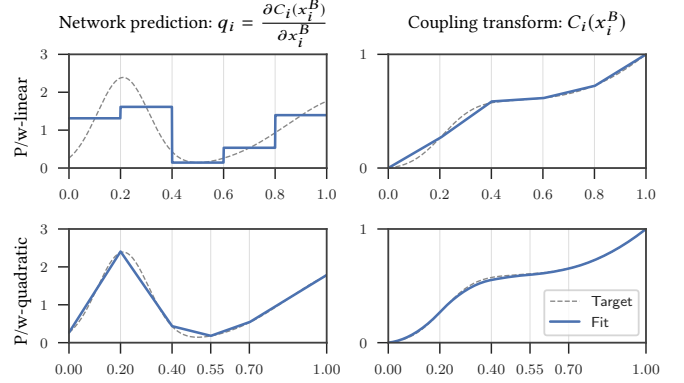
Fig. 2. Predicted probability density functions (PDFs, left) and corresponding cumulative distribution functions (CDFs, right) with $K = 5$ bins fitted to a target distribution (dashed). The top row illustrates a piecewise-linear CDF and the bottom row a piecewise-quadratic CDF. The piecewise-quadratic approximation tends to work better in practice thanks to its first-order continuity (C^1) and adaptive bin sizing. In Appendix B we show that, in contrast to piecewise-quadratic CDFs, adaptive bin sizing is difficult to achieve for piecewise-linear CDFs with gradient-based optimization methods.

previously. Specifically, we introduce the usage of piecewise polynomials with degrees 1 and 2, i.e. piecewise-linear and piecewise-quadratic warps. In contrast to Dinh et al. [2014, 2016], who assume $x, y \in (-\infty, +\infty)^D$ and Gaussian latent variables, we choose to operate in a unit hypercube (i.e. $x, y \in [0, 1]^D$) with uniformly distributed latent variables, as most practical problems span a finite domain. Unbounded domains can still be handled by warping the input of h_1 and the output of h_L e.g. using the sigmoid and logit functions.

Similarly to Dinh and colleagues, we ensure computationally tractable Jacobians via separability. We transform each dimension independently:

$$C(x^B; m(x^A)) = \left(C_1(x_1^B; m(x^A)), \cdots, C_{|B|}(x_{|B|}^B; m(x^A)) \right)^T. \qquad (13)$$

Operating on unit intervals allows interpreting the warping function C_i as a cumulative distribution function (CDF). To produce each C_i, we instrument the neural network to output the corresponding unnormalized probability density q_i, and construct C_i by integration; see Figure 2 for an illustration.

In order to further improve performance, we propose to encode the inputs to the neural network using *one-blob encoding*, which we discuss in Section 4.3.

4.1 Piecewise-Linear Coupling Transform

Motivated by their simplicity, we begin by investigating the simplest continuous piecewise-polynomial coupling transforms: piecewise-linear ones. Recall that we partition the D-dimensional input vector in two disjoint groups, A and B, such that $x = (x^A, x^B)$. We divide the unit dimensions in partition B into K bins of equal width $w = K^{-1}$. To define all $|B|$ transforms at once, we instrument the network $m(x^A)$ to predict a $|B| \times K$ matrix, denoted $\widehat{Q}$. Each i-th row of $\widehat{Q}$ defines the unnormalized probability mass function (PMF) of the
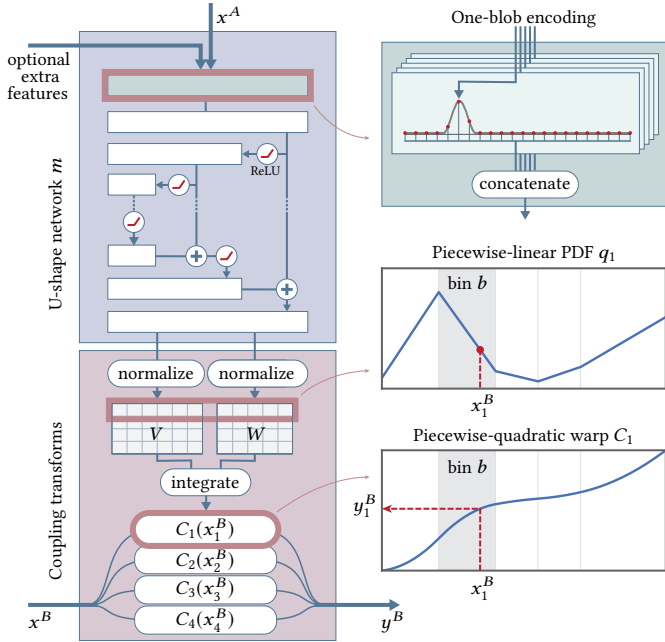
Fig. 3. Our coupling layer with a piecewise-quadratic transform for $|B| = 4$. Signals in partition A (and additional features) are encoded using one-blob encoding and fed into a U-shape neural network m with fully connected layers. The outputs of m are normalized yielding matrices V and W that define warping PDFs. The PDFs are integrated analytically to obtain piecewise-quadratic coupling transforms; one for warping each dimension of x^B.

warp in i-th dimension in x^B; we normalize the rows using the softmax function σ and denote the normalized matrix Q; $Q_i = \sigma(\widehat{Q}_i)$.

The PDF in i-th dimension is then defined as $q_i(x_i^B) = Q_{ib}/w$, where $b = \lfloor Kx_i^B \rfloor$ is the bin that contains the scalar value x_i^B. We integrate the PDF to obtain our invertible piecewise-linear warp C_i:

$$C_i(x_i^B; Q) = \int_0^{x_i^B} q_i(t)\,dt = \alpha Q_{ib} + \sum_{k=1}^{b-1} Q_{ik}, \qquad (14)$$

where $\alpha = Kx_i^B - \lfloor Kx_i^B \rfloor$ represents the relative position of x_i^B in b.

In order to evaluate the change of density resulting from the coupling layer, we need to compute the determinant of its Jacobian matrix; see Equation (6). Since $C(x^B; Q)$ is separable by definition, its Jacobian matrix is diagonal and the determinant is equal to the product of the diagonal terms. These can be computed using Q:

$$\det\left(\frac{\partial C(x^B; Q)}{\partial (x^B)^T}\right) = \prod_{i=1}^{|B|} q_i(x_i^B) = \prod_{i=1}^{|B|} \frac{Q_{ib}}{w}, \qquad (15)$$

where b again denotes the bin containing the value in the i-th dimension. To reduce the number of bins K required for a good fit we would like the network to also predict bin widths. These can unfortunately *not* easily be optimized with gradient descent in the piecewise-*linear* case; see Appendix B. To address this, and to improve accuracy, we propose piecewise-*quadratic* coupling transforms.

4.2 Piecewise-Quadratic Coupling Transform

Piecewise-quadratic coupling transforms admit a piecewise-linear PDF, which we model using $K + 1$ vertices; see Figure 2, bottom left. We store their vertical coordinates (for all dimensions in B) in $|B| \times (K + 1)$ matrix V, and horizontal differences between neighboring vertices (bin widths) in $|B| \times K$ matrix W.

The network m outputs unnormalized matrices $\widehat{W}$ and $\widehat{V}$. We again normalize the matrices using the standard softmax $W_i = \sigma(\widehat{W}_i)$, and an adjusted version in the case of V:

$$V_{i,j} = \frac{\exp\left(\widehat{V}_{i,j}\right)}{\sum_{k=1}^{K} \frac{\exp\left(\widehat{V}_{i,k}\right) + \exp\left(\widehat{V}_{i,k+1}\right)}{2} W_{i,k}}, \qquad (16)$$

where the denominator ensures that V_i represents a valid PDF.

The PDF in dimension i is defined as

$$q_i(x_i^B) = \text{lerp}(V_{ib}, V_{ib+1}, \alpha), \qquad (17)$$

where $\alpha = (x_i^B - \sum_{k=1}^{b-1} W_{ik})/W_{ib}$ represents the relative position of scalar x_i^B in bin b that contains it, i.e. $\sum_{k=1}^{b-1} W_{ik} \leq x_i^B < \sum_{k=1}^{b} W_{ik}$.

The invertible coupling transform is obtained by integration:

$$C_i(x_i^B; W, V) = \frac{\alpha^2}{2}(V_{ib+1} - V_{ib})W_{ib} + \alpha V_{ib}W_{ib} \qquad (18)$$

$$+ \sum_{k=1}^{b-1} \frac{V_{ik} + V_{ik+1}}{2} W_{ik}. \qquad (19)$$

Note that inverting $C_i(x_i^B; W, V)$ involves solving the root of the quadratic term, which can be done efficiently and robustly.

Computing the determinant of the Jacobian matrix follows the same logic as in the piecewise-linear case, with the only difference being that we must now interpolate the entries of V in order to obtain the PDF value at a specific location (cf. Equation (17)).

4.3 One-Blob Encoding

An important consideration is the encoding of the inputs to the network. We propose to use the *one-blob* encoding—a generalization of the *one-hot* encoding [Harris and Harris 2013]—where a kernel is used to activate multiple adjacent entries instead of a single one. Assume a scalar $s \in [0, 1]$ and a quantization of the unit interval into k bins (we use $k = 32$). The one-blob encoding amounts to placing a kernel (we use a Gaussian with $\sigma = 1/k$) at s and discretizing it into the bins. With the proposed architecture of the neural network (placement of ReLUs in particular, see Figure 3), the one-blob encoding effectively shuts down certain parts of the linear path of the network, allowing it to specialize the model on various sub-domains of the input.

In contrast to one-hot encoding, where the quantization causes a loss of information if applied to continuous variables, the one-blob encoding is lossless; it captures the exact position of s.

4.4 Analysis

We compare the proposed piecewise-polynomial coupling transforms to multiply-add affine transforms [Dinh et al. 2016] on a 2D regression problem in Figure 4. To produce columns 1–5, we sample the 2D domain using *uniform* i.i.d. samples (16 384 samples per
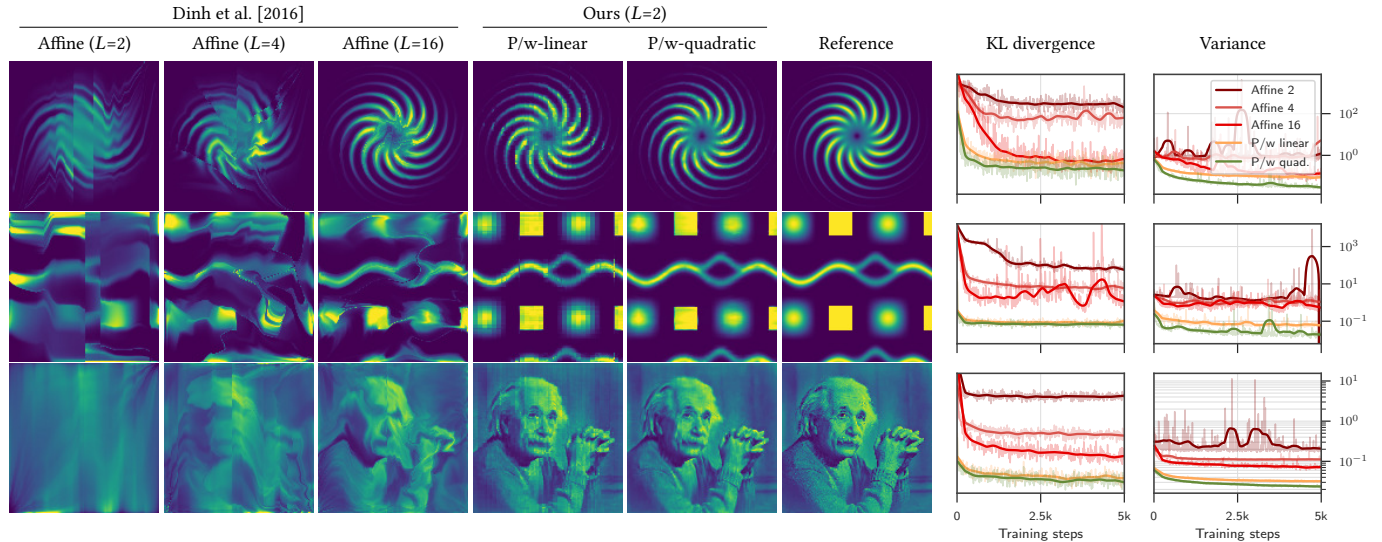
Fig. 4. Our 32-bin piecewise-linear (4-th column) and 32-bin piecewise-quadratic (5-th column) coupling layers achieve superior performance compared to affine (multiply-add) coupling layers [Dinh et al. 2016] on low-dimensional regression problems. The false-colored distributions were obtained by optimizing KL divergence with uniformly drawn i.i.d. samples (weighted by the reference value) over the 2D image domain. The plots on the right show logarithmically scaled training error (KL divergence) and the variance of estimating the average image intensity when drawing samples from one of the distributions.
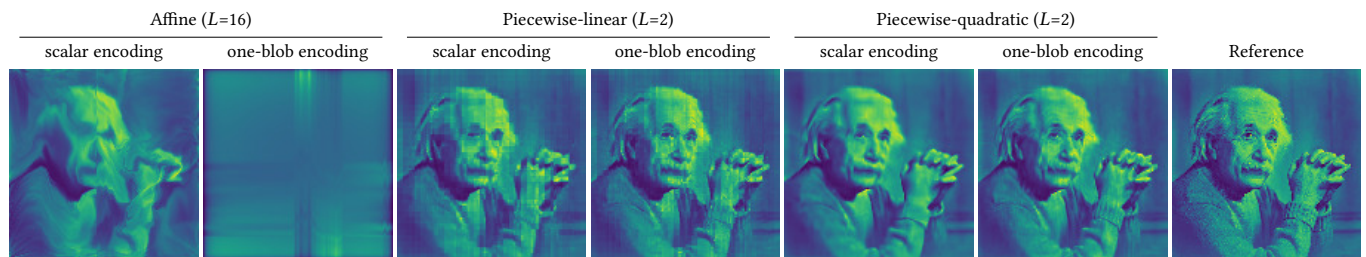

Fig. 5. Comparison of results *with* and *without* the one-blob encoding. The experimental setup is the same as in Figure 4. While the affine coupling transforms fail to converge with one-blob-encoded inputs, the distributions learned by the piecewise-polynomial coupling functions become sharper and more accurate.

training step), evaluate the reference function (column 6) at each sample, and optimize the neural networks that control the coupling transforms using KL divergence described in Section 5.1. We also ran the same experiment with equally weighted i.i.d. samples drawn *proportional* to the reference function—i.e. in a density-estimation setting—producing near-identical results (not shown). Every per-layer network has a U-net (see Figure 3) with 8 fully connected layers, where the outermost layers contain 256 neurons and the number of neurons is halved at every nesting level. We use 2 additional layers to adapt the input and output dimensionalities to and from 256, respectively. The networks only differ in their output layer to produce the desired parameters of their respective coupling transform (s and t, $\widehat{Q}$, or $\widehat{W}$ and $\widehat{V}$).

We use adaptive bin sizes only in the piecewise-quadratic coupling transforms because gradient descent fails to optimize them in the piecewise-linear case as demonstrated in Appendix B.

When using $L = 2$ coupling layers—i.e. 2×10 fully connected layers—the piecewise-polynomial coupling layers consistently perform better thanks to their significantly larger modeling power,

and outperform even large numbers (e.g. $L = 16$) of multiply-add coupling layers, amounting to 16×10 fully connected layers.

Figure 5 demonstrates the benefits of the one-blob encoding when combined with our piecewise coupling transforms. While the encoding helps our coupling transforms to learn sharp, non-linear functions more easily, it also causes the multiply-add transforms of Dinh et al. [2016] to produce excessive high frequencies that inhibit convergence. Therefore, in the rest of the paper we use the one-blob encoding only with our piecewise-polynomial transforms; results with affine transforms do not utilize one-blob encoded inputs.

We tested the piecewise-quadratic coupling layers also on a high-dimensional density-estimation problem: learning the manifold of a specific class of natural images. We used the CelebFaces Attributes dataset [Liu et al. 2015] and reproduced the experimental setting of Dinh et al. [2016]. Our architecture is based on the authors' publicly available implementation and differs only in the used coupling layer and the depth of the network—we use 4 recursive subdivisions while the authors use 5, resulting in 28 versus 35 coupling layers. We chose $K = 4$ bins and *did not* use our one-blob encoding due

Examples from the training set   Generated novel images   Manifold spanned by four images
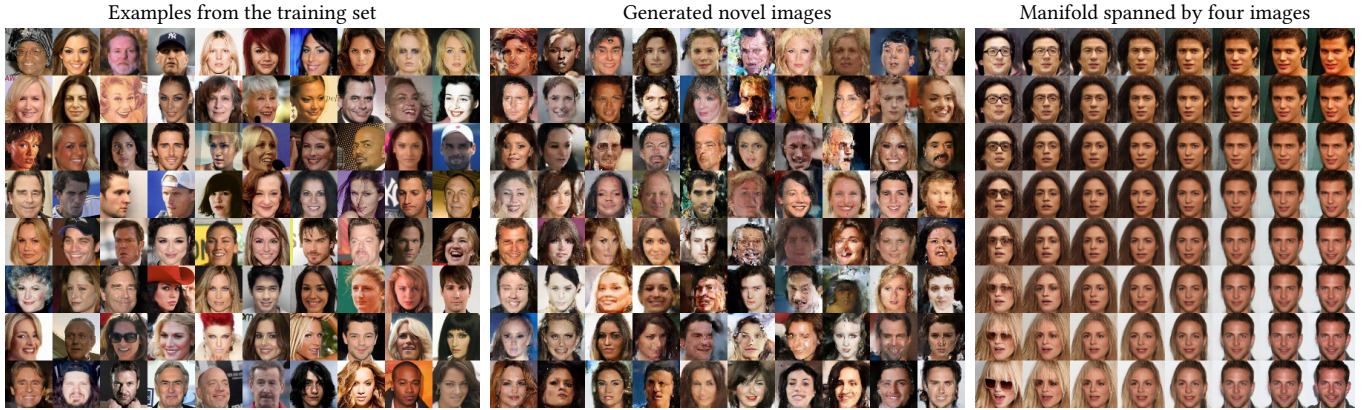

Fig. 6. Generative modeling of facial photographs using the architecture of Dinh et al. [2016] with our piecewise-quadratic coupling transform. We show training examples (left), faces generated by our trained model (middle), and a manifold of faces spanned by linear interpolation of 4 training examples in latent space (right; training examples are in the corners). We achieve validation results of slightly better quality than Dinh et al. [2016] in terms of negative log-likelihood (2.89 vs. 3.02 bits per dimension), suggesting that our approach could also benefit high-dimensional problems, though one may not achieve the same magnitude of improvements as in low-dimensional settings.

to GPU memory constraints. Since our coupling layers operate on $[0, 1]^D$, we do not use batch normalization on the transformed data.

Figure 6 shows a sample of the training set, a sample of generated images, and a visualization of the manifold given by four different faces. The visual quality of our results is comparable to those obtained by Dinh and colleagues. We perform marginally better in terms of negative log-likelihood (lower means better): we yield 2.85 and 2.89 bits per dimension on training and validation data, respectively, whereas Dinh et al. [2016] reported 2.97 and 3.02 bits per dimension. We tried decreasing the number of coupling layers while increasing the number of bins within each of them, but the results became overall worse. We hypothesize that the high-dimensional problem of learning distributions of natural images benefits more from having many coupling layers rather than having fewer but expressive ones.

5 MONTE CARLO INTEGRATION WITH NICE

In this section, we apply the NICE framework to Monte Carlo integration. Our goal is to reduce estimation variance by extracting sampling PDFs from observations of the integrand. Denoting $q(x; \theta)$ the to-be-learned PDF for drawing samples (θ represents the trainable parameters) and $p(x)$ the ground-truth distribution of the integrand, we can rewrite the MC estimator from Equation (3) as

$$\langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i; \theta)} = \frac{1}{N} \sum_{i=1}^{N} \frac{p(X_i) F}{q(X_i; \theta)} . \quad (20)$$

In the ideal case when $q(x; \theta) = p(x)$, the estimator returns the exact value of F. Our objective here is to leverage NICE to learn q *from data* while optimizing the neural networks in coupling layers so that the distance between p and q is minimized.

We follow the standard approach of quantifying the distance using one of the commonly used divergence metrics. While all divergence metrics reach their minimum if both distributions are equal, they differ in shape and therefore produce different q in practice.

In Section 5.1, we optimize using the popular Kullback-Leibler (KL) divergence. We further consider directly minimizing the variance of the resulting MC estimator in Section 5.2 and demonstrate that it is equivalent to minimizing the χ^2 divergence.

5.1 Minimizing Kullback-Leibler Divergence

Most generative models based on deep neural networks do not allow evaluating the likelihood $q(x; \theta)$ of data points x exactly and/or efficiently. In contrast, our work is based on bijective mappings with tractable Jacobian determinants that easily permit such evaluations. In the following, we show that minimizing the KL divergence with gradient descent amounts to maximizing a weighted log likelihood.

The KL divergence between $p(x)$ and the learned $q(x; \theta)$ reads

$$D_{KL}(p \parallel q; \theta) = \int_\Omega p(x) \log \frac{p(x)}{q(x; \theta)} \, dx$$

$$= \int_\Omega p(x) \log p(x) \, dx - \underbrace{\int_\Omega p(x) \log q(x; \theta) \, dx}_{\text{Cross entropy}} . \quad (21)$$

To minimize D_{KL} with gradient descent, we need its gradient with respect to the trainable parameters θ. These appear only in the cross-entropy term, hence

$$\nabla_\theta D_{KL}(p \parallel q; \theta) = -\nabla_\theta \int_\Omega p(x) \log q(x; \theta) \, dx \quad (22)$$

$$= \mathbb{E}\left[-\frac{p(X)}{q(X; \theta)} \nabla_\theta \log q(X; \theta) \right] , \quad (23)$$

where the expectation is over $X \sim q(x; \theta)$, i.e. the samples are drawn from the learned generative model[3]. In most integration problems, $p(x)$ is only accessible in an unnormalized form through $f(x)$: $p(x) = f(x)/F$. Since F is unknown—this is what we are trying to estimate

---

[3]If samples could be drawn directly from the ground-truth distribution—as is common in computer vision problems—the stochastic gradient would simplify to that of just the log likelihood. We discuss a generalization of log-likelihood maximization.

in the first place—the gradient can be estimated only up to the global scale factor F. This is not an issue since common gradient-descent-based optimization techniques such as Adam [Kingma and Ba 2014] scale the step size by the reciprocal square root of the gradient variance, cancelling F. Furthermore, if $f(x)$ can only be estimated via Monte Carlo, the gradient is still correct due to the linearity of expectations. Equation (23) therefore shows that minimizing the KL divergence via gradient descent is equivalent to minimizing the negative log likelihood weighted by MC estimates of F.

5.2 Minimizing Variance via χ^2 Divergence

The most attractive quantity to minimize in the context of (unbiased) Monte Carlo integration is the variance of the estimator. Inspired by previous works that strive to directly minimize variance [Herholz et al. 2018, 2016; Pantaleoni and Heitz 2017; Vévoda et al. 2018], we demonstrate how this can be achieved for the MC estimator $p(X)/q(X; \theta)$, with $X \sim q(x; \theta)$, via gradient descent. We begin with the variance definition and simplify:

$$
\begin{aligned}
\mathbb{V}\left[\frac{p(X)}{q(X; \theta)}\right] &= \mathbb{E}\left[\frac{p(X)^2}{q(X; \theta)^2}\right] - \mathbb{E}\left[\frac{p(X)}{q(X; \theta)}\right]^2 \\
&= \int_\Omega \frac{p(x)^2}{q(x; \theta)} \, dx - \underbrace{\left(\int_\Omega p(x) \, dx\right)^2}_{1}.
\end{aligned}
\tag{24}
$$

The stochastic gradient of the variance for gradient descent is then

$$
\begin{aligned}
\nabla_\theta \mathbb{V}\left[\frac{p(X)}{q(X; \theta)}\right] &= \nabla_\theta \int_\Omega \frac{p(x)^2}{q(x; \theta)} \, dx \\
&= \int_\Omega p(x)^2 \, \nabla_\theta \frac{1}{q(x; \theta)} \, dx \\
&= \int_\Omega -\frac{p(x)^2}{q(x; \theta)} \, \nabla_\theta \log q(x; \theta) \, dx \\
&= \mathbb{E}\left[-\left(\frac{p(X)}{q(X; \theta)}\right)^2 \nabla_\theta \log q(X; \theta)\right].
\end{aligned}
\tag{25}
$$

Relation to the Pearson χ^2 divergence. Upon close inspection it turns out the variance objective (Equation 24) is equivalent to the Pearson χ^2 divergence $D_{\chi^2}(p \parallel q; \theta)$:

$$
\begin{aligned}
D_{\chi^2}(p \parallel q; \theta) &= \int_\Omega \frac{(p(x) - q(x; \theta))^2}{q(x; \theta)} \, dx \\
&= \int_\Omega \frac{p(x)^2}{q(x; \theta)} \, dx - \underbrace{\left(2\int_\Omega p(x) \, dx - \int_\Omega q(x; \theta) \, dx\right)}_{1}.
\end{aligned}
\tag{26}
$$

As such, minimizing the variance of a Monte Carlo estimator amounts to minimizing the Pearson χ^2 divergence between the ground-truth and the learned distributions.

Connection between the χ^2 and KL divergences. Notably, the gradients of the KL divergence and the χ^2 divergence differ only in the weight applied to the log likelihood. In $\nabla_\theta D_{\mathrm{KL}}$ the log likelihood is weighted by the MC weight, whereas when optimizing $\nabla_\theta D_{\chi^2}$, the log likelihood is weighted by the *squared* MC weight. This illustrates

the difference between the two loss functions: the χ^2 divergence penalizes large discrepancies stronger, specifically, low values of q in regions of large density p. As such, it tends to produce more conservative q than D_{KL}, which we observe in Section 6 as fewer outliers at the cost of slightly worse average performance.

6 NEURAL PATH SAMPLING AND PATH GUIDING

In this section, we take NICE (Section 3) with piecewise-polynomial warps (Section 4) and apply it to sequential MC integration of light transport using the methodology described in Section 5. Our goal is to reduce estimation variance by "guiding" the construction of light paths using on-the-fly learned sampling densities. We explore two different settings: a global setting that leverages the path-integral formulation of light transport and employs high-dimensional sampling in the primary sample space (PSS) to build complete light-path samples (Section 6.1), and a local setting, natural to the rendering equation, where the integration spans a 2D (hemi-)spherical domain and the path is built incrementally (Section 6.2).

6.1 Primary-Sample-Space Path Sampling

In order to produce an image, a renderer must estimate the amount of light reaching the camera after taking any of the possible paths through the scene. The transport can be formalized using the path-integral formulation [Veach 1997], where a radiance measurement I to a sensor (e.g. a pixel) is given by an integral over path space $\mathcal{P}$:

$$
I = \int_\mathcal{P} L_e(\mathbf{x}_0, \mathbf{x}_1) \, T(\overline{x}) \, W(\mathbf{x}_{k-1}, \mathbf{x}_k) \, d\overline{x}.
\tag{27}
$$

The chain of positions $\overline{x} = \mathbf{x}_0 \cdots \mathbf{x}_k$ represents a single light path with k vertices. The path throughput $T(\overline{x})$ quantifies the ability of $\overline{x}$ to transport radiance. L_e represents emitted radiance and W is the sensor response to one unit of incident radiance.

The measurement can be estimated as

$$
\langle I \rangle = \frac{1}{N} \sum_{j=1}^N \frac{L_e(\mathbf{x}_{j0}, \mathbf{x}_{j1}) \, T(\overline{x}_j) \, W(\mathbf{x}_{jk-1}, \mathbf{x}_{jk})}{q(\overline{x}_j)},
\tag{28}
$$

where $q(\overline{x})$ is the joint probability density of generating all k vertices of path $\overline{x}$. Drawing samples from the joint distribution is challenging due to the constrained nature of vertices; e.g. they have to reside on surfaces. Several approaches thus propose to operate in the primary sample space (PSS) [Kelemen et al. 2002] represented by a unit hypercube $\mathcal{U}$. A path is then obtained by transforming a vector of random numbers $z \in \mathcal{U}$ using one of the standard path-construction techniques ρ (e.g. camera tracing): $\overline{x} = \rho(z)$.

Operating in PSS has a number of compelling advantages. The sampling routine has to be evaluated only once per path, instead of once per path vertex, and the generic nature of PSS coordinates enables treating the path construction as a black box. Importance sampling of paths can thus be applied to any single path-tracing technique, and, with some effort, also to multiple strategies [Guo et al. 2018; Hachisuka et al. 2014; Kelemen et al. 2002; Lafortune and Willems 1995; Veach and Guibas 1994; Zheng and Zwicker 2018]. Lastly, the sampling routine directly benefits from existing importance-sampling techniques in the underlying path-tracing algorithm since those make the path-contribution function smoother in PSS and thus easier to learn.

Methodology. Given that NICE scales well to high-dimensional problems, applying it in PSS is straightforward. We split the dimensions of $\mathcal{U}$ into two equally-sized groups A and B, where A contains the even dimensions and B contains the odd dimensions. One group serves as the input of the neural network (each dimension is processed using the one-blob encoding) while the other group is being warped; their roles are swapped in the next coupling layer. To infer the parameters θ of the networks, we minimize one of the losses from Section 5 against $p(\overline{x}) = L_e(\mathbf{x}_0, \mathbf{x}_1) T(\overline{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) F^{-1}$, ignoring the unknown normalization factor, i.e. assuming $F = 1$.

In order to obtain a path sample $\overline{x}$, we generate a random vector z, warp it using the reversed inverted coupling layers, and apply the path-construction technique: $\overline{x} = \rho\left(h_1^{-1}\left(\cdots h_L^{-1}(z)\right)\right)$; please refer back to Equation (9) and (10) for details on the inverses.

Before we analyze the performance of primary-sample-space path sampling in Section 6.4, we discuss a slightly different approach to data-driven construction of path samples—the so-called path guiding—which applies neural importance sampling at each vertex of the path and typically yields higher performance.

6.2 Path Guiding

A popular alternative to formalizing light transport using the path-integral formulation is to adopt a local view and focus on the radiative equilibrium of individual points in the scene. The equilibrium radiance at a surface point $\mathbf{x}$ in direction ω_o is given by the rendering equation [Kajiya 1986]:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega L(\mathbf{x}, \omega) f_s(\mathbf{x}, \omega_o, \omega) |\cos\gamma| \, d\omega, \quad (29)$$

where f_s is the bidirectional scattering distribution function, $L_o(\mathbf{x}, \omega_o)$, $L_e(\mathbf{x}, \omega_o)$, and $L(\mathbf{x}, \omega)$ are respectively the reflected, emitted, and incident radiance, Ω is the unit sphere, and γ is the angle between ω and the surface normal.

The rendering task is formulated as finding the outgoing radiance at points directly visible from the sensor. The overall efficiency of the renderer heavily depends on the variance of estimating the amount of *reflected* light:

$$\langle L_r(\mathbf{x}, \omega_o)\rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{L(\mathbf{x}, \omega_j) f_s(\mathbf{x}, \omega_o, \omega_j) |\cos\gamma_j|}{q(\omega_j | \mathbf{x}, \omega_o)}. \quad (30)$$

A large body of research has therefore focused on devising sampling densities $q(\omega | \mathbf{x}, \omega_o)$ that yield low variance. While the density is defined over a 2D space, it is conditioned on position $\mathbf{x}$ and direction ω_o. These extra five dimensions make the goal of $q(\omega | \mathbf{x}, \omega_o) \propto L(\mathbf{x}, \omega) f_s(\mathbf{x}, \omega_o, \omega) |\cos\gamma|$ substantially harder.

Since the 7D domain is fairly challenging to handle using hand-crafted, spatio-directional data structures in the general case, most research has focused on the simpler 5D setting where $q(\omega | \mathbf{x}, \omega_o) \propto L(\mathbf{x}, \omega)$ [Dahm and Keller 2018; Hey and Purgathofer 2002; Jensen 1995; Müller et al. 2017; Pegoraro et al. 2008a,b; Vorba et al. 2014] and only a few attempts have been made to consider the full product [Herholz et al. 2018, 2016; Lafortune and Willems 1995; Steinhurst and Lastra 2006]. These *path-guiding* approaches rely on carefully chosen data structures (e.g. BVHs, kD-trees) in combination with relatively simple PDF models (e.g. histograms, quad-trees,

Gaussian mixture models), which are populated in a data-driven manner either in a pre-pass or online during rendering. Like in previous works our goal is to learn local sampling densities, but we differ in that we utilize NICE to represent and sample from $q(\omega | \mathbf{x}, \omega_o)$.

Methodology. We use a single instance of NICE, which is trained and sampled from in an interleaved manner: drawn samples are immediately used for training, and training results are immediately used for further sampling. In the most general setting, we consider learning $q(\omega | \mathbf{x}, \omega_o)$ that is proportional to the product of *all* terms in the integrand. Since the integration domain is only 2D, partitions A and B in all coupling layers contain only one dimension each—one of the two cylindrical coordinates that we use to parameterize the sphere of direction.

To produce the parameters of the first piecewise-polynomial coupling function, the neural network m takes the cylindrical coordinate from A, the position $\mathbf{x}$ and direction ω_o that condition the density, and additional information that may improve inference; we also input the normal of the intersected shape at $\mathbf{x}$ to aid the network in learning distributions that correlate with the local shading frame.

We one-blob-encode all of the inputs as described in Section 4.3 with $k = 32$. In the case of $\mathbf{x}$, we normalize it by the scene bounding box, encode each coordinate independently, and concatenate the results into a single array of $3 \times k$ values. We proceed analogously with directions, which we parameterize using world-space cylindrical coordinates: we transform each coordinate to $[0, 1]$ interval, encode it, and append to the array. The improved performance enabled by our proposed one-blob encoding is reported in Table 1.

At any given point during rendering, a sample is generated by drawing a random pair $u \in [0, 1]^2$, passing it through the inverted coupling layers in reverse order, $h_1^{-1}(\cdots h_L^{-1}(u))$, and transforming to the range of cylindrical coordinates to obtain ω.

MIS-Aware Optimization. In order to optimize the networks, we use Adam with one of the loss functions from Section 5, but with an important, problem-specific alteration. To sample ω, most current renderers apply multiple importance sampling (MIS) [Veach and Guibas 1995] to combine multiple sampling densities, each tailored to a specific component of the integrand (direct illumination, BSDF, etc.). When learning the product, we take this into account by optimizing the networks with respect to the final *effective* PDF q' instead of the density learned using NICE. If certain parts of the product are already covered well by existing densities, the networks will be optimized to handle only the remaining problematic case.

We minimize $D(p \| q')$, where D is either D_{KL} or D_{χ^2} divergence, and we employ the balance heuristic [Veach and Guibas 1995] for combining the learned distribution q and the BSDF distribution p_{f_s}. This yields the following effective PDF $q' = cq + (1-c)p_{f_s}$, where c is the probability of drawing samples from q. The ideal PDF $p(\omega | \mathbf{x}, \omega_o) = L(\mathbf{x}, \omega) f_s(\mathbf{x}, \omega_o, \omega) |\cos\gamma| F^{-1}$ is evaluated ignoring the normalization constant F (as discussed in Section 5.1).

Learned Selection Probabilities. To further reduce variance we use an additional network $\hat{m}$ that learns approximately optimal selection probability $c = l(\hat{m}(\mathbf{x}, \omega_o))$, where l is the logistic function. We optimize $\hat{m}$ jointly with the other networks; all use the same architecture except for the last layer. To prevent overfitting to local

optima with degenerate selection probability, we use loss funtion $\beta(\tau) D(p \parallel q) + (1 - \beta(\tau)) D(p \parallel q')$ where $\tau \in [0, 1]$ is the fraction of exhausted render budget (either time or sample count) and $\beta(\tau) = 1/2 \cdot (1/3)^{5\tau}$.

Since we employ the balance heuristic, which is provably optimal in the context of the MIS one-sample model [Veach 1997], learning the selection probabilities is the only means to further improve the performance, for instance by shutting down MIS completely in situations when it may hurt (e.g. on near-specular surfaces).

It is worth noting, however, that learning also the MIS weights (instead of relying on the balance heuristic) would remove the need to evaluate all relevant distributions for each sample. This would remove one of the requirements that we stated in Section 2, namely the need for tractable inverses, thereby extending the set of admissible generative models; we leave this branch of investigations as future work.

BSDFs with Delta Components. BSDFs that are a mixture of Dirac-delta and smooth functions—such as smooth plastic—require special handling. While our stochastic gradient in Section 5 is, in theory, well behaved with delta functions, they need to be treated as finite quantities in practice due to the limitations of floating-point numbers. When the path tracer samples delta components, continuous densities need to be set to 0 and optimization of our coupling functions disabled (by setting their loss to 0), effectively only optimizing for selection probabilities.

Discussion. Our approach to sampling the full product of the rendering equation $L(\mathbf{x}, \omega_j) f_s(\mathbf{x}, \omega_o, \omega_j) |\cos \gamma_j|$ has three distinct advantages. First, it is agnostic to the number of dimensions that the 2D domain is conditioned on. This allows for high-dimensional conditionals without sophisticated data structures. One can simply input extra information into the neural networks and let them learn which dimensions are useful in which situations. While we only pass in the surface normal, the networks could be supplied with additional information—e.g. textured BSDF parameters—to further improve the performance in cases where the product correlates with such information. In that sense, our approach is more automatic than previous works.

The second advantage is that our method does not require any precomputation, such as fitting of (scene-dependent) materials into a mixture of Gaussians [Herholz et al. 2018, 2016]. While a user still needs to specify the hyperparameters as is also required by most other approaches, we found our configuration of hyperparameters to work well across all tested scenes. Note, however, that the lack of explicit factorization in our approach can be detrimental in situations where the individual factors are simpler to learn than their product, and the product can be easily importance sampled.

Lastly, our approach offers trivial persistence across renders. A set of networks trained on one camera view can be reused from a different view or within a slightly modified scene; see Section 6.4. Unlike previous approaches, where the learned data structure requires explicit support of adaptation to new scenes, neural networks can be adapted by the same optimization procedure that was used in initial training. Applying our approach to animations could thus yield sub-linear training cost by amortizing it over multiple frames.

6.3 Experimental Setup

We implemented our technique in Tensorflow [Abadi et al. 2015] and integrated it with the Mitsuba renderer [Jakob 2010]. Before we start rendering, we initialize the trainable parameters of our networks using Xavier initialization [Glorot and Bengio 2010]. While rendering the image, we optimize them using Adam [Kingma and Ba 2014]. Our rendering procedure is implemented as a hybrid CPU/GPU algorithm, tracing rays in large batches on the CPU while two GPUs perform all neural-network-related tasks. One GPU is responsible for optimizing the MIS selection probabilities and evaluating and sampling from q, while the other trains the networks using Monte Carlo estimates from completed paths. Both GPUs use minibatch sizes of 100 000 samples. Communication between the CPU and GPUs happens via asynchronous buffers to aid parallelization. Computation of selection probabilities and q-evaluation and q-sampling communicate via asynchronous *queues* that are processed as fast as possible. Our training buffer is configured to always contain the latest 2 000 000 samples of which minibatches are *randomly* selected for optimization. This procedure decorrelates samples that are nearby in the image plane.

In order to obtain the final image with N samples, we perform $M = \lfloor \log_2(N + 1) \rfloor$ iterations with power-of-two sample counts $2^i; i \in \{0, \ldots, M\}$. This approach was initially proposed by Müller et al. [2017] to limit the impact of initial high-variance estimates on the final image. In contrast to their work, we do not reset the learned distributions at every power-of-two iteration and keep training the same set of networks from start to finish. Furthermore, instead of discarding the pixel estimates of earlier iterations, we weight the images produced within each iteration by their reciprocal mean pixel variance, which we estimate on-the-fly. While this approach introduces bias, it is imperceptibly small in practice due to averaging across all pixels. Furthermore, the bias vanishes as the quality of the variance estimate increases, making this approach consistent. We apply the same weighting scheme to our implementation of the method by Müller et al. [2017] to ensure a fair comparison.

All results were produced on a workstation with two Intel Xeon E5-2680v3 CPUs (24 cores; 48 threads) and two NVIDIA Titan Xp GPUs. Due to the combined usage of both the CPU and the GPU, runtimes of different techniques depend strongly on the particular setup. We therefore focus on comparing the performance using *equal-sample-count* metrics that are independent of hardware. Absolute timings and an equal-time comparison of a subset of the scenes and methods are provided for completeness.

We quantify the error using the *mean absolute percentage error* (MAPE), which is defined as $\frac{1}{N} \sum_{i=1}^{N} |v_i - \hat{v}_i|/(\hat{v}_i + \epsilon)$, where $\hat{v}_i$ is the value of the i-th pixel in the ground-truth image, v_i is the value of the i-th rendered pixel, and $\epsilon = 0.01$ serves the dual objective of avoiding the singularity at $\hat{v}_i = 0$ and down-weighting close-to-black pixels. We use a relative metric to avoid putting overly much emphasis on bright image regions. We also evaluated L1, L2, *mean relative squared error* (MRSE) [Rousselle et al. 2011], *symmetric MAPE* (SMAPE), and SSIM, which all can be inspected as false-color maps and aggregates in the supplemented image viewer.
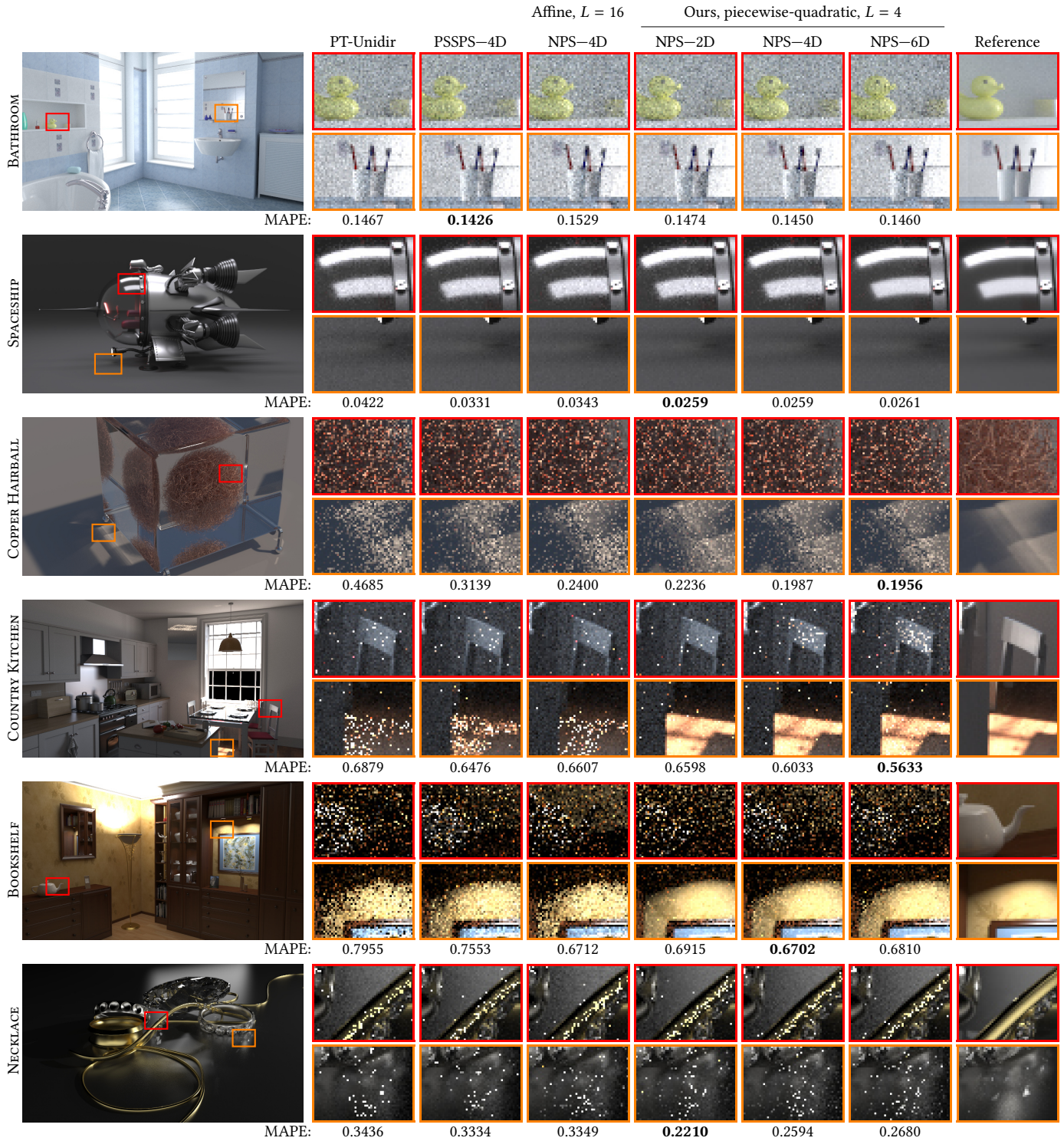
Fig. 7. Neural path sampling in primary sample space. We compare a standard uni-directional path tracer (PT-Unidir), the method by Guo et al. [2018] (PSSPS), neural path sampling using $L = 16$ multiply-add coupling layers [Dinh et al. 2016], and $L = 4$ of our proposed piecewise-quadratic coupling layers, both optimized using the KL divergence. We experimented with sampling the 1, 2, or 3 first non-specular bounces (NPS–2D, NPS–4D and NPS–6D). Overall, our technique performs best in terms of *mean absolute percentage error* (MAPE) in this experiment, but only offers improvement beyond the 4D case if paths stay coherent, e.g. in the top crop of the SPACESHIP scene. More results and error visualizations can be found in the supplemented image viewer.

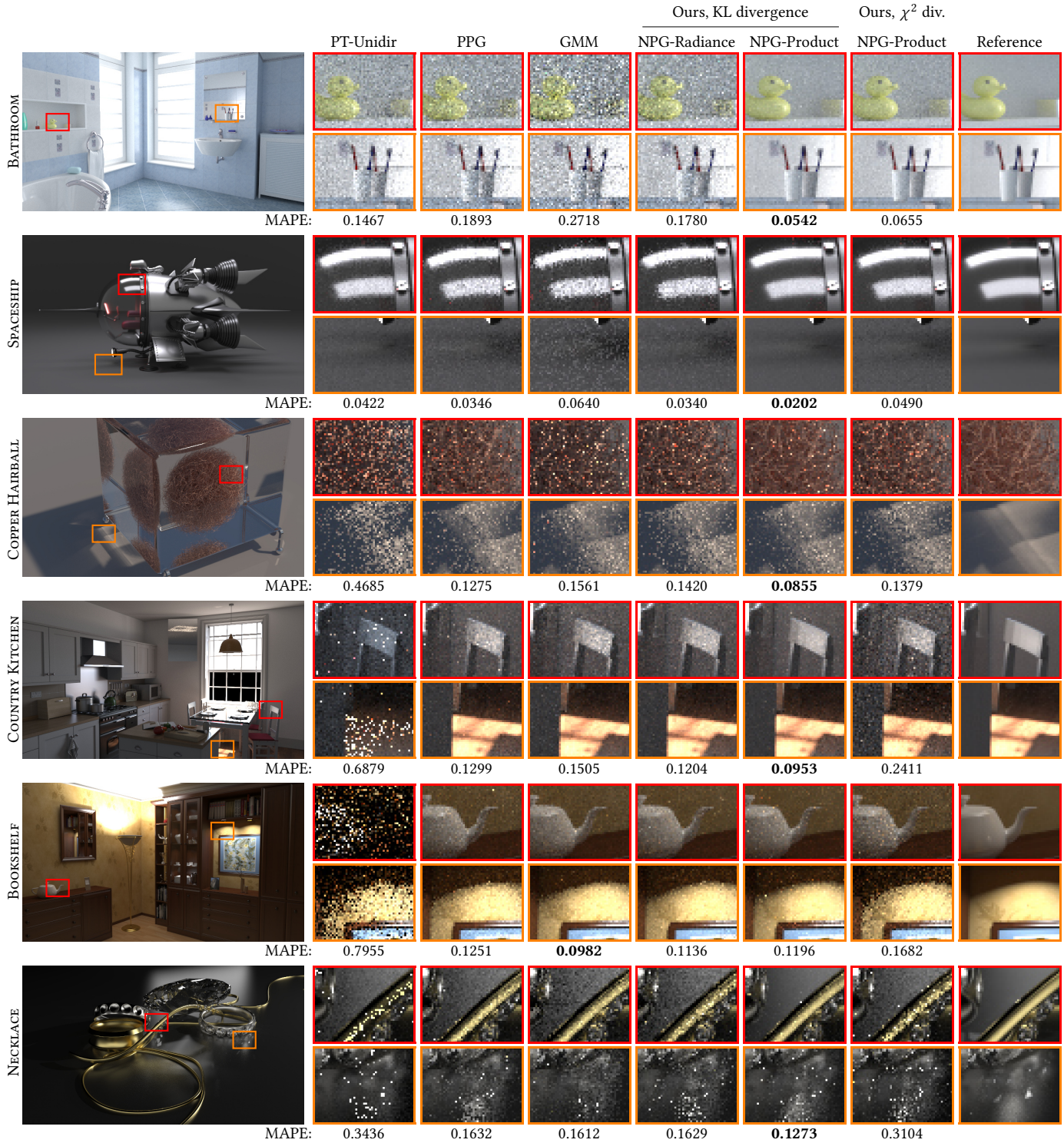|	PT-Unidir	PPG	GMM	Ours, KL divergence NPG-Radiance	NPG-Product	Ours, χ^2 div. NPG-Product	Reference
BATHROOM MAPE:	0.1467	0.1893	0.2718	0.1780	**0.0542**	0.0655	
SPACESHIP MAPE:	0.0422	0.0346	0.0640	0.0340	**0.0202**	0.0490	
COPPER HAIRBALL MAPE:	0.4685	0.1275	0.1561	0.1420	**0.0855**	0.1379	
COUNTRY KITCHEN MAPE:	0.6879	0.1299	0.1505	0.1204	**0.0953**	0.2411	
BOOKSHELF MAPE:	0.7955	0.1251	**0.0982**	0.1136	0.1196	0.1682	
NECKLACE MAPE:	0.3436	0.1632	0.1612	0.1629	**0.1273**	0.3104	

Fig. 8. Neural path guiding. We compare a uni-directional path tracer (PT-Unidir), the practical path-guiding (PPG) algorithm of Müller et al. [2017], the Gaussian mixture model (GMM) of Vorba et al. [2014], and variants of our framework with $L = 4$ coupling layers sampling the incident radiance alone (NPG-Radiance) or the whole integrand (NPG-Product), when optimizing either the KL and χ^2 divergences. Overall, sampling the whole integrand with the KL divergence yields the most robust results. More results and error visualizations can be found in the supplemented image viewer.

Table 1. Mean average percentage error (MAPE) and render times of various importance-sampling approaches. At equal sampling rates—we report the number of samples in each scene as mega samples (MS)—our technique performs on par or better than the practical path guiding (PPG) algorithm of Müller et al. [2017] and the bidirectionally trained Gaussian mixture model (GMM) of Vorba et al. [2014] in all scenes but the Bookshelf, but incurs a large computational overhead. Since the GMMs are trained in a pre-pass, we report both their training and rendering times. Please note, that the provided implementation of the GMM training does not scale well beyond 8 threads. Furthermore, we do not report GMM results for the Glossy Kitchen and Veach Door scenes due to crashes and bias, respectively. Our neural path sampling (NPS) likewise compares favorably against the method by Guo et al. [2018] (PSSPS). Using one-blob encoding significantly improves the quality of our results; see Figure 9 for a histogram visualization of these metrics. We also evaluated SMAPE, L1, MRSE, L2, and SSIM, which all can be inspected as false-color maps and aggregates in the supplemented image viewer.

| | | PT-Unidir | | PPG | | GMM | | PSSPS | | Ours, KL divergence | | | | | | | | Ours, χ^2 div. | |
										NPS one-blob		NPG-Rad. one-blob		NPG-Product scalar		NPG-Product one-blob		NPG-Product one-blob	
Bathroom	236 MS	0.147	88s	0.189	2.3m	0.272	9.1m + 48s	0.143	89s	0.146	3.5m	0.178	9.3m	0.071	11m	**0.054**	12m	0.066	15m
Bedroom	236 MS	0.078	75s	0.053	1.8m	0.063	34m + 60s	0.068	77s	0.068	3.5m	0.045	6.4m	0.037	7.2m	**0.032**	7.7m	0.042	10m
Bookshelf	236 MS	0.796	74s	0.125	2.5m	**0.098**	16m + 66s	0.755	78s	0.681	3.5m	0.114	8.1m	0.250	10m	0.120	10m	0.168	12m
Copper Hairball	472 MS	0.468	2.0m	0.128	1.8m	0.156	11m + 2.0m	0.314	2.0m	0.196	6.9m	0.142	4.9m	0.092	15m	**0.086**	16m	0.138	17m
Cornell Box	268 MS	0.185	23s	0.044	82s	0.049	6.2m + 24s	0.130	26s	0.109	4.0m	0.035	6.6m	0.027	8.5m	**0.021**	10m	0.027	9.1m
Country Kitchen	236 MS	0.688	48s	0.130	81s	0.151	13m + 33s	0.648	49s	0.563	3.5m	0.120	5.4m	0.123	6.9m	**0.095**	7.8m	0.241	8.1m
Glossy Kitchen	236 MS	1.476	78s	0.308	87s		—	1.452	77s	1.491	3.5m	0.243	5.6m	0.810	11m	**0.136**	11m	0.251	13m
Necklace	236 MS	0.344	29s	0.163	42s	0.161	3.2m + 15s	0.333	31s	0.268	3.5m	0.163	2.6m	0.249	10m	**0.127**	11m	0.310	10m
Salle de Bain	236 MS	0.185	51s	0.071	89s	0.080	11m + 37s	0.161	54s	0.158	3.5m	0.057	5.4m	0.052	5.7m	**0.042**	6.1m	0.062	7.9m
Spaceship	236 MS	0.042	27s	0.035	56s	0.064	4.4m + 4.6m	0.033	28s	0.026	3.5m	0.034	2.8m	0.027	3.5m	**0.020**	3.9m	0.049	4.1m
Sponza Atrium	236 MS	1.709	84s	0.353	91s	0.115	11m + 53s	1.692	81s	1.616	3.5m	**0.109**	7.4m	0.213	9.0m	0.110	11m	0.247	11m
Staircase	236 MS	0.163	57s	0.057	81s	0.065	14m + 41s	0.138	58s	0.130	3.5m	0.044	5.5m	0.038	5.7m	**0.033**	6.3m	0.043	7.7m
Swimming Pool	236 MS	0.688	40s	0.082	63s	0.077	29m + 22s	0.494	41s	0.198	3.5m	0.077	3.0m	0.073	4.5m	**0.066**	5.1m	0.087	5.1m
Veach Door	236 MS	0.910	38s	0.227	70s		—	0.903	41s	0.716	3.5m	0.135	9.0m	0.135	12m	**0.099**	13m	0.137	12m
White Room	236 MS	0.102	79s	0.066	1.9m	0.076	24m + 55s	0.090	79s	0.093	3.5m	0.058	7.0m	0.046	7.8m	**0.040**	8.3m	0.045	11m
Yet Another Box	1073 MS	1.228	1.9m	0.141	4.7m	0.079	6.0m + 1.7m	1.200	2.1m	1.108	15m	**0.059**	23m	0.222	30m	0.065	36m	0.110	33m
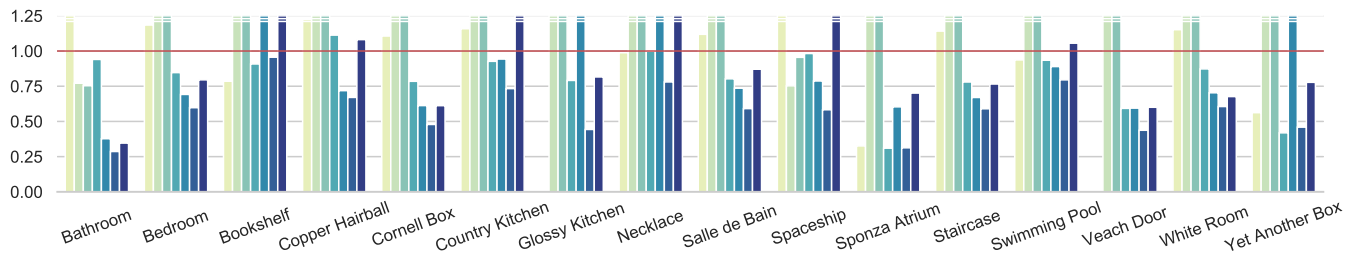
Fig. 9. MAPE achieved by the bidirectionally trained Gaussian mixture model by Vorba et al. [2014] (the Glossy Kitchen and Veach Door are omitted because of limitations of their implementation), the primary-sample-space method by Guo et al. [2018], and our neural importance-sampling approaches on different scenes (the order and colors of bars follows Table 1). The bars are normalized with respect to practical path guiding [Müller et al. 2017]; a height below 1 signifies better performance. Some bars exceed outside of the displayed range; Table 1 provides the actual numbers. Primary-sample-space techniques generally perform worse than path-guiding approaches. The product-driven neural path guiding usually performs the best.

6.4 Results

In order to best illustrate the benefits of different neural-importance-sampling approaches, we compare their performance when used on top of a unidirectional path tracer that uses BSDF sampling only. While none of the algorithms utilized next-event estimation (including prior works) to emphasize the performance of individual path sampling/guiding approaches, we recommend using it in practice for best performance. In the following, all results with our piecewise-polynomial coupling functions utilize $L = 4$ coupling layers. We use 1023 samples per pixel (spp) on all scenes except for the Copper Hairball (2047 spp) and Yet Another Box (4095 spp). The images were rendered at resolutions 640×360 and 512×512. It is

worth noting that since the quality of learned distribution depends primarily on the *total* number of drawn samples (reported as "mega samples" (MS)) rendering at higher resolutions yields high-quality distributions "faster", i.e. with fewer spp.

Path Sampling. In Figure 7, we study *primary-sample-space path sampling* (PSSPS) using our implementation of the technique by Guo et al. [2018] and our *neural path sampling* (NPS) with piecewise-polynomial and affine coupling transforms. We apply the sampling to only a limited number of non-specular interactions in the beginning of each path and sample all other interactions using uniform random numbers. We experimented with three different prefix dimensionalities: 2D, 4D, and 6D, which correspond to importance
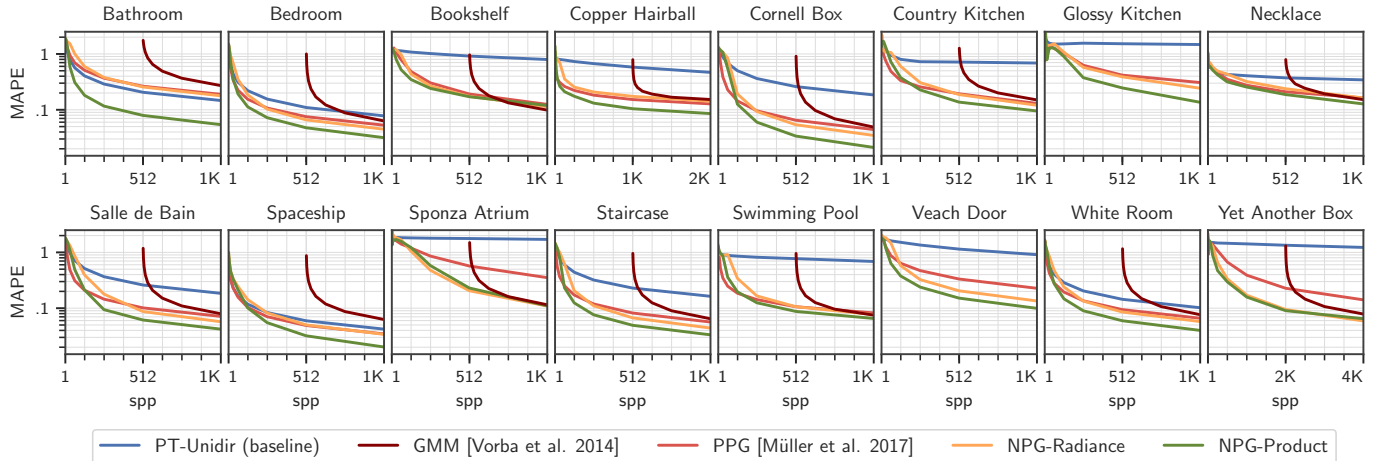
Fig. 10. Convergence plots of unidirectional path tracing (PT-Unidir), practical path guiding (PPG) [Müller et al. 2017], the algorithm of Vorba et al. [2014] (GMM), and our radiance- and product-based neural path guiding (NPG-Radiance and NPG-Product, respectively). We plot MAPE as a function of samples per pixel (spp) on a logarithmic scale. All guiding methods perform slightly worse than naïve path tracing initially, but overtake it rapidly on most scenes as they learn to importance sample. PPG tends to learn slightly faster than our NPG, but falls behind due to learning a worse distribution. The algorithm of Vorba et al. [2014] starts producing images only half-way through the sample budget as the first half of samples is used for offline pretraining.

sampling path prefixes of 1, 2, and 3 non-specular vertices, respectively. As shown in the figure, going beyond 4D brings typically little improvement in tested scenes, except for the highlights in the SPACESHIP, where even longer paths are correlated thanks to highly-glossy interactions with the glass of the cockpit[4]. This confirms the observation of Guo et al. [2018] that cases where more than two bounces are needed to connect to the light source offer minor to no improvement. We speculate that the poor performance in higher dimensions is due to the relatively weak correlation between path geometries and PSS coordinates, i.e. paths with similar PSS coordinates may have drastically different vertex positions. The correlation tends to weaken at each additional bounce (e.g. in the diffuse CORNELL BOX) unless the underlying path importance-sampling technique preserves path coherence.

Path Guiding. In Figure 8, we analyze the performance of different path-guiding approaches, referring to ours as *neural path guiding* (NPG). We compare our work to the respective authors' implementations of *practical path guiding* (PPG) by Müller et al. [2017] and the bidirectionally trained *Gaussian mixture model* (GMM) by Vorba et al. [2014], which are both learning sampling distributions that are, in contrast to ours, proportional to incident radiance only. We extended the GMM implementation to (oriented) spherical domains.

To isolate the benefits of using NICE with piecewise-quadratic coupling layers, we created a variant of our approach, NPG-Radiance, that learns PDFs proportional to incident radiance only and without the MIS-aware optimization. The *radiance-driven* neural path guiding outperforms PPG and GMM in 13 out of 16 scenes and follows closely in the others (BOOKSHELF, COPPER HAIRBALL, NECKLACE), making it the most robust method out of the three radiance-only approaches.

The performance of our neural approach is further increased by learning and sampling proportional to the full product and incorporating MIS into the optimization—this technique yields the best results in nearly all scenes. As seen on the COPPER HAIRBALL, our technique can learn the product even under high-frequency spatial variation by passing the surface normal as an additional input to the networks. We trained all techniques with the same number of samples as used for rendering. The SD-tree of PPG and our neural networks used between 5 MB and 10 MB, the Gaussian mixture model used between 5 MB and 118 MB.

Table 1 reports the MAPE metric and absolute timings of 9 methods on a set of 16 tested scenes. We also visualize the results of all methods using bar charts in Figure 9; the height is normalized with respect to PPG. We exclude GMM results for the GLOSSY KITCHEN and VEACH DOOR as we could not obtain representative results on these scenes. Path sampling in PSS typically yields significantly worse results than all path-guiding approaches. Neural path guiding always benefits (sometimes significantly) from encoding the inputs with one-blob encoding as opposed to inputing raw (scalar) values. This version performs the best except for two scenes where radiance-only NPG scored better.

Empirical Convergence Analysis. Convergence plots in Figure 10 provide further insight into the differences between unidirectional path tracing, the GMMs by Vorba et al. [2014], PPG by Müller et al. [2017], and our radiance- and product-based neural-path-guiding algorithms. In most cases, the online path-guiding algorithms quickly learn a superior sampling density compared to the baseline path tracer. The GMM algorithm—being trained offline—is inferior in the beginning of rendering, but produces competitive results after the total sample budget is exhausted. Our neural approaches produce the best results most of the time, with our product-based approach usually being superior to our radiance-based approach.

---

[4]Due to faster training of lower-dimensional distributions, the 2D case still has the least overall noise in the SPACESHIP scene.
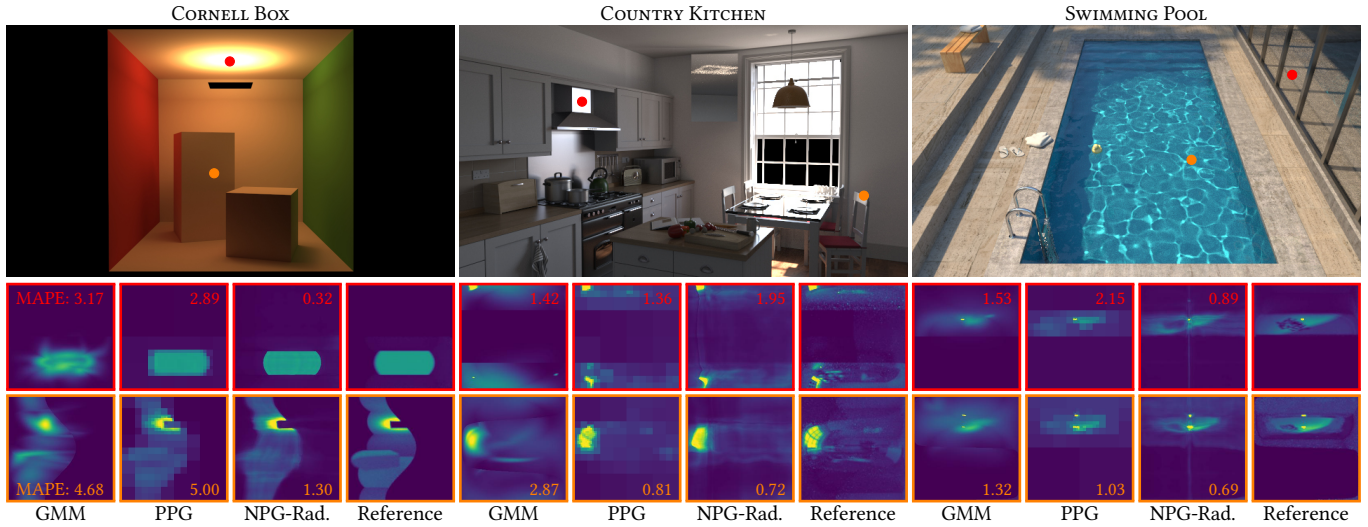
Fig. 11. Directional radiance distributions. From left to right: we visualize the distributions learned by a Gaussian mixture model (GMM) [Vorba et al. 2014], an SD-tree (PPG) [Müller et al. 2017], our neural path-guiding approach trained on radiance (NPG-Rad.), and a spatial binary tree with a directional regular 128×128 grid (Reference). The first three approaches were trained with an equal sample count and require roughly equal amounts of memory in the above scenes (around 10 MB). We used 2^{16} samples per pixel to generate the reference distributions, which require roughly 5 GB per scene. Despite its large computational cost, the reference solution is still slightly blurred (see e.g. CORNELL BOX, red inset). Our approach produces the most accurate distributions in the majority of cases, measured here using the mean average percentage error (MAPE). Unlike the competing techniques, however, we learn a continuous function in both the spatial and the directional domain, allowing for a smaller amount of blur in some cases, e.g. in the CORNELL BOX.

Optimizing KL vs. χ^2 Divergence. We compare variants of product-driven neural path guiding optimized using the Kullback-Leibler (KL) and χ^2 divergences during training. The squared Monte Carlo weight in the χ^2 gradient causes a large variance, making it difficult to optimize with. We remedy this problem by clipping the minibatch gradient norm to a maximum of 50. While the χ^2 divergence in theory minimizes the estimator variance most directly (see Section 5.2), it performs worse in practice according to all tested metrics on all test scenes (see Table 1 and the supplemented image viewer). A notable aspect of optimizing the χ^2 divergence is that it tends to produce results with higher variance overall, but fewer and less-extreme outliers.

Accuracy of Learned Distributions. We visualize learned radiance distributions in Figure 11, comparing our path-guiding neural distributions against the SD-tree, the GMM, and a reference solution. In most cases, NPG learns more accurate directional distributions than the competing methods in terms of the MAPE metric. Additionally, NPG produces a spatially *and* directionally continuous function; we illustrate the spatial continuity in the supplementary video.

Learned Selection Probabilities and MIS-Aware Optimization. In Figure 12, we demonstrate the increased robustness of neural path guiding offered by learning optimal selection probabilities. The impact is particularly noticeable on the cockpit of the spaceship seen through specular interactions, which are handled nearly optimally by sampling the material BSDF. In this region, a standard path tracer outperforms the learned sampling PDFs. With MIS-aware optimization—including the learning of selection probabilities—the
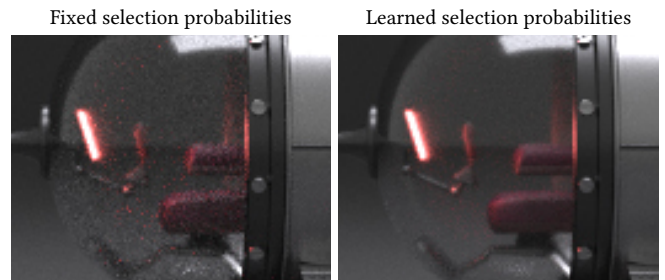

Fig. 12. Learning MIS selection probabilities—even with product-driven path guiding—leads to significantly better results on the SPACESHIP cockpit, where BSDF sampling is near optimal.

system downweighs the contribution of the learned PDF on the cockpit, but increases it in regions where it is more accurate, resulting in significantly improved results overall.

Weight Reuse Across Camera Views. Figure 13 demonstrates the benefits of reusing network weights, optimized for a particular camera view, in a novel view of the scene. We took network weights that resulted from generating images for Figure 8 as the *initial* weights for rendering images in the right column of Figure 13. Similarly to training from scratch, we keep optimizing the networks. If the initial distributions are already a good fit, our weighting by the reciprocal mean pixel variance automatically keeps initial pixel estimates rather than discarding them.

High-Frequency Material Parameters. In Figure 14, we show the benefits of passing spatially high-frequency material parameters
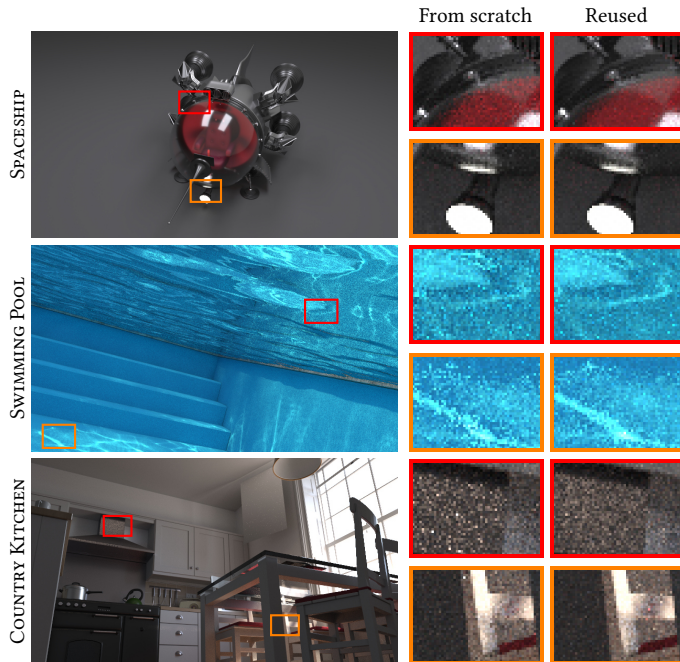
From scratch   Reused

Fig. 13. Learned distributions can be reused for novel camera views. The right column shows results where the network weights were initialized with weights learned for camera views in Figure 8.

w/o features   w/ features
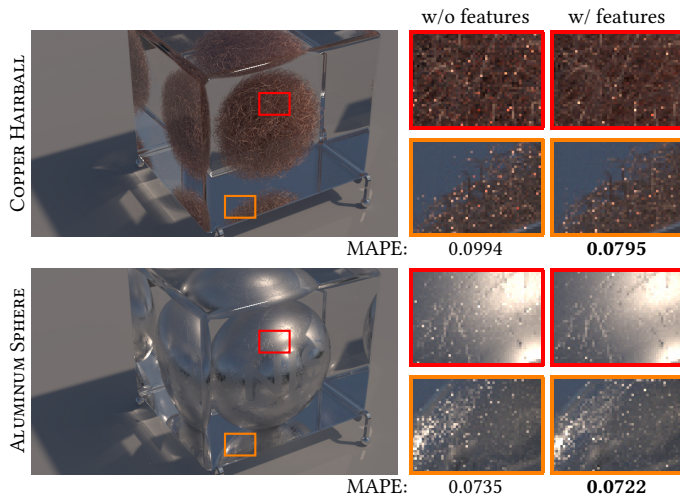
MAPE: 0.0994   **0.0795**

MAPE: 0.0735   **0.0722**

Fig. 14. Product-driven neural path guiding by itself has difficulties capturing high-frequency material properties (left). Passing material properties as additional input features enables the neural networks to learn parts of the appearance in the potentially lower-frequency material-parameter space (right), leading to a slightly better fit and thereby slightly reduced noise.

(e.g. the surface normal and roughness) as additional network inputs. When material parameters are *not* passed, the network must learn the product distribution purely as a function of spatio-directional coordinates, which is difficult. However, when the network receives material parameters as input, it can learn a potentially lower-frequency appearance model that exists partially in the material parameters' space, similar to the explicit factorization present in

other product-guiding methods [Herholz et al. 2018, 2016]. In contrast to our method, Herholz et al.'s techniques explicitly factorize incident radiance and the BSDF which avoids this problem entirely, potentially achieving better fits than shown in the figure.

Equal-Time Comparison. Lastly, we analyze the computational overhead of neural importance sampling in an equal-time comparison of unidirectional path tracing, PPG, and our product-driven NPG; see Figure 15. All techniques utilize a CPU for tracing paths. In addition, PPG uses the CPU for building and sampling the SD-tree, while our NPG also leverages two GPUs.

The radiance-driven PPG often performs best due to its small computational overhead, except when light transport is simple and/or the radiance-driven distribution is a poor fit to the product (e.g. the scenes in the top row). Despite utilizing two extra GPUs, the product-driven NPG is comparatively slow, on average constructing only about a quarter of the number of samples that PPG constructs. However, since these samples are of "higher quality", the technique manages to close most of the performance gap to PPG and unidirectional path tracing, in some cases producing the best results.

7 DISCUSSION AND FUTURE WORK

Runtime Cost. An important property of practical sampling strategies is a low computational cost of generating samples and evaluating their PDF, relative to the cost of evaluating the integrand. In our path-guiding applications, the cost is dominated by the evaluation of coupling layers: roughly 10% of the time is spent on one-blob encoding, 60% on fully connected layers, and 30% on the piecewise-polynomial warp. This makes the overhead of our implementation prohibitive in simple scenes. While we focused on the theoretical challenge of applying neural networks to the problem of importance sampling in this work, accelerating the computation to make our approach more practical is an important and interesting future work. We believe specialized hardware (e.g. NVIDIA's TensorCores) and additional computation graph optimization (e.g. NVIDIA's TensorRT) are promising next steps, which alone might be enough to bring our approach into the realm of practicality.

Optimizing for Multiple Integrals. In Section 5.1, we briefly discussed that the ground-truth density may be available only in unnormalized form. We argued that this is not a problem since the ignored factor F scales all gradients uniformly; it thus does not impact the optimization. These arguments pertain to handling a *single* integration problem. In Section 6, we demonstrated applications to path sampling and path guiding, where the learned density is conditioned on additional dimensions and we are thus solving many different integrals at once. Since the normalizing F varies between them, our arguments do not extend to this particular problem. Because neglecting the normalization factors is potentially negatively influencing the optimization, we experimented with tabulating F, but we did not experience noticeable improvements. This currently stands as a limitation of applying our work to path guiding/sampling and it would be worth addressing in future work.

Convergence of Optimization. Although our optimization based on stochastic gradients has many advantages, it also brings certain disadvantages. Techniques based on stochastic gradient descent do
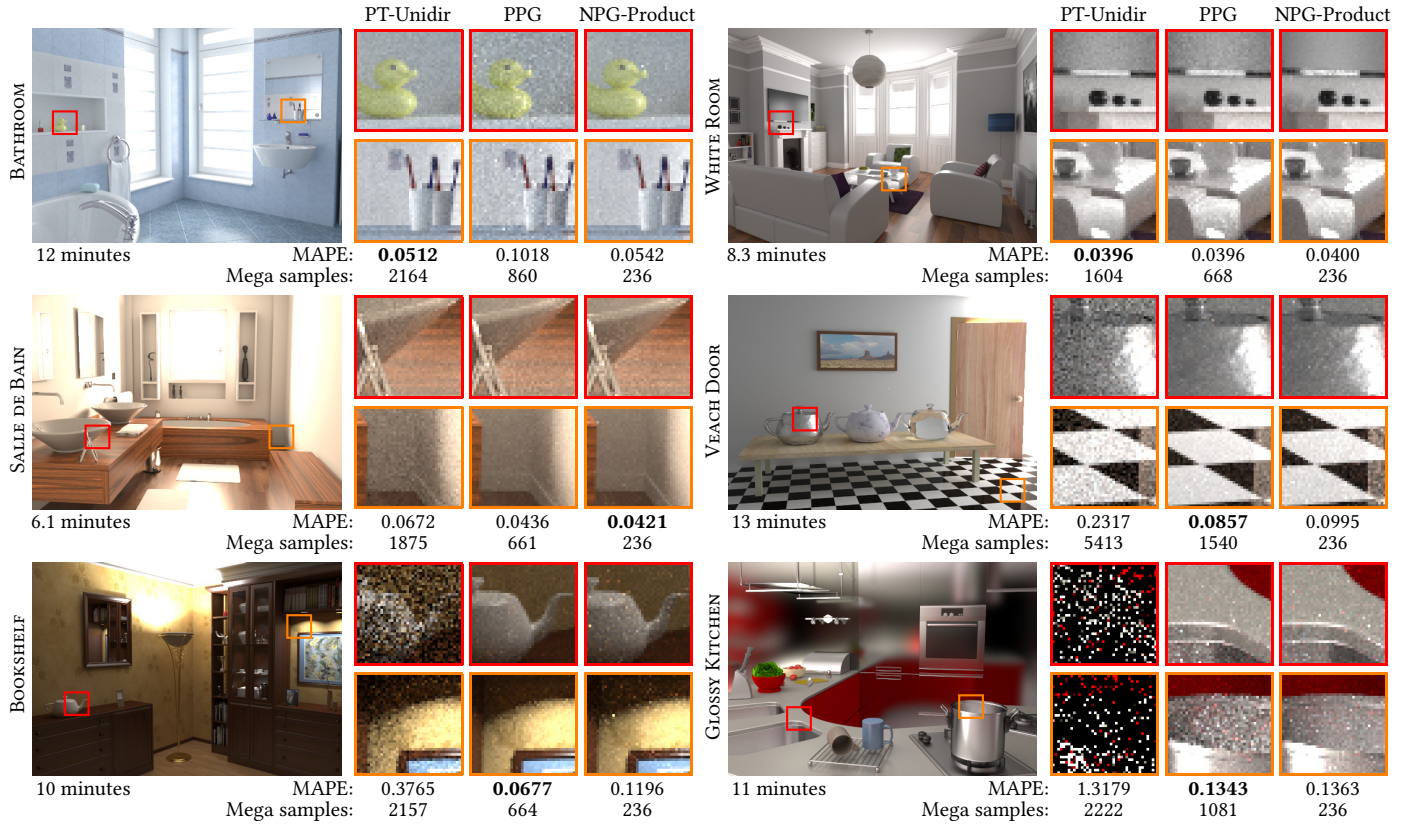
Fig. 15. Equal-time comparison of unidirectional path tracing (PT-Unidir), practical path guiding (PPG) [Müller et al. 2017], and our product-driven neural path guiding (NPG-Product). Despite the large computation cost of NPG, it performs competitively with PPG and outperforms unidirectional path tracing in scenes with difficult light transport (bottom two rows, sorted by difficulty in ascending order). The radiance-driven PPG algorithm tends to perform best because of its low computational cost, except when incident radiance is a poor approximation of the product (top row).

not converge to local optima, but oscillate around them. This can be observed in the 2D examples in our supplementary video and also happens during neural path guiding. The problem is well known in machine learning literature and is usually addressed by decaying the learning rate over time. We opted not to decay our learning rate for simplicity, because finding an optimal decay schedule is a difficult problem. Solving this issue in the future would likely improve our results further, perhaps significantly.

Scene Scale. We studied the performance of neural path guiding when all positions that are input to it are relatively close compared to the scene bounding box. We artificially scaled the positional inputs by 10^{-5} in the COUNTRY KITCHEN scene, observing a roughly 2× larger error. While the method still outperforms path tracing by a big margin, alleviating this limitation is important future work.

Alternative Variance Reduction Techniques. In this paper, we studied the application of neural networks to importance sampling. Other variance-reduction techniques, such as control variates, could enjoy analogous benefits. We believe similar derivations to Section 5 can be made, leading to an interconnected gradient-descent-based optimization of multiple variance reduction techniques.

Alternative Training Schemes. Keller and Dahm [2019] learn near-optimal light selection probabilities for next event estimation by minimizing an approximation (via Q-learning) of the total-variation divergence. This optimization strategy and variations thereof are an interesting alternative to our KL and χ^2 divergence loss functions.

Another attractive goal is a unified optimization across multiple different scenes, rather than training from scratch for each one. A potentially fruitful extension of our approach would be to apply a higher-level optimization strategy in the spirit of "learning to learn" [Andrychowicz et al. 2016; Chen et al. 2017].

Failure Cases. In the pathologically difficult YET ANOTHER BOX scene, the theoretically inferior radiance-based NPG produces slightly better results than product-based NPG. We suspect that this is caused by the product distribution being much more complicated and therefore more difficult to learn than the radiance distribution. Furthermore, in the BOOKSHELF scene, our approaches perform worse than the GMM algorithm by Vorba et al. [2014]. Although our method exhibits fewer of such failure cases than PPG and the GMMs, an investigation into their causes is still to be carried out and could offer interesting insights; per-scene results with discussions of rendering challenges are provided in the supplementary material.

8 CONCLUSION

We introduced a technique for importance sampling with neural networks. Our approach builds partly on prior works and partly on three novel extensions: we proposed piecewise-polynomial coupling transforms that increase the modeling power of coupling layers, we introduced the one-blob encoding that helps the network to specialize its parts to different input configurations, and, finally, we derived an optimization strategy that aims at reducing the variance of Monte Carlo estimators that employ trainable probabilistic models. We demonstrated the benefits of our online-learning approach in a number of settings, ranging from canonical examples to production-oriented ones: learning the distribution of natural images and path sampling and path guiding for simulation of light transport. In the vast majority of cases, our technique performed favorably in equal sample count comparisons against prior art.

This paper brings together techniques from machine learning, developed initially for density estimation, and applications to Monte Carlo integration, with examples from the field of rendering. We hope that our work will stimulate further applications of deep neural networks to importance sampling and integration problems.

9 ACKNOWLEDGMENTS

A DETERMINANT OF COUPLING LAYERS

Here we include the derivation of the Jacobian determinant akin to Dinh et al. [2016]. The Jacobian of a single coupling layer, where $A = [\![1, d]\!]$ and $B = [\![d + 1, D]\!]$, is a block matrix:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} I_d & 0 \\ \frac{\partial C(x^B; m(x^A))}{\partial (x^A)^T} & \frac{\partial C(x^B; m(x^A))}{\partial (x^B)^T} \end{bmatrix}, \quad (31)$$

where I_d is a $d \times d$ identity matrix. The determinant of the Jacobian matrix reduces to the determinant of the lower right block. Note that the Jacobian $\frac{\partial C(x^B; m(x^A))}{\partial (x^A)^T}$ (lower left block) does not appear in the determinant, hence m can be arbitrarily complex.

For the multiply-add coupling transform [Dinh et al. 2016] we get

$$\frac{\partial C(x^B; m(x^A))}{\partial (x^B)^T} = \begin{bmatrix} e^{s_1} & & 0 \\ & \ddots & \\ 0 & & e^{s_{D-d}} \end{bmatrix}. \quad (32)$$

The diagonal nature stems from the separability of the coupling transform. The determinant of the coupling layer in the forward and the inverse pass therefore reduce to $e^{\sum s_i}$ and $e^{-\sum s_i}$, respectively.

In our piecewise-polynomial coupling transforms, we maintain separability to preserve the diagonal Jacobian, i.e.

$$C(x^B; m(x^A)) = \left(C_1(x_1^B; m(x^A)), \cdots, C_{D-d}(x_{D-d}^B; m(x^A)) \right)^T,$$

and therefore, using $\frac{\partial C_i(x_i^B; m(x^A))}{\partial x_i^B} = q_i(x_i^B)$, we get

$$\frac{\partial C(x^B; m(x^A))}{\partial (x^B)^T} = \begin{bmatrix} q_1(x_1^B) & & 0 \\ & \ddots & \\ 0 & & q_{D-d}(x_{D-d}^B) \end{bmatrix}. \quad (33)$$

The determinant thus is the product of the marginal PDFs defining the piecewise-polynomial warp along each dimension $\prod_{i=1}^{D-d} q_i(x_i^B)$.

B ADAPTIVE BIN SIZES IN PIECEWISE-LINEAR COUPLING FUNCTIONS

Without loss of generality, we investigate the simplified scenario of a one-dimensional input $A = \emptyset$ and $B = \{1\}$, a single coupling layer $L = 1$ and the KL-divergence loss function. Further, let the coupling layer admit a piecewise-linear coupling transform—i.e. it predicts a piecewise-constant PDF—with $K = 2$ bins. Let the width W of the 2 bins be controlled by trainnable parameter $\theta \in \mathbb{R}$ such that $W_1 = \theta$ and $W_2 = 1 - \theta$ and $S = Q_1\theta + Q_2(1 - \theta)$, then

$$q(x; \theta) = \begin{cases} Q_1/S & \text{if } x < \theta \\ Q_2/S & \text{otherwise.} \end{cases} \quad (34)$$

Using Equation (22) , the gradient of the KL divergence w.r.t. θ is

$$\nabla_\theta D_{\text{KL}}(p \| q; \theta) = \nabla_\theta \int_0^1 \begin{cases} p(x) \log(Q_1/S) & \text{if } x < \theta \\ p(x) \log(Q_2/S) & \text{otherwise} \end{cases} dx, \quad (35)$$

where—in contrast to our piecewise-quadratic coupling function—the gradient can *not* be moved into the integral (see Equation (23)) due to the discontinuity of q at θ. This prevents us from expressing the stochastic gradient of Monte Carlo samples with respect to θ in closed form and therefore optimizing with it.

We further investigate ignoring this limitation and performing the simplification of Equation (23) regardless, resulting in

$$\nabla_\theta D_{\text{KL}}(p \| q; \theta) \approx \mathbb{E} \left[\begin{cases} p(X)\left(1 - \frac{Q_2}{Q_1}\right) & \text{if } X < \theta \\ p(X)\left(\frac{Q_1}{Q_2} - 1\right) & \text{otherwise} \end{cases} \right], \quad (36)$$

which has the same sign *regardless of the value of* θ, resulting in divergent behavior. A similarly undesirable (albeit different) behavior emerges when normalizing q in a slightly different way by interpreting Q as probability masses rather than unnormalized densities:

$$q(x; \theta) = \begin{cases} Q_1/\theta & \text{if } x < \theta \\ Q_2/(1 - \theta) & \text{otherwise.} \end{cases} \quad (37)$$

The KL divergence gradient is then

$$\nabla_\theta D_{\mathrm{KL}}(p \parallel q; \theta) \approx \int_0^1 \begin{cases} p(x)/\theta & \text{if } x < \theta \\ p(x)/(\theta - 1) & \text{otherwise,} \end{cases} \mathrm{d}x$$

$$= \frac{1}{\theta} \int_0^\theta p(x)\, \mathrm{d}x - \frac{1}{1-\theta} \int_\theta^1 p(x)\, \mathrm{d}x. \qquad (38)$$

To illustrate the flawed nature of this gradient, consider the simple scenario of $p(x) = 1$, in which the RHS *always* equals to zero, suggesting *any* θ being a local minimum. However, θ clearly influences $D_{\mathrm{KL}}(p \parallel q; \theta)$ in this example, and therefore can not be optimal everywhere. Empirical investigations with other shapes of p, e.g. the examples from Figure 4, also suffer from a broken optimization and do not converge to a meaningful result.

While we only discuss a simplified setting here, the simplification in Equation (23) is also invalid in the *general* case of piecewise-linear coupling functions, likewise leading to a broken optimization.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. *arXiv:1606.04474* (June 2016).

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural Ordinary Differential Equations. *arXiv:1806.07366* (June 2018).

Yutian Chen, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matt Botvinick, and Nando de Freitas. 2017. Learning to Learn without Gradient Descent by Gradient Descent. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 748–756.

Ken Dahm and Alexander Keller. 2018. Learning Light Transport the Reinforced Way. In *Monte Carlo and Quasi-Monte Carlo Methods. Proceedings in Mathematics & Statistics*, Art B. Owen and Peter W. Glynn (Eds.). Vol. 241. Springer, 181–195.

Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. NICE: Non-linear independent components estimation. *arXiv:1410.8516* (Oct. 2014).

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using Real NVP. *arXiv:1605.08803* (March 2016).

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. 2015. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*. 881–889.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. 13th International Conference on Artificial Intelligence and Statistics* (May 13–15). JMLR.org, 249–256.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

Jerry Jinfeng Guo, Pablo Bauszat, Jacco Bikker, and Elmar Eisemann. 2018. Primary Sample Space Path Guiding. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, Wenzel Jakob and Toshiya Hachisuka (Eds.). The Eurographics Association.

Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. 2014. Multiplexed Metropolis Light Transport. *ACM Trans. Graph.* 33, 4, Article 100 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601138

David Money Harris and Sarah L. Harris. 2013. 3.4.2 - State Encodings. In *Digital Design and Computer Architecture* (second ed.). Morgan Kaufmann, Boston, 129–131. https://doi.org/10.1016/B978-0-12-394424-5.00002-1

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik P. A. Lensch. 2018. A Unified Manifold Framework for Efficient BRDF Sampling based on Parametric Mixture Models. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, Wenzel Jakob and Toshiya Hachisuka (Eds.). The Eurographics Association.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* (2016). https://doi.org/10.1111/cgf.12950

Heinrich Hey and Werner Purgathofer. 2002. Importance Sampling with Hemispherical Particle Footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics (SCCG '02)*. ACM, 107–114. https://doi.org/10.1145/584458.584476

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron C. Courville. 2018. Neural Autoregressive Flows. *arXiv:1804.00779* (April 2018).

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Henrik Wann Jensen. 1995. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques*. Springer Vienna, Vienna, 326–335. https://doi.org/10.1007/978-3-7091-9430-0_31

James T. Kajiya. 1986. The Rendering Equation. *Computer Graphics* 20 (1986), 143–150.

Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. 2002. A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm. *Computer Graphics Forum* 21, 3 (May 2002), 531–540. https://doi.org/10.1111/1467-8659.t01-1-00703

Alexander Keller and Ken Dahm. 2019. Integral Equations and Machine Learning. *Mathematics and Computers in Simulation* 161 (2019), 2–12.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (June 2014).

Diederik P. Kingma and Prafulla Dhariwal. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. *arXiv:1807.03039* (July 2018).

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*. 4743–4751.

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

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 3730–3738. https://doi.org/10.1109/ICCV.2015.425

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

Jacopo Pantaleoni and Eric Heitz. 2017. Notes on optimal approximations for importance sampling. *arXiv:1707.08358* (July 2017).

George Papamakarios, Iain Murray, and Theo Pavlakou. 2017. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*. 2338–2347.

Vincent Pegoraro, Carson Brownlee, Peter S. Shirley, and Steven G. Parker. 2008a. Towards Interactive Global Illumination Effects via Sequential Monte Carlo Adaptation. In *Proceedings of the 3rd IEEE Symposium on Interactive Ray Tracing*. 107–114.

Vincent Pegoraro, Ingo Wald, and Steven G. Parker. 2008b. Sequential Monte Carlo Adaptation in Low-Anisotropy Participating Media. *Computer Graphics Forum* 27, 4 (Sept. 2008), 1097–1104. https://doi.org/10.1111/j.1467-8659.2008.01247.x

Danilo Rezende and Shakir Mohamed. 2015. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning*. 1530–1538.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph.* 30, 6 (Dec. 2011). https://doi.org/10.1145/2024156.2024193

Joshua Steinhurst and Anselmo Lastra. 2006. Global Importance Sampling of Glossy Surfaces Using the Photon Map. *IEEE Symposium on Interactive Ray Tracing* (Sept. 2006), 133–138. https://doi.org/10.1109/RT.2006.280224

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016a. Wavenet: A generative model for raw audio. *arXiv:1609.03499* (Sept. 2016).

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016b. Pixel Recurrent Neural Networks. In *International Conference on Machine Learning*. 1747–1756.

Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation*. Ph.D. Dissertation. Stanford, CA, USA.

Eric Veach and Leonidas J. Guibas. 1994. Bidirectional estimators for light transport. In *EG Rendering Workshop*.

Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proc. SIGGRAPH*. 419–428. https://doi.org/10.1145/218380.218498

Petr Vévoda, Ivo Kondapaneni, and Jaroslav Křivánek. 2018. Bayesian online regression for adaptive direct illumination sampling. *ACM Trans. Graph.* 37, 4 (Aug. 2018).

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 (Aug. 2014).

Quan Zheng and Matthias Zwicker. 2018. Learning to Importance Sample in Primary Sample Space. *arXiv:1808.07840* (Sept. 2018).