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 component analysis, 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 χ² divergence for the specific application of Monte Carlo integration with stochastic estimates of the target distribution. Our approach enables fast and accurate inference and efficient sample generation independent of the dimensionality of the integration domain. We demonstrate the benefits of our approach for generating natural images and in two applications to light-transport simulation. First, we show how to learn joint path-sampling densities in primary sample space and how to importance sample multi-dimensional path prefixes thereof. Second, we use our technique to extract conditional directional densities driven by the triple product of the rendering equation, and leverage them for path guiding. In all applications, our approach yields on-par or higher performance at equal sample count than competing techniques.

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. 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, the application of multiple accuracy levels, or the utilization of quasi-random numbers.

In this work, we focus on the simple 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 Monte Carlo integration—in the form of sampling densities—remains to be investigated; this is what we strive for in the present paper.

Given an integral

\[ F = \int_{D} f(x) \, dx, \]  

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:

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

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

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

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 probability density that is 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 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 density from these observations while handling complex distributions with possibly many modes and arbitrary frequencies. To that end, we employ variational inference to 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 density estimation. 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 present piecewise-polynomial coupling layers that greatly increase the expressive power of individual coupling layers, allowing us to employ fewer of those and thereby reducing the total cost of evaluation.
After reviewing related work on variational inference and generative modeling in Section 2, we detail the framework of non-linear independent components estimation (NICE) [Dinh et al. 2014, 2016] in Section 3, which forms the foundation of our approach. In Section 4, we describe a class of invertible piecewise polynomial coupling transforms that replace affine transforms proposed in the original work, and one-blob-encoded network inputs, which stimulate localization of computation and improve inference. We illustrate the benefits on a few low-dimensional density-estimation problems and test the performance when learning the (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, we apply the proposed approach to light-transport problems in Section 6: we use NICE with our polynomial warps to guide the construction of light paths and demonstrate that it outperforms the current state of the art in primary sample space and incremental path guiding.

2 RELATED WORK

Neural networks have been successfully applied to many diverse problems—a body of research too large to be reviewed here. We thus restrict the discussion to probabilistic generative models obtained via variational inference and review only the most relevant prior works.

Generative modeling commonly considers the following probabilistic model:

\[ p(x, z; \theta) = p(x|z; \theta)p(z), \]

where \( z \) is a latent variable that is not directly observed, but controls some of the factors of variation in the observed data \( x \). \( p(x|z; \theta) \) is the likelihood function with parameters \( \theta \) and \( p(z) \) is the prior. The inferential quantity of interest is the posterior distribution of the latent variable \( p(z|x; \theta) \). Given observed data \( x \), the posterior is defined by the Bayes’ theorem,

\[ p(z|x; \theta) = \frac{p(x|z; \theta)p(z)}{\int p(x|z; \theta)p(z)\,dz}, \tag{4} \]

where the denominator is the marginal likelihood of the data \( p(x) \), and is generally intractable; we build on exceptions discussed below.

Variational methods [Jordan et al. 1999] consider approximating the true posterior distribution with another distribution \( q(z|x; \phi) \), which is chosen to have a simple functional form: commonly a Gaussian [Challis and Barber 2013] or a distribution which factorizes across variables [Ghahramani et al. 2000]. Variational inference amounts to optimizing the parameters \( \phi \), such that some divergence metric (e.g. the Kullback-Leibler (KL) divergence) between \( p(z|x; \theta) \) and \( q(z|x; \phi) \) is minimized. Variational methods do not directly estimate \( p(x; \theta) \) but instead optimize a lower bound on this quantity with respect to the approximating distribution.

Recently, generative modeling has seen a resurgence due to the development of new techniques based on neural networks. Variational autoencoders (VAEs) [Kingma and Welling 2014; Rezende et al. 2014] consider representing \( q \) as a Gaussian distribution whose mean and variance are parameterized by a neural network. The likelihood function, or decoder, \( p(x|z; \theta) \) is also a neural network whose parameters are trained jointly with the posterior \( q(z|x; \phi) \). In this setting, \( q(z|x; \phi) \) is called an inference network, or encoder: by defining a global set of network parameters to represent the latent variables it allows the cost of inference to be amortized over the entire optimization procedure. In the forward pass of the VAE, a latent variable is sampled from a standard Gaussian before being scaled and translated to match the distribution \( q \).

Rezende and Mohamed [2015] propose a more flexible class of approximating distributions based on the application of a sequence of invertible mappings to a simple base density. These normalizing flows rely on repeated application of the change-of-variables formula, and hence require the computation of the log-determinant of the Jacobian matrix of the transform. Rezende and Mohamed [2015] describe a class of planar flows for which this quantity has quadratic cost in the number of dimensions per network layer. Chen et al. [2018] propose a continuous analog of normalizing flows that applies the continuous change-of-variables formula, significantly reducing the computational cost.

A related class of models based on an auto-regressive factorization of the marginal data distribution has been proposed [Germain et al. 2015; van den Oord et al. 2016a,b]. These models exploit the general application of the chain rule of probability to decompose the joint as the product of one-dimensional conditionals \( p(x) = \prod_{i=1}^d p(x_i|x_1, \ldots, x_{i-1}) \). Auto-regressive models have been shown to perform extremely well, however, sampling is often slow due to their inherent sequential nature. Recent advances have allowed extremely fast sampling, but not without significant engineering effort [van den Oord et al. 2018]. The main drawback is that they can only evaluate the density of samples generated by the model and not of arbitrary data points. This would pose a problem in MC integration, e.g. if multiple densities are combined using multiple importance sampling [Veach and Guibas 1995].

A growing literature has emerged which investigates combinations of ideas from normalizing flows and auto-regressive density estimation [Huang et al. 2018; Kingma et al. 2016; Papamakarios et al. 2017]. Non-linear independent component estimation (NICE) [Dinh et al. 2014], which was later augmented by real-valued non-volume-preserving (RealNVP) transforms [Dinh et al. 2016], is a special case of the normalizing flows framework. It uses the same computation graph (except for direction) for the encoder and the decoder and relates the data and latent variables using bijection. The bijective mapping has the effect of modeling the likelihood and posterior conditionals as delta functions. The previously intractable marginals \( p(x) \) and \( p(z) \) suddenly become tractable (thanks to the delta functions in the integrals) and are related only through the simple change-of-variables formula. As such, this approach admits exact inference and efficient sample generation (thanks to tractable Jacobian determinants) and satisfies our aforementioned requirements for a good sampling PDF. Since our work builds directly on NICE and RealNVP, we review these in greater detail in the following section.

Finally, the zoo of generative adversarial networks (GANs) [Goodfellow et al. 2014] represent a class of implicit generative models: they do not require a likelihood, only a generative procedure to be specified. In the context of modeling images, these models are often able to generate compelling samples [Karras et al. 2017] but their lack of an explicit likelihood function makes them unsuitable for importance sampling.
3 NON-LINEAR INDEPENDENT COMPONENT ESTIMATION

In this section, we detail the works of Dinh et al. [2014, 2016] which 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 \( h = h_L \circ \cdots \circ h_2 \circ h_1 \), 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 and therefore exact and fast density estimation.

Given a differentiable mapping \( h : X \rightarrow Y \) of points \( x \sim p_X(x) \) to points \( y \in Y \), we can compute the density \( p_Y(y) \) of transformed points \( y = h(x) \) using the change-of-variables formula:

\[
p_Y(y) = p_X(x) \left| \frac{\partial h(x)}{\partial x^T} \right|^{-1},
\]

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

The cost of computing the determinant grows with the dimensionality of the Jacobian (e.g. cubically if decomposition methods are used). If \( X \) and \( Y \) are high-dimensional, computing \( p_Y(y) \) 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. This is formalized using a definition that follows from Dinh et al. [2014].

**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 \( \mathbb{R}^{\mid A\mid} \), then the output of a coupling layer \( y = (y^A, y^B) = h(x) \) is defined as

\[
y^A = x^A,
\]

\[
y^B = C(x^B; m(x^A)),
\]

where the coupling transform \( C : \mathbb{R}^{|B|} \times m(\mathbb{R}^{|A|}) \rightarrow \mathbb{R}^{|B|} \) 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:

\[
x^A = y^A,
\]

\[
x^B = C^{-1}(y^B; m(y^A)).
\]

The invertibility is crucial in our setting as we require both density estimation 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 \( B \):

\[
C(x^B; t) = x^B + t,
\]

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 \( e^s \):

\[
C(x^B; s, t) = x^B \odot e^s + t,
\]

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

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 introduce more complex coupling transforms—invertible piecewise-polynomial warps—that provide more expressive mappings and allow reducing the number of coupling layers and thereby the sample-generation and density-estimation 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 in Section 3.2. In contrast to Dinh et al. [2014, 2016], who assume \( x, y \in (-\infty, +\infty)^D \), we choose to operate in a unit hypercube, i.e. \( x, y \in [0, 1]^D \), 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, respectively.

Similarly to Dinh and colleagues, we ensure computationally tractable Jacobians via separability, i.e. \( C(x^B) = \prod_{i=1}^{|B|} C_i(x_i^B) \). 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

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 \( w = 1/K \). To define all \( |B| \) transforms at once, we instrument the network \( m(x^A) \) to predict \( |B| \times K \) matrix, denoted \( \hat{Q} \). Each \( i \)-th row of \( \hat{Q} \) defines the unnormalized probability mass function (PMF) of the warp in \( i \)-th dimension in \( x^B \); we normalize the rows using the softmax function \( \sigma \) and denote the normalized matrix \( \hat{Q}, Q_i = \sigma(Q_i) \).

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

\[
C_i(x_i^B) = \int_0^{x_i^B} q_i(t) \, dt = \alpha Q_{ik} + \sum_{k=1}^{b-1} Q_{ik} ,
\]

(12)

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 (5). Since \( C(x^B) \) is separable by definition, its Jacobian matrix is diagonal and the determinant is equal to the
The piecewise-linear PDF is modeled using $K$ error (KL divergence) and the variance obtained when using the distribution to importance-sample a Monte Carlo estimator of the average image color. The plots on the right show the training affine (multiply-add) coupling layers [Dinh et al. 2016] on low-dimensional density-estimation problems. The false-colored distributions were obtained by 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 density-estimation problems. The false-colored distributions were obtained by optimizing KL divergence with uniformly drawn samples (weighted by the reference value) over the 2D image domain. The plots on the right show the training error (KL divergence) and the variance obtained when using the distribution to importance-sample a Monte Carlo estimator of the average image color.

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.

<table>
<thead>
<tr>
<th>Affine ($L=2$)</th>
<th>Affine ($L=4$)</th>
<th>Affine ($L=16$)</th>
<th>P/w-linear</th>
<th>P/w-quadratic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>10□1</td>
<td>10□1</td>
<td>10□1</td>
<td>10□1</td>
<td>10□1</td>
<td>10□1</td>
</tr>
<tr>
<td>10□2</td>
<td>10□2</td>
<td>10□2</td>
<td>10□2</td>
<td>10□2</td>
<td>10□2</td>
</tr>
<tr>
<td>10□3</td>
<td>10□3</td>
<td>10□3</td>
<td>10□3</td>
<td>10□3</td>
<td>10□3</td>
</tr>
</tbody>
</table>

where the denominator ensures that $V_i$ represents a valid (normalized) probability density.

The PDF in dimension $i$ is defined as

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

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 piecewise-quadratic coupling transform is obtained by integration:

$$C_i(x_i^B) = \frac{\alpha^2}{2} (V_{ib+1} - V_{ib}) + \alpha V_{ib} + \sum_{k=1}^{b-1} \frac{V_{ik} + V_{ik+1}}{2} W_{ik}. \quad (16)$$

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

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 (15)).

$$V_{k,j} = \frac{\exp(V_{i,j})}{\sum_{k=1}^{K} \exp(V_{i,k}) + \exp(V_{i,k+1})} W_{i,k}, \quad (14)$$

where the denominator again denotes the bin containing the value in $i$-th dimension.
We compare the proposed piecewise-polynomial coupling transform with \( \sigma \)-uniformly we \( \sigma \) with large modeling power, and outperform even large numbers of coupling layers perform consistently better thanks to their significantly larger modeling power, and outperform even large numbers of \( \sigma \)-uniformly we \( \sigma \) with the piecewise-linear case due to the discontinuous Jacobian.

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—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 \( s \).

We compare the proposed piecewise-polynomial coupling transforms to multiply-add affine transforms [Dinh et al. 2016] on a 2D density-estimation problem in Figure 4. To produce columns 1–5, we uniformly sample the 2D domain, 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. Every per-layer network is a U-net (see Figure 3) with 4 recursive subdivisions of fully connected layers. The networks only differ in their output layer to produce the desired parameters of their respective coupling transform.

We only use adaptive bin sizes in the piecewise-quadratic coupling transforms because gradient descent fails to optimize them in the piecewise-linear case due to the discontinuous Jacobian.

When using \( L = 2 \) coupling layers, the piecewise-polynomial coupling layers perform consistently better thanks to their significantly larger modeling power, and outperform even large numbers (e.g. \( L = 16 \)) of affine coupling layers, where the total depth of the computation graph approaches 100 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 affine coupling 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 coupling layer used 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 to GPU memory constraints. Since our coupling layers operate on the \([0, 1]^D\) domain, 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 the bits-per-dimension metric (lower means better): we yield 2.85 and 2.89 bits on training and validation data, respectively, whereas Dinh et al. [2016] reported 2.97 and 3.02 bits. 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 reduce the variance of Monte Carlo integration by extracting sampling PDFs from observations of the integrand. Denoting \( q(x; \theta) \) the to-be-learned PDF for drawing samples \( \theta \) represents the trainable parameters and \( p(x) \) the ground-truth distribution of the integrand, we can rewrite the MC estimator from Equation (3) as

\[
\langle F \rangle_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{q(X_i; \theta)} = \frac{1}{N} \sum_{i=1}^N \frac{p(X_i)}{q(X_i; \theta)} f(X_i) \tag{17}
\]

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 vari-
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 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 expressed as

\[
I = \int_{\mathcal{P}} L_e(x_0, x_1) \ T(\mathcal{X}) \ W(x_{k-1}, x_k) \ d\mathcal{X}. \tag{23}
\]

The chain of positions \( \mathcal{X} = x_0 \cdots x_k \) represents a single light path with \( k \) vertices. The path throughput \( T(\mathcal{X}) \) quantifies the ability of \( \mathcal{X} \) to transport radiance. \( W(x_{k-1}, x_k) \) is the sensor response to one unit of incident radiance.

The measurement can be estimated as

\[
(I) = \frac{1}{N} \sum_{j=1}^{N} \frac{L_e(x_0, x_j) \ T(\mathcal{X}_j) \ W(x_{jk-1}, x_{jk})}{q(\mathcal{X}_j)}, \tag{24}
\]

where \( q(\mathcal{X}) \) is the joint probability density of generating all \( k \) vertices of path \( \mathcal{X} \). Drawing samples directly 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) [Guo et al. 2018; Kelemen et al. 2002], represented by a unit hypercube \( \mathcal{U} \). A path is then obtained by transforming a random vector \( z \in \mathcal{U} \) using one of the standard path-construction techniques \( \rho \) (e.g. camera tracing): \( \mathcal{X} = \rho(z) \).

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. The generic nature of PSS coordinates enables treating the path construction as a black box. Path guiding can thus be easily applied to any single path-tracing technique, and, with some effort, also to multiple strategies [Hachisuka et al. 2014; Kelemen et al. 2002; Lafortune and Willems 1995; Veach and Guibas 1994]. 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 \( \theta \) of the networks, we minimize one of the losses in Section 5 against \( r(\mathcal{X}) = L_e(x_0, x_1) \ T(\mathcal{X}) W(x_{k-1}, x_k) \ F^{-1} \), ignoring the unknown normalization factor, i.e. assuming \( F = 1 \).

In order to obtain a path sample \( \mathcal{X} \), we generate a random vector \( z \), warp it using the reversed inverted coupling layers, and apply the path-construction technique: \( \mathcal{X} = \rho \left( h_1^{-1}(\cdots h_k^{-1}(z)) \right) \); refer back to Equation (8) and (9) 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, which 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 \( x \) in direction \( \omega \) is given by the rendering equation [Kajiya 1986]:

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} L(x, \omega) f_s(x, \omega_o, \omega) \cos y \ d\omega, \tag{25}
\]

where \( f_s \) is the bidirectional scattering distribution function, \( L_o(x, \omega_o) \), \( L_e(x, \omega_o) \), and \( L(x, \omega) \) are respectively the reflected, emitted, and incident radiance, \( \Omega \) is the unit sphere, and \( y \) is the angle between \( \omega \) 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:

\[
\langle L_r(x, \omega_o) \rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{L(x, \omega_j) f_s(x, \omega_o, \omega_j) \cos y_j}{q(\omega_j | x, \omega_o)}. \tag{26}
\]

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

Since the 7D domain is fairly challenging to handle using handcrafted, spatio-directional data structures in the general case, most research has focused on the simpler 5D setting where \( q(\omega|x, \omega_o) \propto L(x, \omega) \) [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; Steinbrust 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. Our goal is also to learn accurate local sampling densities, but we utilize NICE to represent and sample from \( q(\omega|x, \omega_o) \).
Methodology. We use a single instance of NICE, which is trained and sampled from in an interleaved manner. In the most general setting, we consider learning \( q(o|x, \omega_o) \) 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 piecewise-polynomial coupling function, the neural network \( m \) takes the cylindrical coordinate from \( A \), the position \( x \) and direction \( \omega_o \) that condition the density, and any other local shading information that may improve inference; we also input the normal of the intersected shape at \( x \) to aid the network in learning distributions which correlate strongly with the local shading frame.

We one-blob encode all of the inputs as described in Section 4.3. In the case of \( x \), 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, \( h^{-1}_2(\cdots h^{-1}_i(u)) \), and transforming to the range of cylindrical coordinates to obtain \( \omega \).

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 \( \omega \), most current renderers apply multiple importance sampling (MIS) [Veach and Guibas 1995] to combine multiple sampling densities (e.g. to estimate direction illumination or importance sample the BSDF). When learning the product, we take this into account by optimizing the networks with respect to the final (MIS) PDF instead of the density learned using NICE. If certain parts of the product are already covered well by existing densities, the networks will therefore be optimized to only handle the remaining problematic case.

We therefore optimize \( D(p \parallel q') \) where \( q' = \omega q + (1 - \omega)q_f \) and \( q(o|x, \omega_o) = L(x, \omega) f(x, \omega_o, \omega) \cos y \) (we again ignore the normalization constant) and \( q_f \) is the PDF for sampling the BSDF.

We use an additional network to learn optimal \( w \) and optimize it jointly with the other networks; all networks use the same architecture. To prevent overfitting to local optima with degenerate MIS weights, we use the loss function \( \beta(i) D(p \parallel q) + (1 - \beta(i)) D(p \parallel q') \) where \( i \) is the current training iteration and \( \beta(i) = 1/2 \cdot (1/3)^{\lfloor \log_2(N) \rfloor} \).

Discussion. Our approach to product sampling has three distinct advantages. First, it is agnostic to the number of dimensions that the 2D domain is conditioned on. This allows for accounting for the full triple product \( L(x, \omega_l) f(x, \omega_o, \omega_l) \cos y \) without additional sophisticated data structures. One can simply input extra information into the neural networks and let them learn from data which dimensions are useful in which situations. While we only pass in the surface normal currently, it is conceivable that 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]. A user only needs to specify the hyper-parameters (e.g. the number of coupling layers, number of bins for the piecewise-polynomial warp, and the architecture of neural networks) as is required by most other approaches as well.

Lastly, our approach offers trivial persistence across renders. A set of networks trained on one camera view can be easily reused from a different view or within a slightly modified scene; we demonstrate this in 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 the initial training. Applying our approach to animations could thus yield super-linear cost savings.

6.3 Experimental Setup

We implemented our technique in Tensorflow [Abadi et al. 2015] and integrated it with the Mitsuba renderer [Jakob 2010]. When rendering the images, we start with trainable parameters of the networks initialized using Xavier initialization [Glorot and Bengio 2010] and 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 evaluating and sampling from \( q \), while the other trains our networks using the Monte Carlo radiance estimates from completed paths as they come in. Communication between the CPU and GPU happens via asynchronous queues to aid parallelization.

In order to obtain the final image with \( N \) samples, we use the samples since the last power-of-two sample count, discarding the first \( 2^{\lceil \log_2(N) \rceil} - 1 \) samples. 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.

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 compare the performance using equal sample count metrics that are agnostic to used hardware. Timings are provided for completeness.

We quantify the error using the mean relative squared error (MRSE), which is defined as \( \frac{1}{N} \sum_{i=1}^{N} (v_i - \hat{v}_i)^2 / (v_i^2 + \epsilon) \), 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. Additionally, we clip \( v_i \) and \( \hat{v}_i \) to the range \([0, 1]\), as the error is otherwise dominated by outliers in some scenes. We use a relative metric to avoid putting overly much emphasis on bright image regions.
Fig. 7. Neural path sampling in primary sample-space. We compare a standard uni-directional path tracer (PT-Unidir), neural path sampling using $L = 16$ affine coupling layers [Dinh et al. 2016] and $L = 4$ of our proposed piecewise-quadratic coupling layers. 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 this experiment, but only offers improvement beyond the 4D case if paths stay coherent, e.g. in the top crop of the Spaceship scene. All results were obtained by optimizing the KL divergence.
Neural Importance Sampling

<table>
<thead>
<tr>
<th>Scene</th>
<th>PT-Unidir</th>
<th>PPG-Radiance</th>
<th>Ours, KL divergence</th>
<th>Ours, $\chi^2$ div.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell Box</td>
<td>0.0613</td>
<td>0.0033</td>
<td>0.0012</td>
<td>0.0015</td>
</tr>
<tr>
<td>Spaceship</td>
<td>0.0033</td>
<td>0.0030</td>
<td>0.0005</td>
<td>0.0026</td>
</tr>
<tr>
<td>Swimming Pool</td>
<td>0.5177</td>
<td>0.0437</td>
<td>0.0197</td>
<td>0.0192</td>
</tr>
<tr>
<td>Copper Hairball</td>
<td>0.5646</td>
<td>0.0712</td>
<td>0.0660</td>
<td>0.0628</td>
</tr>
<tr>
<td>Country Kitchen</td>
<td>0.2633</td>
<td>0.0408</td>
<td>0.0225</td>
<td>0.0435</td>
</tr>
<tr>
<td>Yet Another Box</td>
<td>1.1975</td>
<td>0.0822</td>
<td>0.0153</td>
<td>0.0277</td>
</tr>
</tbody>
</table>

Fig. 8. Neural path guiding. We compare a uni-directional path tracer (PT-Unidir), the practical path guiding (PPG-Radiance) algorithm of Müller et al. [2017], and variants of our framework sampling the incident radiance alone (NPG-Radiance) or the whole integrand (NPG-Product), when optimizing either the KL and $\chi^2$ divergences. Overall, sampling the whole integrand with the KL divergence yields the most robust results. Note how optimizing the $\chi^2$ divergence tends to produce higher variance overall, but fewer outliers, in particular in the Copper Hairball scene.
Table 1. Improved performance using our proposed one-blob encoding described in Section 4.3. At equal sampling rates, our technique performs on par or better than the practical path guiding (PPG-Radiance) algorithm of Müller et al. [2017], but incurs a large computational overhead. Using one-blob encoding significantly improves the quality of our results when guiding paths by sampling the incident radiance (PG-Radiance) or the whole integrand (PG-Product); see Figure 9 for a histogram visualization of these metrics.

<table>
<thead>
<tr>
<th>Scene</th>
<th>PT-Unidir</th>
<th>PPG-Radiance</th>
<th>NPS one-blob</th>
<th>NPG-Radiance</th>
<th>NPG-Product</th>
<th>Ours, $\chi^2$ div.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bedroom</td>
<td>0.0158</td>
<td>0.0080</td>
<td>0.00101</td>
<td>0.0071</td>
<td>22m</td>
<td>0.0027</td>
</tr>
<tr>
<td>Copper Hairball</td>
<td>0.0562</td>
<td>0.0715</td>
<td>0.03080</td>
<td>0.0038</td>
<td>27m</td>
<td>0.0528</td>
</tr>
<tr>
<td>Cornell Box</td>
<td>0.0613</td>
<td>0.0033</td>
<td>0.00197</td>
<td>0.0023</td>
<td>24m</td>
<td>0.0015</td>
</tr>
<tr>
<td>Country Kitchen</td>
<td>0.2633</td>
<td>0.0408</td>
<td>0.3366</td>
<td>0.0040</td>
<td>25m</td>
<td>0.0516</td>
</tr>
<tr>
<td>Kitchenette</td>
<td>0.3498</td>
<td>0.1095</td>
<td>0.3517</td>
<td>0.1146</td>
<td>20m</td>
<td>0.0441</td>
</tr>
<tr>
<td>Necklace</td>
<td>0.1922</td>
<td>0.0968</td>
<td>0.1533</td>
<td>0.1031</td>
<td>16m</td>
<td>0.1178</td>
</tr>
<tr>
<td>Salle de Bain</td>
<td>0.0775</td>
<td>0.0101</td>
<td>0.0513</td>
<td>0.0007</td>
<td>18m</td>
<td>0.0006</td>
</tr>
<tr>
<td>SpaceShip</td>
<td>0.0013</td>
<td>8.6s</td>
<td>0.0009</td>
<td>7.8m</td>
<td>20m</td>
<td>0.0001</td>
</tr>
<tr>
<td>Swimming Pool</td>
<td>0.5177</td>
<td>0.0437</td>
<td>0.1042</td>
<td>0.0025</td>
<td>15m</td>
<td>0.0007</td>
</tr>
<tr>
<td>Staircase</td>
<td>0.0298</td>
<td>26s</td>
<td>0.0036</td>
<td>60s</td>
<td>19m</td>
<td>0.0011</td>
</tr>
<tr>
<td>White Room</td>
<td>0.0268</td>
<td>0.0103</td>
<td>0.0178</td>
<td>7.8m</td>
<td>24m</td>
<td>0.0031</td>
</tr>
<tr>
<td>Yet Another Box</td>
<td>1.1975</td>
<td>0.0822</td>
<td>0.5754</td>
<td>34m</td>
<td>1.1h</td>
<td>0.0183</td>
</tr>
</tbody>
</table>

Fig. 9. MRSE achieved by our neural importance-sampling approaches on different scenes. The bars are normalized with respect to practical path guiding [Müller et al. 2017]; height below 1 signifies better performance. Some bars exceed outside of the displayed range; Table 1 provides the actual numbers. Except for the Spaceship scene, the primary sample-space path sampling (NPS) performs significantly worse than path-guiding approaches, with the product-driven neural path guiding yielding best or second best results.

6.4 Results

In order to best illustrate the benefits of different path-guiding approaches, we compare their performance when used on top of a unidirectional path tracer that uses BSDF sampling only. While none of the results in the paper utilized next-event estimation (including prior work), 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.

In Figure 7, we study primary sample-space path sampling (dubbed 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 sampling path prefixes of 1, 2, and 3 non-specular vertices, respectively. As shown in the figure, going beyond 4D brings typically little improvement, except for the highlights in the Spaceship, where even longer paths are correlated thanks to specular interactions with the glass of the cockpit. 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.

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 practical path guiding (PPG-Radiance) by Müller et al. [2017], which is learning sampling distributions that are, in contrast to ours, proportional to incident radiance only. To ensure a fair comparison, and to disentangle the benefits of using...
We took network weights that resulted from generating images for a particular camera view, in a novel view of the scene. We also compared variants of product-driven neural path guiding optimized using the Kullback-Leibler (KL) and $\chi^2$ divergences during training. While the $\chi^2$ divergence in theory minimizes the estimator variance (see Section 5.2), it performed worse in practice according to the MRSE metric (which amounts to relative variance during training). While the $\chi^2$ divergence is that it tends to produce results with higher variance overall, but fewer outliers, e.g. in the Copper Hairball scene.

In Figure 10, we demonstrate the increased robustness of neural path guiding offered by optimizing MIS weights. 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 learned MIS weights, the system downweights 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.

Figure 11 demonstrates the benefits of reusing networks, 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 11. Similarly to training from scratch, we keep optimizing the networks and discard samples from initial iterations as the reused distributions could potentially be inaccurate if the scene geometry changed significantly.

NICE with piecewise-quadratic coupling layers, and the benefits of guiding in an MIS-aware manner proportional to the product, we created a variant of our approach that learns densities proportional to incident radiance and disregards MIS (NPG-Radiance). Except for the Spaceship, the radiance-driven neural path guiding outperforms PPG. The performance of our neural approach is further increased by learning and sampling proportional to the full triple product and incorporating MIS into the optimization—this technique yields the best results in nearly all scenes.

Table 1 reports the MRSE metric for a set of 12 tested scenes. We also compared variants of product-driven neural path guiding optimized using the Kullback-Leibler (KL) and $\chi^2$ divergences during training. While the $\chi^2$ divergence in theory minimizes the estimator variance (see Section 5.2), it performed worse in practice according to the MRSE metric (which amounts to relative variance for our unbiased estimators) on all but one of the test scenes. A notable aspect of optimizing the $\chi^2$ divergence is that it tends to produce results with higher variance overall, but fewer outliers, e.g. in the Copper Hairball scene.

We also compared variants of product-driven neural path guiding optimized using the Kullback-Leibler (KL) and $\chi^2$ divergences during training. While the $\chi^2$ divergence in theory minimizes the estimator variance (see Section 5.2), it performed worse in practice according to the MRSE metric (which amounts to relative variance for our unbiased estimators) on all but one of the test scenes. A notable aspect of optimizing the $\chi^2$ divergence is that it tends to produce results with higher variance overall, but fewer outliers, e.g. in the Copper Hairball scene.

In Figure 10, we demonstrate the increased robustness of neural path guiding offered by optimizing MIS weights. 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 learned MIS weights, the system downweights 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.

In Section 6.2, we demonstrated an application to path guiding, where the 2D density is conditioned on five other dimensions and we are thus solving many different integrals. Since we are optimizing all conditionals at once, and the normalizing factor $F$ varies between them, our arguments do not extend to this particular problem. Neglecting the normalization factors is potentially negatively influencing the optimization. We experimented with tabulating $F$ spatially, but we did not experience noticeable improvements. Nevertheless, this currently stands as a limitation of applying our work to local path guiding and it would be worth addressing in future work.

7 DISCUSSION AND FUTURE WORK

Speed. An important property of practical sampling strategies is a low computational cost of generating samples, relative to the cost of evaluating the function. In our path-guiding applications, the sample generation cost is dominated by the evaluation of coupling layers. For simple scenes, such as those in our test set where shading and ray tracing is cheap, the overhead of our method is currently prohibitive. 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.

Optimizing for Multiple Integrals. In Section 5.1, we briefly discussed the fact that the ground-truth density may be available only in unnormalized form. We suggested this not to be a problem since the ignored factor $F$ scales all gradients uniformly; it would thus not impact the optimization (at least with some optimizers). These arguments pertain to handling a single integration problem.

In Section 6.2, we demonstrated an application to path guiding, where the 2D density is conditioned on five other dimensions and we are thus solving many different integrals. Since we are optimizing all conditionals at once, and the normalizing factor $F$ varies between them, our arguments do not extend to this particular problem. Neglecting the normalization factors is potentially negatively influencing the optimization. We experimented with tabulating $F$ spatially, but we did not experience noticeable improvements. Nevertheless, this currently stands as a limitation of applying our work to local path guiding and it would be worth addressing in future work.
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 each case, our technique outperformed the methods that we compared against.

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

We thank Yining Karl Li for proofreading, Laurent Din, Jascha Sohl-Dickstein, and Samy Bengio for releasing the source code of their work [Dinh et al. 2016] and Thüjs Vogels for bringing their paper to our attention. We also thank the following people for providing scenes and models that appear in our figures: Benedikt Bitterli [2016], Ondřej Karlík (NECKLACE), Samuli Laine and Olesya Jakob (COPPER HAIRBALL), Jay-Artist (COUNTRY KITCHEN, WHITEROOM), Nacimus (SALLE DE BAIN), SlyDrako (BEDROOM), thecali (SPACESHIP), and Wig42 (STAIRCASE). Production baby: Vít Novák.

A DETERMINANT OF COUPLING LAYERS

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

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial C(x^B, \mu(x^A))}{\partial x^A} & \frac{\partial C(x^B, \mu(x^A))}{\partial x^B} \end{bmatrix},$$

(27)

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 $\frac{\partial C(x^B, \mu(x^A))}{\partial x^A}$ (lower left block) does not appear in the determinant, hence $\mu$ can be arbitrarily complex.

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

$$\frac{\partial C(x^B, \mu(x^A))}{\partial (x^B)^T} = \begin{bmatrix} e^x \cdot \cdots & 0 \\ \cdots & e^{SD-d} \cdot \end{bmatrix}.$$


