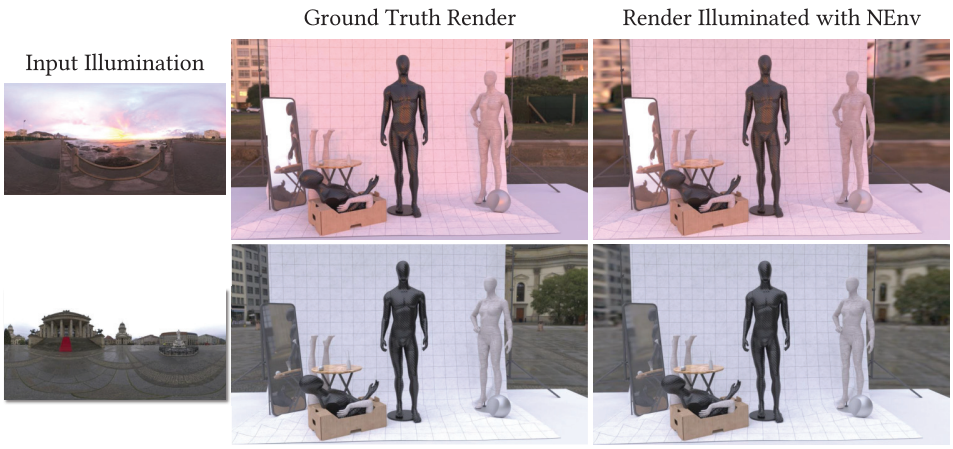
CHAPTER 7. NEURAL MODELS FOR GLOBAL ILLUMINATION

Figure 7.1: Renders using analytical illumination methods with multiple importance sampling and our learned lighting approach for two environment maps.



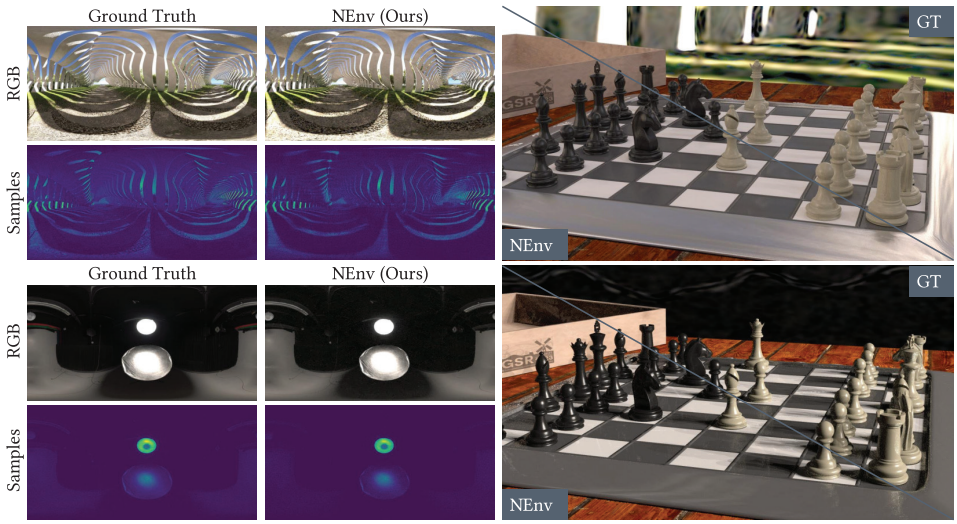
Environment maps are commonly used to represent and compute far-field illumination in virtual scenes. However, they are expensive to evaluate and sample from, limiting their applicability to real-time rendering. Previous methods have focused on compression through spherical-domain approximations, or on learning priors for natural, day-light illumination. These hinder both accuracy and generality, and do not provide the probability information required for importance-sampling Monte Carlo integration. In this chapter, we propose NEnv, a deep-learning fully-differentiable method, capable of compressing and learning to sample from a single environment map. NEnv is composed of two different neural networks: A normalizing flow, able to map samples from uniform distributions to directions of the environment map, also providing their corresponding probabilities; and an implicit neural representation which compresses the environment map into an efficient differentiable function. The computation time of environment samples with NEnv is two orders of magnitude less than with traditional methods. NEnv makes no assumptions regarding environment map content, achieving higher generality than previous learning-based approaches. The contributions in this chapter led to the following publication, currently under review:

"NEnv: Neural Environment Maps for Global Illumination"

Carlos Rodriguez-Pardo, Javier Fabre, Elena Garces, Jorge Lopez-Moreno

Under Review (2023)

Figure 7.2: From left to right, Ground Truth and our learned RGB environment map compression and sampling algorithms, and renders using both illumination configurations on a complex scene.



7.1 Introduction

Environment Maps are widely used in rendering to represent far-field illumination around a point with a single texture, usually a High Dynamic Range image (HDRi). There are two typical scenarios for using these maps: as infinite spherical light sources in offline rendering methods (such as path tracing), or as local probes that encode near-field irradiance to approximate global illumination in real-time applications.

In the first case, many algorithms such as Multiple Importance Sampling (MIS) [VG95] require the ability to sample lights in the scene, which can be problematic when dealing with environment maps. Since the source data comes from an image, there is no analytic Cumulative Distribution Function (CDF) that can be used for sampling. Instead, tabulated methods can be used, but these require a pre-computation step that consumes a considerable amount of memory, as well as a search step to find the sample probability.

In real-time scenarios, irradiance sampling is becoming more relevant with the increasing capabilities of GPUs and sample reservoir techniques for global illumination [Bit+20; Ouy+21], yet the usual techniques to sample environment maps do not easily benefit from parallel GPU environments. Also, in real-time, analytical approximations (such as Spherical Harmonics or Spherical Gaussians) are often used to overcome this issue, however, they are not able to capture final-scale details present in most environment maps.

This work presents a novel learning-based method for sampling, PDF evaluation, and compression of high-resolution environment maps, able to encode any type of illumination with high fidelity. Our method builds on recent advances in neural representations, that we found particularly suitable for this kind of problem, where it is important to encode both the high-frequency and low-frequency patterns. Key to our solution is to use a normalizing flow [Dur+19a] to encode an *invertible* representation of the environment map PDF, reducing both sampling and evaluation time by two orders of magnitude compared with analytic solutions. Thanks to being invertible, our representation is differentiable and compatible with importance-sampling techniques. We also propose a method based on implicit neural representations to compress the environment map, reducing the memory footprint of the original image with minimal loss. We demonstrate that our approach works for a wide range of scenes and is faster and more accurate than previous methods.

7.2 Related Work

In offline rendering, environment maps are usually encoded in high resolution (> 4*K*) HDR images (Radiance HDR [LS98], OpenEXR [KBH03]), requiring tabulated approximations for sampling and evaluation. Techniques such as Piecewise–Constant 2D Distributions [PJH16] or Hierarchical Warping [Cla+21] are used for this purpose, but they are not suited for RT applications due to limited time and memory budget, being a costly and difficult-to-parallelize operation, even for offline rendering. This cost can be reduced, for instance, by taking into account scene information, such as occluders [Aga+03] to avoid poor-quality samples.

Heitz [Hei20] proposed a method to invert non–analytically invertible CDFs, which can be applied to some distribution functions (e.g.: BSDFs) to obtain an analytical sampling function. However, environment maps are not good candidates for triangle-cut parametrization, since their PDF and CDF come from a discrete tabulated source (HDR image).

To avoid using high resolution images, real time methods rely on environment map prefiltering [Kau+00], or analytical approximations, such as Spherical Harmonics (SH) [See66], and Spherical Gaussians (SG) [Xu+13]. These methods allow for analytical evaluation of an environment map, as well as analytical sampling (directly, in the case of SG, or using Hierarchical Sampling for SH [JCJ09]) and easy interpolation (useful when not enough samples can be requested). Still, they do not accurately represent complex HDR images commonly used when representing high–detailed light setups by using environment maps, and blending important light features in the original images, sometimes generating artifacts. More recently, neural representations have been explored for this purpose. For instance, Gardner *et al.* [GES22] propose a conditional neural field representation, based on a variational auto-decoder (RENI), which leverages natural image priors to efficiently encode a full HDR environment image into a few-dimensional latent space vector. Being low-dimensional, it shares the same accuracy limitations that its predecessors, and does not consider sampling, or evaluation in its design. In the following, we describe the most relevant neural approaches for sampling and 2D image representations.

7.2.1 Neural Sampling and Representations

Learned Sampling and Normalizing Flows Normalizing Flows (NFs) have been proposed as powerful models for learned sampling for rendering applications. The seminal work of Müller *et al.* [Mül+19] proposed *Neural Importance Sampling*, using *Piecewise-Linear* and *Piecewise-Quadratic Coupling Flows* for learned sampling for Monte Carlo rendering. This method showed the first successful use of these neural models for Path-Tracing, but their results showed a prohibitive overhead in runtime cost over previous methods. In a later work, Müller *et al.*proposed *Neural Control Variates* [Mül+20], an *Autorregresive Flow* which allows for more efficient unbiased integration. Beyond Monte Carlo integration, Flows have been used for BRDF representations in inverse rendering [CNN22]. Relatedly, Sztrajman *et al.* [Szt+21] propose a method for neural sampling of BRDFs. However, instead of relying on NFs, they propose an encoder network that maps from *Neural BRDF* to fitted Blinn-Phong parameters for which importance sampling is known. Besides these tasks, NFs have been utilized in computer graphics and vision for image [KD18; WZY22] and video [Kum+19] generation, compression [Hel+20], super-resolution [Lug+20], domain translation [Gro+20], and uncertainty quantification [Wan+22c; Sel+20]. We build upon *Neural Importance Sampling*

[Mül+19] and propose a lightweight model for sampling and PDF evaluation of environment maps, as we describe in Section 7.4. We show that the pro-posed architecture and training produce a single network that can be integrated into any rendering pipeline, providing a significant speedup in image-based lighting with importance sampling.

Implicit Neural Representations (INRs) have emerged in recent years as a powerful alternative to traditional representations for natural signals. They allow for differentiable, continuous, and expressive functions which can represent many types of data, such as images [Sit+20; Meh+21], video [Gao+21], or *Neural Fields* for rendering [Mil+20]. INRs typically build upon standard Multi Layer Perceptrons (MLPs) but benefit from working on the frequency domain to better handle higher frequencies, which are present on natural signals. These can be introduced to the models using fixed *Positional Encodings* [Mil+20; Bar+21], *Fourier Features* [Tan+20; Kar+21] or *Sinusoidal Activations* [GES22; Meh+21; Sit+20], among other approaches. These general-purpose models have been utilized extensively in inverse and neural rendering, for representing geometry [Tak+21; Yan+21], illumination [GES22; GMX22; Att+22], material reflectance [Baa+22; Fan+22; Szt+21; Kuz+22; Kuz+21; Yao+22] or entire scenes [Mil+20; Fri+22; Tan+22; Ver+22; Sit+21]. We refer the reader to the seminal surveys in *Neural Fields* [Xie+22; Tew+22] for more comprehensive reviews of these representations for neural rendering and computer vision. Besides their effectiveness for rendering, INRs have also been used for image [Str+22] and video [Rho+22] compression. We build upon these findings to design an efficient yet expressive INR for environment maps, as described in Section 7.5.

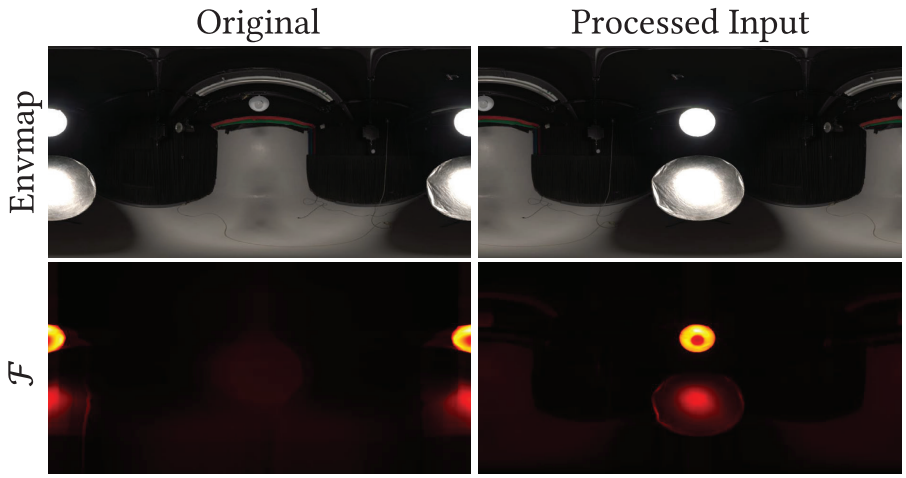
Figure 7.3: Overview of *NEnv*. We propose an invertible generative neural network $\boldsymbol{F}$ (i.e. a *normalizing flow*) which simultaneously learns to sample directions (θ, ϕ) from an environment map, as well as evaluating the probability of a given direction. This model, illustrated on the left, can be integrated seamlessly into path tracer render engines, allowing for efficient multiple importance sampling. We additionally use an implicit neural representation c (second column) which learns to map from directions to linear RGB values, allowing for faster evaluation.

7.3 Overview

Our method takes as input a single HDRi environment map of any resolution, which is an image-based 360º representation of the illumination of a scene $I \in \mathbb{R}^{H \times W \times 3}$. First, we perform a series of pre-processing steps (Section 7.3.1) and compute the ground truth probability distribution function,(Section 7.3.2). Then, we train a normalizing flow $\mathbf{F}$ for joint sampling and PDF evaluation (Section 7.4), and an implicit neural representation c for environment map compression (Section 7.5). We illustrate NEnv in Figure 7.3.

$\boldsymbol{F}$ is an invertible neural network which can generate directional samples $(\theta,\phi) = F(\mathrm{z})$ by mapping from a known and simple noise distribution $\mathrm{z} \in \mathbb{R}^2, \mathrm{z} \sim p_z(\mathrm{z})$, as well as evaluating the probability density of any direction $p(\theta, \phi)$. F provides an efficient and differentiable approximation of the PDF encoded in I, which is particularly useful for Monte Carlo integration in rendering systems using Multiple Importance Sampling (MIS). We model c as a sinusoidal Multilayer Perceptron (MLP) which maps directions (θ, ϕ) to RGB values: $RGB = C(\theta, \phi)$; compressing I into a differentiable, continuous function. We train $\boldsymbol{F}$ and $\mathbf{c}$ individually for each input $\boldsymbol{I}$, and design their architectures and training configurations to achieve high compression rates, efficient evaluation, and minimal loss in rendering quality. Once trained, both $\boldsymbol{F}$ and $\boldsymbol{C}$ can be integrated seamlessly into rendering engines, considerably reducing the time and memory consumption per sample compared to traditional global illumination approaches.

Figure 7.4: Environment map and the PDF encoded by NEnv for an unprocessed (left) and rotated (right) environment map. Our simple rotation preprocessing avoids discontinuities in the borders of the learned reconstructions, enhancing the performance of our models.



Implementation details are included in Section 7.6. To validate our method, we perform extensive ablation studies, and perform comparisons with previous work on environment map approximations (Section 7.8), both in environment map reconstruction and its impact on final renders.

7.3.1 Map Pre-processing

Some environment maps have large and intense light sources close to the horizontal borders of the image, separating the same light source into extreme values of $\phi \approx 0$ and $\phi \approx 2\pi$. While this is not a problem when rendering with traditional approaches, we observe these cases become very challenging for our normalizing flows, creating artifacts and tails in the encoded PDFs, as we show in Figure 7.4. We solve this by rotating the environment map by $\pi - \phi_{max}$ angles, so that its most intense light source is in $\phi = \pi$, thus avoiding strong discontinuities at the borders:

$$\phi_{\text{max}} = \underset{\phi}{\text{argmax}} \sum_{\theta} \mathcal{I}\left(\theta, \phi\right) \tag{7.1}$$

While there are specific parameterizations for normalizing flows on spheres [Rez+20], we empirically observe that this rotation algorithm effectively solves the discontinuity issues we observed on real-world environment maps without requiring any modifications in our model design.

7.3.2 Computing Ground Truth Tabular PDFs

As mentioned, there are two widely–used approaches for sampling directions according to an environment map distribution:

- Computing a *Piecewise-Constant* [PJH16] 2D Distribution by sampling a marginal PDF to select a row/column of I, then sampling a conditional PDF to choose an individual pixel.
- A Hierarchical Warping [Cla+21; Pha19] algorithm based on MIP–mapping over I to iteratively warp 2D samples from an uniform space until they match the target distribution.

Pharr [Pha19] compared these methods, stating that despite obtaining different spatial warpings, both produce similar error in renders. Interestingly, the official implementation for Mitsuba3 [Jak+22] relies on the latter approach, while the available implementation for the 4th edition of PBRT [PJH20] uses the former. These are both widely–used engines in the literature. Due to its comparably better efficiency (see Section 7.8.3), which is important during model training, we choose to use the Piecewise-Constant method to compute our ground truth PDFs and samples.

To obtain the PDF for a given environment map I, we first define $f(\theta, \phi)$, by mapping directions to the luminance values stored in the image I. These values are proportional to the probability of selecting a particular pixel that we want to compute, but require additional processing and normalization. We start by multiplying I by $\sin(\theta)$, given the θ corresponding to each row, thus eliminating the distortion caused by mapping the image to the unit sphere [PJH16]. Using $f(\theta, \phi)$ as source data, we then compute one conditional density distribution $f_{conditional_\theta}(\phi)$ for each ϕ, so we can later obtain a single marginal distribution by integrating all conditional distributions (rows) to build 1D Piecewise distributions. We use this marginal distribution to sample a row, and in turn, sample the corresponding conditional distribution stored for such row to select a column, hence getting a single pixel defined by θ, ϕ coordinates.

Finally, to compute the probability density of choosing a given pixel $p_I(\theta, \phi)$, we obtain the CDF for the conditional distribution at row φ for value θ, and normalize it to the marginal distribution integral:

$$p_{\mathcal{I}}(\theta, \phi) = \frac{f_{conditional_\theta}(\phi)}{\sum_{i=0}^{n_\theta} f_{\text{marginal}}(i)} \tag{7.2}$$

where $f_{conditional_\theta}$ represents the data used to compute the conditional PDF for row ϕ, and $f_{marginal}$ is the data used to compute the marginal distribution over the n_θ rows of the Y axis.

7.4 Learned Sampling and PDF evaluation

We aim to find a learning-based representation useful to sample directions from an environment map following its distribution, as well as evaluating the probability density (PDF) for any given direction $p(\theta, \phi)$. While this could be approached using two independent neural networks, there is no guarantee that a sample generated by a network is drawn with the probability estimated by the other network, surely affecting the illumination integral and pro-ducing visual artifacts (*e.g.* fireflies). Neural networks by default are typically non-invertible and can become prohibitively expensive once model sizes grow, either by stacking more layers or adding more neurons. We instead leverage *normalizing flows*, defined by specifying a bijective function F or its inverse F^{-1}. This enables both sampling and evaluation simultaneously and coherently with a single invertible neural network.

7.4.1 Formalization

We define our normalizing flow as a differentiable transformation F which maps from a random vector $\mathrm{z} \in \mathbb{R}^2$, sampled from a 2-dimensional uniform distribution $\mathrm{p}_z(z)$, to samples $(\theta,\boldsymbol{\phi}),\theta\in[0,\pi)),\boldsymbol{\phi}\in[0,2\pi)$:

$$(\theta, \phi) = \mathcal{F}(z) \text{ where } z \sim p_z(z) \tag{7.3}$$

The probability density of a direction $p_F(\theta,\phi) \in \mathbb{R}$ under the normalizing flow F is obtained through a change of variables. This can be computed analytically because F is designed as an invertible function. Formally:

$$p_{\mathcal{F}}(\theta, \phi) = p_z\left(\mathcal{F}^{-1}(\theta, \phi)\right) \left| \det \frac{\delta \mathcal{F}^{-1}}{\delta(\theta, \phi)} \right| \tag{7.3}$$

$\mathbf{F}$ locally transforms the density of original noise distribution $pz(z)$. The intensity of this change if measured by the determinant of the Jacobian of the transformation, as in the right hand side of the equation above. $p_z(z)$ is chosen to be a bidimensional uniform distribution bounded between 0 and 1: $p_z(z) = \mathrm{V}^2(0, 1)$.

As shown, F provides the two calculations required in MIS for Monte Carlo integration. However, F needs to meet two competing requirements: it has to be as accurate as possible, *i.e.*, close to the real PDF, while being computationally efficient, both in evaluation and sampling. These competing requirements define our design choices, which we motivate next.

7.4.2 Design of F

The design of F follows three principles: First, to achieve accurate renderings, we need it to be expressive enough to represent any complex probability density distribution. Second, we need an analytic form to compute its inverse F^{-1}. Finally, for efficiency, computing F, F^{-1}, and its Jacobian determinant needs to be as fast as possible.

Coupling Flows [DKB14; DSB16; KD18] achieve the efficiency requirements by using a single feed-forward pass of F and F^{-1}, and preserving a reduced level of computation, required to evaluate the Jacobian determinant, which is a lower triangular matrix for this type of flows. However, because all computation is done on a single pass to the models, coupling flows are typically less expressive than *Autoregressive Flows* [Mül+20; Kin+16; PPM17; Hua+18a; PSM19; Khe+21], which in turn, are less efficient during test time. We refer the reader to recent surveys [KPB20; Pap+21] for more comprehensive analyses of these topics.

To balance between efficiency and accuracy, we design F as a *Coupling Flow* and incorporate expressiveness through the coupling layer design. To this end, we evaluate several design choices available in the literature in the context of our problem. Recent work include *Affine Coupling* layers [DSB16], *Piecewise Linear* and *Piecewise Quadratic* [Mül+19], *Cubic-Spline Flows* [Dur+19b] and *Rational-Quadratic Spline Flows* [Dur+19a]. The type of coupling controls the

complexity of F, which can further be tuned by modifying the number of *coupling layers*, the *width* and *depth* of the neural network that define each coupling, and the number of *bins* which define the complexity of the polynomial function inside each coupling layer.

In addition, as in any neural network, it is possible to tweak the internal normalization layers, activation functions, optimizer type and configuration, regularization choices, etc. These create a combinatorial explosion of design choices, with two competing objectives of accuracy and efficiency. We found that *Rational-Quadratic Spline Flows* [Dur+19a] with a reduced amount of small coupling layers are expressive to a sufficient degree for every environment map that we tested, while being very efficient during evaluation, achieving orders of magnitude less computational cost than tabulated sampling. Because F is efficient and fully differentiable, it could potentially be incorporated into inverse rendering problems. We specify our model design choices in Section 7.6 and evaluate them in Section 7.8.1.

7.4.3 Training F

We train our normalizing flow to minimize the aggregated negative log-likelihood of a set of samples, $\mathcal{B} = \left\{(\theta, \phi)^{(n)}\right\}_{n=1}^{N}$, that are drawn dynamically in batches of size N from the input environment map I as described in Section 7.3.2.

During each training step, we minimize the average negative log likelihood across the elements in the batch $\frac{1}{N}\sum_{b}^{N}\mathcal{L}_{nll}(\theta, \phi)^{(b)}$ where:

$$\mathcal{L}_{nll}(\theta, \phi) = -\log\left(p\mathcal{F}(\theta, \phi)\right) = -\log\left(p_z\left(\mathcal{F}^{-1}(\theta, \phi)\right)\left|\det\frac{\delta\mathcal{F}^{-1}}{\delta(\theta, \phi)}\right|\right) \quad (7.5)$$

This is done by learning to map from the distribution of samples to the noise distribution $p_z(z)$. Given a sufficient number of training batches, F learns to accurately sample from the distribution of samples encoded in the input environment map I. Because F is invertible and due to our efficient model design, the probability of a sample (θ, ϕ) can easily be computed by its inverse F^{-1}.

During training, we observe that most of the computation time is used to generate the ground truth batches of samples B, which is the very process that we are interested in improving with F. We also notice that achieving a training procedure that works for every environment map is challenging due to gradient instabilities. We introduce a series of modifications to the training procedure to allow for stable training dynamics and avoid NANs for any input environment map, which we specify in Section 7.6.

7.5 Environment Map Compression

In addition to learning to sample and evaluate the PDF, we train a compression neural net-work $\boldsymbol{C}$ which learns to map from directions to RGB values: $\boldsymbol{C}(\theta,\phi) = \boldsymbol{RGB}$, $\theta \in [0,\pi)$, $\phi \in [0,2\pi)$,$\boldsymbol{RGB} \in \mathbb{R}^{+3}$. This serves two purposes: First, it provides a memory and time efficient representation of the environment map, thus further improving the render times and com-putational cost. Second, it transforms the tabulated representation into a continuous and differentiable function for which gradients can be computed, which may prove useful for inverse rendering scenarios. For the model design of $\boldsymbol{C}$, we build upon previous work on implicit neural representations. In particular, neural network architectures that work on the frequency domain have shown increased performance with respect to MLP for modeling natural signals [Tan+20; Sit+20], both in reconstruction quality and parameter efficiency. We thus model $\boldsymbol{C}$ as a shallow *sinusoidal MLP*, a *SIREN* [Sit+20]. This architectural design has been explored in recent work on environment map approximation, such as *RENI* [GES22]. However, as we do not aim to learn any prior over natural illumination, we can remove some constraints introduced in *RENI*, most notably the *rotation equivariance* requirement. Further, we do not use any latent vector to condition the output of our model, nor we do require *Vector Neurons* for our representation. With our simplifications, we achieve much higher reconstruction quality with the same parameter count. We train $\boldsymbol{C}$ on linear RGB space, with a pixel-wise reconstruction loss. Specifically, we use an $\boldsymbol{L}_1$ loss, as it produces sharper and more accurate reconstructions than higher-order norms, such as $\boldsymbol{L}_2$ [Iso+17; RG21]. We also observe strong gradient instabilities when training with $\boldsymbol{L}_2$ on linear RGB. As c expects inputs in radians, but the input image has ranges of $H \times W$ pixels, our full loss becomes:

$$\mathcal{L}_{recon} = \frac{1}{H \times W} \sum_{i}^{H} \sum_{j}^{W} \left\| C\left(\frac{i \times \pi}{H}, \frac{j \times 2\pi}{W} \right), \mathcal{I}(i, j) \right\| 1 \tag{7.6}$$

We use mini-batching to train our compression models $\boldsymbol{C}$, using uniformly sampled directions θ, ϕ. Previous work [GES22; Szt+21; Rod+23a; Lav+21] introduce adaptive sampling, cosine weighting or specular peak attenuation to similar loss functions for environment map or BRDF reconstruction. However, for our particular problem, we empirically observed that these additions tend to contribute negatively either to the training dynamics, or to the final quality of the reconstruction.

7.6 Implementation Details

Preprocessing Before the rotation algorithm described In Section 7.3.1, we resize every input HDRi environment map to a resolution of (2000, 4000) pixels using area interpolation.

This is an optional step, but it helps us standardize our experiments so that input resolution does not change training times. Note that evaluation times of our NEnv models depend on the model sizes and parameterizations, not on the resolution they were trained on.

Model Design and Training Our models sizes, loss functions, coupling layer design, normalization, optimizer and training configurations were selected using a combination of manual tuning and Bayesian hyperparameter optimization using *Weights and Biases* [Bie20] for a variety of representative environment maps.

We train the models using PyTorch [Pas+19] 1.11 and Torchvision [MR10]. Our flow and compression networks are trained independently, and individually for each input environment map. We empirically observe that mixed precision training [Mic+18] introduces strong training instabilities for this problem, so we train the models using *float32* precision. However, our models can be evaluated using half precision, which significantly increases their efficiency during test time. F and C each use around 3 MBs each, while a (2000, 4000) resolution HDRi environment map uses around 25 MBs.

Normalizing Flow Our normalizing flows F build upon the official implementation in [Dur+19a]. We use Batch Normalization [IS15] in the residual layers of our flows. To maximize evaluation efficiency, our flows have a limited number of trainable parameters. In particular, we use only 2 coupling layers, with 2 hidden layers, each with 256 hidden units and 256 *bins*. We use spline flows as our coupling layer type [Dur+19a]. Our initial learning rate is of $5e-4$, which is halved every 2500 iterations. We use Adam [KB15] as our optimizer. To stabilize training, we use gradient norm clipping [Zha+20] (*maxnorm* = 1). This is crucial for well-behaved models and efficient convergence. We train our models with a batch size of 100000 for 15000 epochs, which takes around 2 hours on an Nvidia RTX 3060 GPU. Note that fewer iterations (eg 5000) are typically sufficient for adequate results, while longer training helps to resolve additional details.

Compression Network We use a small sinusoidal MLP as our environment map compression network C. We use the official implementation in [Sit+20]. We train our models with a batch size of 500000 of randomly sampled directions and linear RGB values, for 10000 epochs, which takes around 0.5 hours on an Nvidia RTX 3060 GPU. Our initial learning rate is $5e-4$, which is halved every 2000 iterations. We use Adam [KB15] as our optimizer. To stabilize training, we use gradient norm clipping (*maxnorm* = 50). Our compression networks have 3 layers with 512 hidden units each. We initialize their weights as described in [Sit+20].

Rendering We use our own path tracer (PT) rendering pipeline as a baseline engine. This PT uses an implementation inspired by *Wavefront* [Van11;

LKA13], powered by *Embree* [Wal+14]. We distribute render work in tiles of (25,25) pixels that perform path–tracing in usual Wave-front steps (ray generation, intersection, shading, connection), each tile using a separate CPU thread. Note that tile size could be changed, as our selected tile dimensions are fixed due to hardware limitations and the amount of current thread active simultaneously (Intel Core i7-7700K CPU @ 4.20GHz, 8 threads).

Under this architecture, we perform our environment map sampling step once per tile, before every pixel in each tile performs its connection step. This way, we can either use a CPU approach [Cla+21] per pixel or integrate a single call to F to obtain all samples that are required in the current tile. We use *LibTorch* to use our PyTorch implementation from our C++ code. Intersections are performed in a single call using a ray packet call from Embree. Integrating C is more challenging, since we require ray data. However, we can store all this information simultaneously: We prepare all rays for intersection, and perform a single call to C. This way, we obtain all values at once. Note that these evaluations will be discarded if the next path–tracing bounce does not go out of bounds.

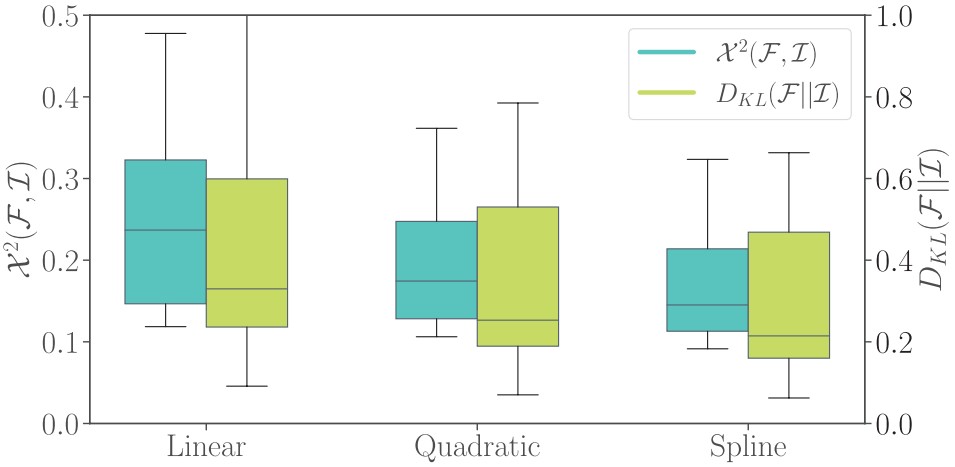
7.7 Dataset

Our goal is to propose a method that works for any type of global illumination which can be represented using environment maps. To this end, we gather a dataset of 32 high resolution HDRi environment maps from publicly available sources, notably HDRMaps and Poly Haven, to thoroughly evaluate our models. Our dataset contains natural daylight, including sunny midday illumination, cloudy diffuse images, sunset, and dusk. Further, we also test our method on indoor scenes, from studio lighting, to a concert hall or cathedral illuminations. We purposefully include very challenging cases with multiple colored light sources, which helps us understand the limitations and capabilities of previous work and our models.

7.8 Evaluation

In this section, we evaluate our models, both quantitatively and qualitatively, We first measure their performance in terms of reconstructing the probability density and the RGB values of the input environment maps, and in computational cost. Finally, we measure their integrated quality in final rendered images compared to a variety of baselines. Unless stated otherwise, we use the full dataset described in Section 7.7 for our analyses.

Figure 7.5: PDF X^2 and KL Divergence [KL51] of normalizing flows using different coupling layer types, including Linear and Quadratic Couplings from [Mül+19] and Spline Layers [Dur+19a].



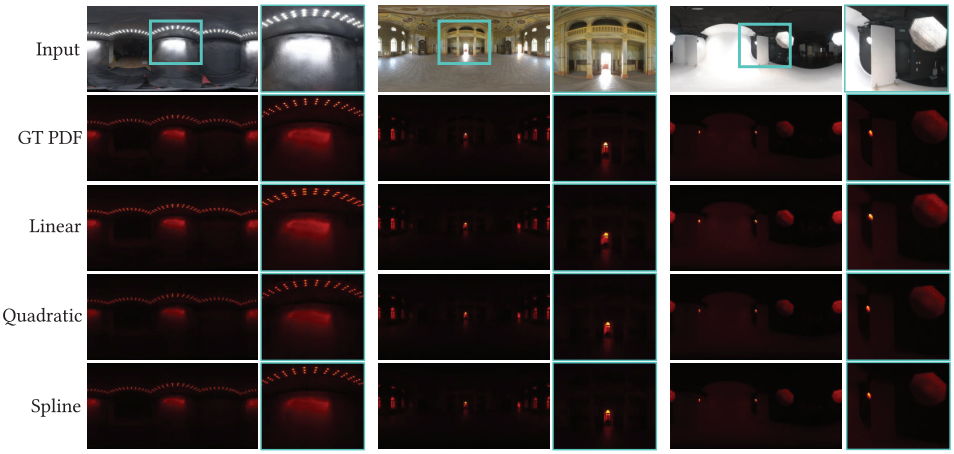
To make comparisons fair, we use the following configuration: For RENI [GES22], we train a new model from scratch with the exact same architecture design as our C, and train it for 3000 epochs. For Spherical Harmonics (SH) [See66; RH01b], we use 1024 coefficients, and for Spherical Gaussians (SG) [Xu+13], 16 × 32 dimensions. These configurations have a larger amount of parameters than what is typically used for these methods, especially in real time rendering. However, our goal is to maximize the reconstruction quality for each method and environment map, even if it results in an increased memory footprint. For the three methods we evaluate, the output resolution is set to (256, 512) pixels. We follow the official implementation of [GES22] for all these comparisons.

7.8.1 PDF Fit Accuracy

To validate our design choices for the coupling layers of our normalizing flow F, we train different versions of F for every environment map in our dataset, exclusively changing the type of polynomial function in the coupling layer. We measure the X^2 distance, as well as the KL divergence $D_{\mathrm{KL}}(F||I)$ [KL51] between the probabilities encoded by our normalizing flow p_{F} and the ground truth counterparts p_{I} as follows:

$$\mathcal{D}_{KL}\left(\mathcal{F}\middle\|\mathcal{I}\right)=\sum_{\theta_i=0}^{\pi}\sum_{\phi_j=0}^{2\pi}p\mathcal{F}\left(\theta_i,\phi_j\right)\log\left(\frac{p\mathcal{F}\left(\theta_i,\phi_j\right)}{p\mathcal{I}\left(\theta_i,\phi_j\right)}\right) \tag{7.7}$$

Figure 7.6: A qualitative comparison between the type of coupling layer used in our normalizing flows. On the top two rows, we show the input environment maps and their ground truth PDFs. Using the linear and quadratic couplings defined in [Mül+19] we achieve somewhat accurate encodings. With spline flows [Dur+19a], we achieve sharper and more accurate probability distributions. We use a $\gamma = 2.2$ to tonemap the RGB and PDF maps to help visualization. We highlight relevant regions using blue insets. Best viewed in color on a screen.

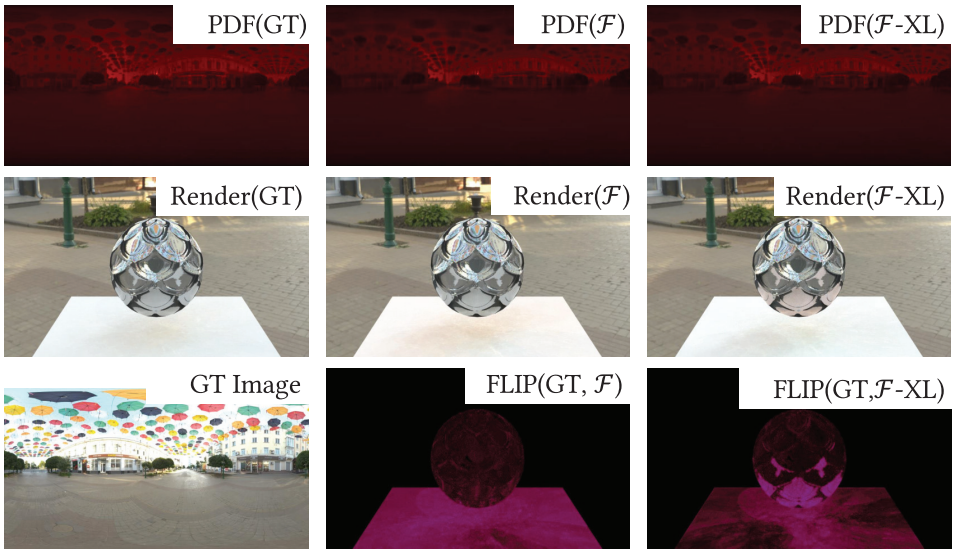


This metric, also used in [Mül+19], measures the distance between the learned probability distribution and the real PDF encoded in the environment map. In Figure 7.5, we show the aggregated divergences for our dataset for three different coupling layer types. As shown, the *Piecewise Linear* coupling proposed in [Mül+19] is outperformed by its more expressive *Piecewise Quadratic* variation, while the *Rational-Quadratic Spline Flows* proposed in [Dur+19a] achieve the lowest divergence overall, albeit by a relatively small margin. This improvement is achieved solely by adding expressiveness to the coupling layers, without any additional parameter cost. We provide a qualitative evaluation in Figure 7.6, where we show that linear and quadratic coupling layers encode less detailed probability distributions. These differences are particularly relevant for inputs with multiple distant light sources.

We also evaluate whether a larger model can yield significant improvements in accuracy. In Figure 7.7, we show the results of a large flow (F-XL), with twice as many layers and neurons per layer, a total of approximately 7 times more trainable parameters and model size in memory. Even though it indeed reduces the divergence from $D_{KL}(F \,||\, \mathrm{I}) = 0.308$ to $D_{KL}(F\text{–XL}||\mathrm{I}) = 0.281$, this model takes approximately one day to train and makes sampling 3.43 times slower. This larger model only

achieves a small gain in render accuracy (from 0.059 to 0.048 in FLIP [And+20]), suggesting that larger models exhibit diminishing returns in terms of quality. We believe our chosen model sizes adequately balance accuracy and speed. However, for practitioners that would like better accuracy at the cost of computational performance, we suggest using more coupling layers and more bins, as they tend to be the most effective design decisions beyond the coupling type.

Figure 7.7: A visualization of the impact of the size of F. *NEnv-XL* is a larger version of F, with twice as many layers, bins and neurons per layer. This yields on small improvements on render and reconstruction quality, at the cost of sampling efficiency.



7.8.2 Environment Map Reconstruction

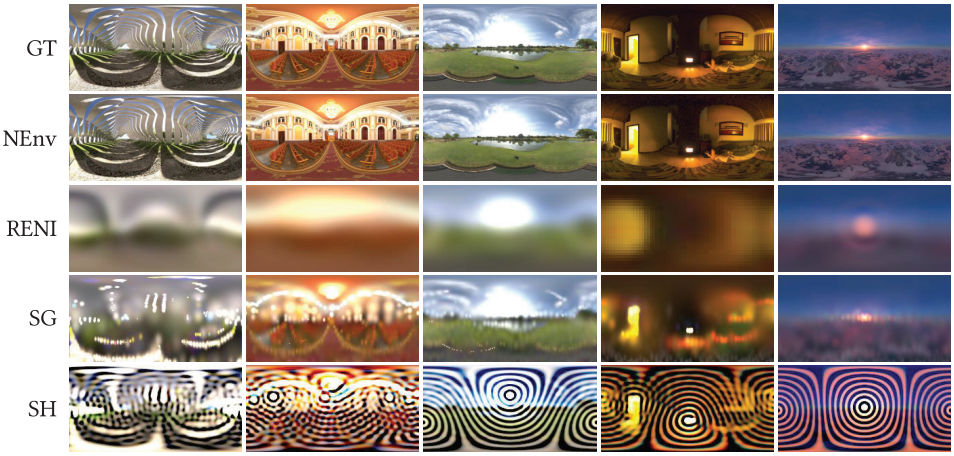
We now aim to measure the accuracy of the environment maps encoded by $\boldsymbol{C}$ and previous work on this problem. In Table 7.1, we show the average error on a variety of metrics for every environment map in our dataset. We use pixel-wise (PSNR) and perceptual metrics (SSIM [Wan+04], LPIPS-VGG [Zha+18], FLIP [And+20]) to thoroughly understand the dif-ferences between each method. We compute these metrics using *PIQ* [KZP19] and the implementation in [And+20], on images of (512, 256) pixels. We apply a γ = 2.2 to tonemap them before computing the distances. As shown, our network $\boldsymbol{C}$ clearly outperforms every other method, with SH struggling significantly, while RENI and SG achieve comparably better performances. We also provide a qualitative comparison in Figure 7.8, for a subset of our dataset, where it can be seen that $\boldsymbol{C}$ provides high-quality

reconstructions. RENI is optimized for outdoor, natural lighting, and it struggles with indoor illumination, providing overly smooth outputs. SG provides sharper results, while SH yields visually unpleasant results for every case: it is well known that the cyclic nature of SH tends to produce ringing artifacts, especially for high-contrast illumination environments and a large number of coefficients. We study the impact of these reconstructions on final renders in Section 7.8.4. Our representation c has the additional advantage that, thanks to its reconstruction quality, it can be used directly as background for samples that do not hit any geometry on the scene, while other alternatives require additional storage for higher quality versions, or even the original image.

Table 7.1: Average (± std.) environment map reconstruction error for different methods, with pixel-wise and perceptual metrics. We use a color code to highlight **best** and **worst** cases.

Method	SH [RH01b]	SG [Xu+13]	RENI [GES22]	**NEnv c (Ours)**
PSNR ↑	13.68±4.88	20.54±5.04	18.92±3.24	**30.65**±5.59
SSIM [Wan+04] ↑	0.276±0.20	0.559±0.16	0.557±0.16	**0.873**±0.08
LPIPS [Zha+18] ↓	0.687±0.07	0.637±0.12	0.657±0.12	**0.155**±0.08
FLIP [And+20] ↓	0.505±0.21	0.248±0.11	0.278±0.09	**0.062**±0.03

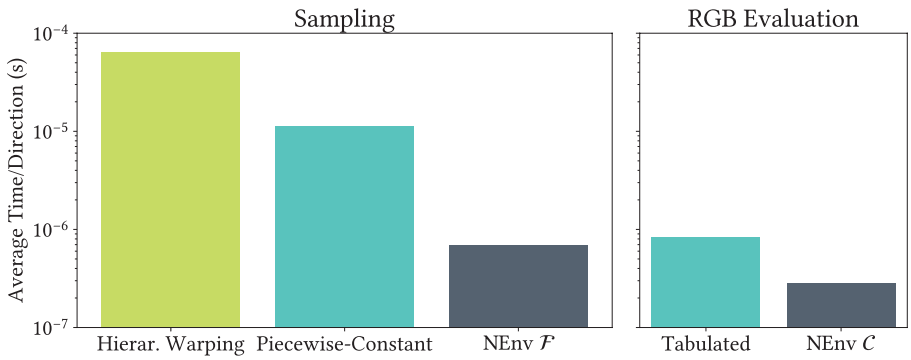
Figure 7.8: Environment maps encoded by our method and previous work, including RENI [GES22], Spherical Gaussian (SG) [Xu+13] and Spherical Harmonics (SH) [RH01b]. We use a $\gamma = 2.2$ to tonemap the RGB maps to help visualization.



7.8.3 Computational Cost

In Figure 7.9, we show the average time per sample obtained by the previous work and our normalizing flow F. On average, *Hierarchical Warping* [Cla+21; Pha19] obtains $6.48e^{-2} \pm 1.84e^{-3}$ milliseconds (ms) per sample, with the *PiecewiseśConstant* method [PJH16] achieving $1.12e^{-2} \pm 3.24e^{-4}$ ms, while F obtains $6.83e^{-4} \pm 3.69e^{-6}$. To make these comparisons more favorable to each method, the timings for *Hierarchical Warping* [Cla+21; Pha19] and *Piecewiseś Constant* [PJH20] are measured on CPU, since both algorithms rely on binary search which is difficult to parallelize on GPU, while the timings for NEnv are measured on GPU. For F, we use a sampling batch size of 200000, and we also include the times required to upload and download the samples from GPU, as in a real rendering scenario. We measure these times on the same hardware to make results comparable.

Figure 7.9: On the left, average seconds per sample for the baseline sampling algorithms and our model F. On the right, evaluation times for a tabulated baseline and our c. Note that we use a logarithmic scale for this plot.



While being differentiable and losing little reconstruction accuracy, F achieves, on average, 16.83 times more samples per second than the *Piecewise-Constant* method and 94.85 than *Hierarchical Warping*, with a maximum difference of 106.06. Furthermore, the timings for the *Piecewise-Constant* and *Hierarchical Warping* methods vary up to 15% and 12.5% respectively depending on the content of the environment map, while the variance of F is negligible. Not only did we achieve up to two orders of magnitude faster sampling than previous work, our method was also significantly more consistent in terms of timings. We found that *Hierarchical Warping* and the *Piecewise-Constant* methods typically struggle more with environment maps with multiple, distant light sources, while they are more efficient for natural daylight illumination.

In terms of RGB evaluation, a tabulated version of the environment map requires an average of $8.43e^{-4} \pm 1.44e^{-5}$ ms per direction, while our model $2.83e^{-4} \pm 7.85e^{-6}$ ms. Note that we use *TorchScript* with *Pytorch 1.11* to measure the times for $\boldsymbol{C}$ and $\boldsymbol{F}$. Newer versions of these libraries, which allow for compiled models, will likely help achieve faster evaluations without changing the model architectures.

7.8.4 Rendering Comparisons

The goal of NEnv is to generate accurate yet fast environment map representations that do not make compromises in terms of rendering quality. We use the *Piecewise-Constant* PDFs as our ground truth (GT) and perform qualitative and quantitative analyses to show the reconstruction quality of NEnv with respect to previous work. We provide a fine grained analysis of NEnv, in which we evaluate each of the models separately and a full version of the model which uses both $\boldsymbol{F}$ and $\boldsymbol{C}$. We use our path tracing engine configured as explained in Section 7.6 to render the same scene for each environment map. Each scene is rendered using a simple pinhole camera (no Depth of Field) and the environment maps as the only source of light, so we can reduce as much as possible external sources of Monte Carlo noise at low sampling. We use MIS, so both sampling and PDF functions for our maps impact the resulting image. All images are rendered with a resolution of (1080, 1920) pixels, 32 samples, maximum depth of 16, and path throughput weight Russian Roulette [PJH16].

In Figure 7.10, we show a qualitative comparison between different methods on the same scene, illuminated with different environment maps. We use an object from [Sch+17], which helps in highlighting the differences between far field illumination representations. As shown, our method, in any of their configurations, generates images that closely match the ground truth, while previous work struggles significantly, particularly in indoor illumination. Finally, in Table 7.2, we show a quantitative comparison of the average error obtained by these methods, for our entire dataset. For this analysis, we only measure the error on the object and floor on the renders in Figure 7.10, masking out the background. We observe that, even when combining our sampling and compression networks, our reconstruction quality is much higher than any of the methods in the previous work. Interestingly, the differences in environment map reconstruction quality we measured in Table 7.1, which were very unfavorable to SH, are not exactly correlated to rendering error, suggesting that SH, while being a pixel-wise inaccurate image representation, it tends to be precise on the more relevant information of the image, *i.e.*, where the key light sources lie. In terms of rendering, neither of the previous methods that we tested behaved consistently better than any other across every metric.

7.9 Conclusions

In this work, we have presented a neural rendering method for joint compression, evaluation, and sampling of environment maps for global illumination. We have validated our models quantitatively using a diverse dataset of environment maps. Our proposed lightweight normalizing flows provide accurate yet efficient sampling and pdf evaluation, achieving significantly faster computational times than analytical approaches with negligible loss in quality. Further, our sinusoidal compression networks achieve high-quality environment map approximations, surpassing previous work in both quality and generality. Every component in our models is differentiable, and they can be seamlessly integrated into engines like Mitsuba [Nim+19; Jak+22] or PRDPT [FR22] for inverse rendering applications. All in all, NEnv can represent global illumination with higher generality, quality and computational efficiency than previous work, with fully differentiable components.

Table 7.2: Average (± std.) render reconstruction error for different methods, with pixel-wise and perceptual metrics. We use a color code to highlight **best** and **worst** cases.

	PSNR ↑	SSIM [Wan+04] ↑	LPIPS [Zha+18] ↓	FLIP [And+20] ↓
SH [RH01b]	21.77±5.35	**0.933**±0.05	**0.160**±0.03	0.150±0.06
SG [Xu+13]	21.62±6.34	0.940±0.05	0.153±0.03	**0.154**±0.07
RENI [GES22]	**21.53**±5.62	0.936±0.04	0.159±0.03	0.151±0.05
NEnv (F)	37.97±3.73	0.996±0.00	0.024±0.01	**0.032**±0.02
NEnv (c)	**41.05**±8.89	**0.997**±0.01	**0.016**±0.01	0.036±0.02
NEnv (Both)	35.16±4.97	0.994±0.01	0.033±0.01	0.049±0.02

Our work could be extended in several ways. We require two networks, one for sampling and pdf evaluation, and another for compression. While this dual approach allows us to maximize their individual efficiency, a future research avenue is to build a single model which can solve the three tasks at the same time, to simplify our representation and possibly help with parameter reuse across models. However, fusing a sinusoidal MLP with a spline-based normalizing flow is not straightforward and requires a significant research effort that lies outside of the scope of this work. Recent work on architectural design [Meh+21] may provide cues on how this could be achieved. Further, we require to train new models for each input environment map. Building upon

recent work [GES22; Rai+22; GMX22; Szt+21; Hu+20], we could build a prior over environment maps, which should help reduce training times and solve inverse illumination problems, as in [Li+20; Azi+19; HHM22; YS19; Wan+21; YS21; Sri+20; Gar+19; Wan+23]. Achieving this without losing computational efficiency is a challenging future research problem. Besides, recent work on neural rendering [Mül+22; Che+22; Wan+22d; Att+22; Liu+22a] has shown that with carefully designed CUDA kernels or architectural rep-resentations, it is possible to achieve more efficient training and evaluation times. Integrating these ideas into our framework could help further increase its computational efficiency.

Finally, we believe that the ideas we present in this work can be used for other path tracing and physically based rendering problems, like aggregate scattering [Blu+16] or BxDF representations [Szt+21; CNN22; Zha+21; Che+20a]. We hope our work inspires future work on generative models for physically-based and neural rendering.

Figure 7.10: Render comparisons of different configurations of our work with the ground truth environment map and previous work. Best viewed in color on a screen. Please zoom in for details.