# Real-Time Neural Appearance Models

TIZIAN ZELTNER*, NVIDIA, Switzerland
FABRICE ROUSSELLE*, NVIDIA, Switzerland
ANDREA WEIDLICH*, NVIDIA, Canada
PETRIK CLARBERG*, NVIDIA, Sweden
JAN NOVÁK*, NVIDIA, Czech Republic
BENEDIKT BITTERLI*, NVIDIA, USA
ALEX EVANS, NVIDIA, United Kingdom
TOMÁŠ DAVIDOVIČ, NVIDIA, Czech Republic
SIMON KALLWEIT, NVIDIA, Switzerland
AARON LEFOHN, NVIDIA, USA

Fig. 1. Close-up renderings of a Teapot asset with our neural BRDF. Our model learns the intricate details and complex multi-layered material behavior of the ceramic, fingerprints, smudges, and dust which are responsible for the realism of the object while being faster to evaluate than traditional non-neural models of similar complexity. The system we present allows us to include such high-fidelity objects in real-time renderers in a scalable way.

We present a complete system for real-time rendering of scenes with complex appearance previously reserved for offline use. This is achieved with a combination of algorithmic and system level innovations.

Our appearance model utilizes learned hierarchical textures that are interpreted using neural decoders, which produce reflectance values and importance-sampled directions. To best utilize the modeling capacity of the decoders, we equip the decoders with two graphics priors. The first prior—transformation of directions into learned shading frames—facilitates accurate reconstruction of mesoscale effects. The second prior—a microfacet sampling distribution—allows the neural decoder to perform importance sampling efficiently. The resulting appearance model supports anisotropic sampling and level-of-detail rendering, and allows baking deeply layered material graphs into a compact unified neural representation.

By exposing hardware accelerated tensor operations to ray tracing shaders, we show that it is possible to inline and execute the neural decoders efficiently inside a real-time path tracer. We analyze scalability with increasing number of neural materials and propose to improve performance using code optimized for coherent and divergent execution. Our neural material shaders can be over an order of magnitude faster than non-neural layered materials. This opens up the door for using film-quality visuals in real-time applications such as games and live previews.

CCS Concepts: • **Computing methodologies** → **Reflectance modeling**.

Additional Key Words and Phrases: appearance models, neural networks, real-time rendering

**ACM Reference Format:**
Tizian Zeltner, Fabrice Rousselle, Andrea Weidlich, Petrik Clarberg, Jan Novák, Benedikt Bitterli, Alex Evans, Tomáš Davidovič, Simon Kallweit, and Aaron Lefohn. 2024. Real-Time Neural Appearance Models. *ACM Trans. Graph.* 43, 3, Article 33 (June 2024), 17 pages. https://doi.org/10.1145/3659577

*Equal contribution. Order determined by a rock-paper-scissors tournament.

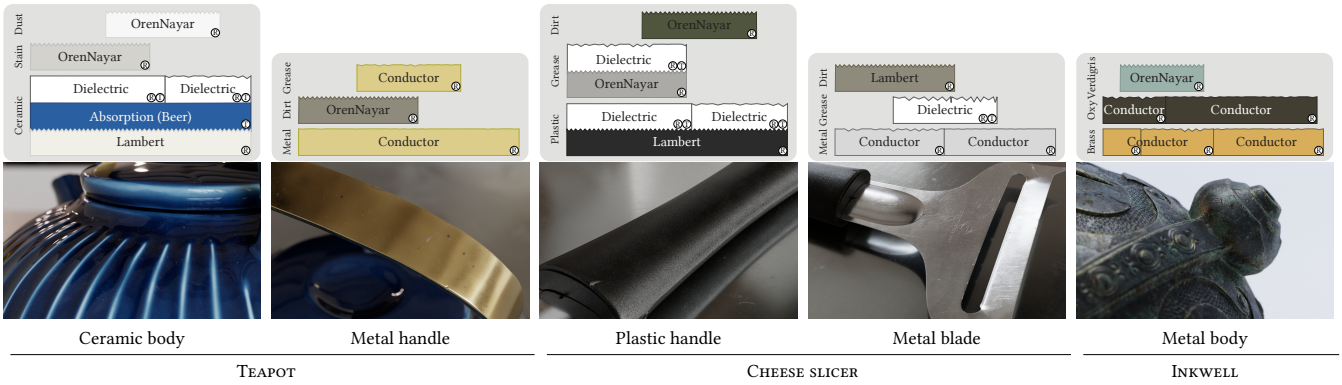



Fig. 2. We show rendered images of five reference materials created with a layering approach similar to [Jakob et al. 2019] that we approximate with neural models for representing the BRDF and importance sampling. All objects are challenging for real-time renderers due to their complex reflection behavior and high resolution textures (see Table 1). The corresponding shading graphs are provided in the supplementary material.

## 1 INTRODUCTION

Recent progress in rendering algorithms, light transport methods, and ray tracing hardware have pushed the limits of image quality that can be achieved in real time. However, progress in real-time material models has noticeably lagged behind. While deeply layered materials and sophisticated shading graphs are commonplace in offline rendering, such approaches are often far too costly to be used in real-time applications. Aside from computational cost, sophisticated materials pose additional challenges for importance sampling and filtering: highly detailed materials will alias severely under minification, and the complex multi-lobe reflectance of layered materials causes high variance if not sampled properly.

Recent work in neural appearance modelling [Kuznetsov et al. 2022; Sztrajman et al. 2021; Zheng et al. 2021] has shown that multilayer perceptrons (MLPs) can be an effective tool for appearance modelling, importance sampling, and filtering. Nevertheless, these models do not support film-quality appearance and a scalable solution for high-fidelity visuals in real time has yet to be demonstrated.

In this paper, we set our goal accordingly: to render film-quality materials, such as those used in the VFX industry exemplified in Figure 2 with statistics in Table 1, in real time. These materials prioritize realism and visual fidelity, relying on very high-resolution textures. Layering of reflectance components, rather than an ubershader, is used to generate material appearance yielding arbitrary BRDF combinations with tens of parameters. Approximating such materials with simple analytical models is inaccurate (see Figure 3) and porting to real-time applications is therefore challenging.

In order to render film-quality appearance in real time we i) carefully cherry-pick components from prior works, ii) introduce algorithmic innovations, and iii) develop a scalable solution for inlining neural networks in the innermost rendering loop, both for classical rasterization and path tracing. We choose to forgo editability in favor of performance, effectively "baking" the reference material into a neural texture interpreted by neural networks. Our model can thus be viewed as an optimized representation for fast rendering, which is baked (via optimization) after editing has taken place.

Our model consists of an *encoder* and two *decoders*, with the neural (latent) texture in between. The encoder maps BRDF parameters to a latent space, thereby converting a set of traditional textures (per-layer albedo, normal map, etc.) into a single multi-channel latent texture. Using the encoder is key to support materials with high-resolution textures. The latent texture is decoded using two networks: an evaluation network that infers the BRDF value for a given pair of directions, and a sampling network that maps random numbers to sampled (outgoing) directions.

Table 1. Statistics of our reference materials from Figure 2. The shading graph with shading nodes (a) is programmatically converted to a number of BRDF layers (b) controlled by parameters (c), which are varied spatially using RGB textures (with the total number of used channels in parenthesis) (d); the total number of RGB megatexels is reported in column (e).

| | Nodes (a) | Layers (b) | Parameters (c) | Textures (d) | MTexels (e) |
|---|---|---|---|---|---|
| Teapot ceramic | 37 | 5 | 121 | 5 (11) | 1174 |
| Teapot handle | 41 | 2 | 91 | 11 (19) | 152 |
| Slicer handle | 20 | 5 | 43 | 3 (7) | 201 |
| Slicer blade | 54 | 3 | 114 | 16 (40) | 324 |
| Inkwell | 49 | 5 | 143 | 4 (11) | 201 |

Our main algorithmic contributions can be characterized as embedding fixed-function elements—graphics priors—in the two neural decoders. First, we insert a standard rotation operation between trainable components of the BRDF decoder to handle normal mapped surfaces. Second, we utilize a network-driven microfacet distribution for importance sampling. These priors are necessary to efficiently utilize the (limited) expressive power of small networks.

On the system level, we present an efficient method for inlining fully fused neural networks in rendering code. To the best of our knowledge, this is the first complete and scalable system for running neural material shaders inside real-time shading languages. A key contribution is an execution model that utilizes tensor operations whenever possible and efficiently handles divergent code paths. This allows fast inferencing in any shader stage including ray tracing and fragment shaders, which is important for adoption in game engines and interactive applications.

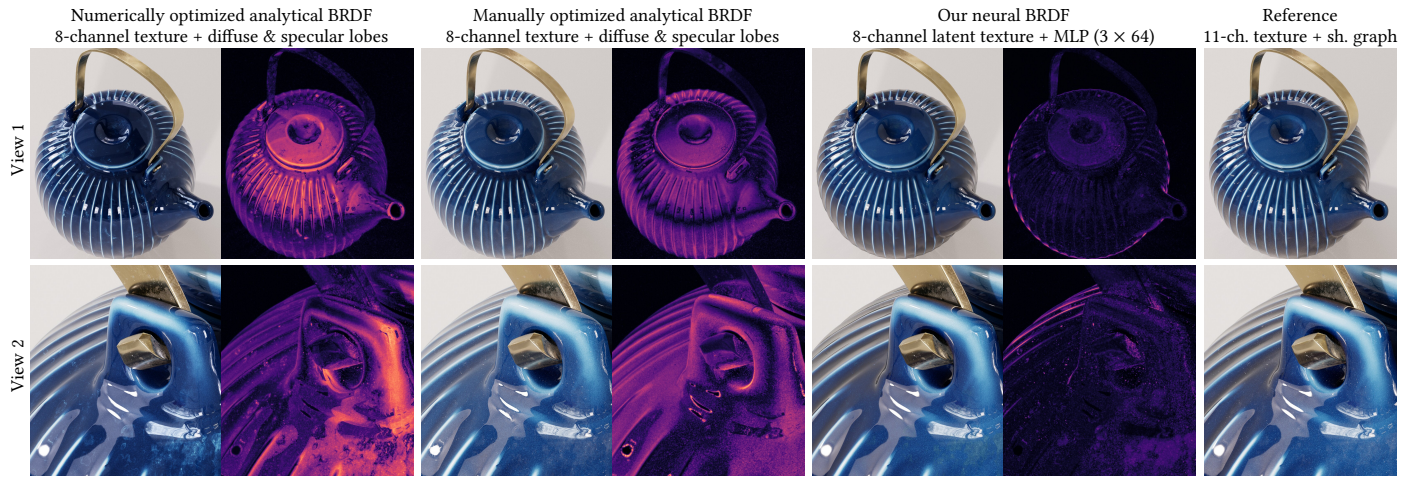



Fig. 3. First two columns: approximations of the multi-layer Teapot materials from Figure 2 using a simple analytical BRDF, parameterized by only 8 spatially-varying input channels: base color (3), specular roughness (1), specular normal map (2), specularity (1), and metallness (1). Third column: our neural BRDF parameterized by an 8-channel latent texture. ꟻLIP visualizations emphasize the perceptual differences against the reference (last column, Figure 2, Table 1). The parameters for the analytic BRDF are either numerically optimized or tuned manually. In both cases, we see a much larger approximation error as it lacks the expressive power to capture the complexity of the reference, e.g. the view-dependent blue color of the ceramic glazing.

Our neural model has a fixed evaluation cost, independent of the material complexity, allowing us to render complex materials in a real-time path tracer. To that end, we authored highly detailed assets with layered materials (Figure 2) that provide visual detail down to a 10 cm viewing distance. We can reproduce the visual fidelity of such complex assets, with shading being up to 10× faster than the original, moderately optimized shading models, while also providing additional sampling and filtering facilities (Figure 1).

Achieving the desired visual fidelity at real-time rates required innovations both in the neural model and at the system level:

- a complete and scalable system for film-quality neural materials,
- tractable training for gigatexel-sized assets using an encoder,
- decoders with priors for normal mapping and sampling, and
- efficient execution of neural networks in real-time shaders.

We believe the joint evolution of models and systems to be crucial to bringing neural shaders to real-time, and we built our system to serve as a solid foundation in this regard.

## 2 RELATED WORK

In this section, we review previous work related to neural material representation, filtering, and sampling, and refer to Pharr et al. [2016] for a detailed overview of classical material models.

### 2.1 Neural appearance modeling

We focus on representing existing materials neurally and rendering them in real time on classical geometry. We therefore do not utilize ray marched neural fields [Baatz et al. 2022; Mildenhall et al. 2020; Müller et al. 2022], although these could present a viable alternative in the future. Our goals generally align with prior work on neural BRDFs [Fan et al. 2022; Kuznetsov et al. 2019, 2021; Rainer et al. 2020, 2019; Sztrajman et al. 2021; Zheng et al. 2021]. Common to these methods is a conditioning of a neural network on a pair of directions, and optionally a trained latent code. Latent codes are typically stored in a texture [Thies et al. 2019] and sampled using classical UV mapping to support spatially varying BRDFs.

However, we differ from prior work on a number of key axes:

*Obtaining latent textures.* Kuznetsov et al. [2019] in their NeuMIP work employ *direct optimization*, updating a randomly-initialized latent texture via backpropagation—a simple but costly solution for large textures with millions of texels. In contrast, Rainer et al. [2019] rely on an auto-encoder architecture to *encode* a set of reflectance measurements into latent codes. We pursue a hybrid approach: we first train an encoder and, partway through training, we use it to create a hierarchical latent texture, which we then *finetune* through direct optimization. This approach combines the speed of the encoder-decoder architecture with the flexibility of direct optimization. Contrary to Rainer et al. [2019], we do not encode the reflectance measurements, but the set of corresponding material parameters (albedo, roughness, normal, etc.).

*Encodings and priors.* Both Zheng et al. [2021] and Sztrajman et al. [2021] reparametrize input directions into a half-angle coordinate system [Rusinkiewicz 1998]. While this specific encoding did not provide much benefit in our case, we leverage the principle and incorporate a novel graphics prior—rotation to learned shading frames—to better handle normal-mapped, layered materials.

*Rendering novel BRDFs.* Fan et al. [2022] are able to render novel BRDFs not part of the training set through layering of latents. However, this requires large neural networks unsuitable for real-time. We focus on small networks that render only materials they were trained on and do not pursue generalization. We support layered materials by capturing the joint effect of *all* layers at once, dispensing with the explicit layering of the original material, and avoiding any layering of neural components.

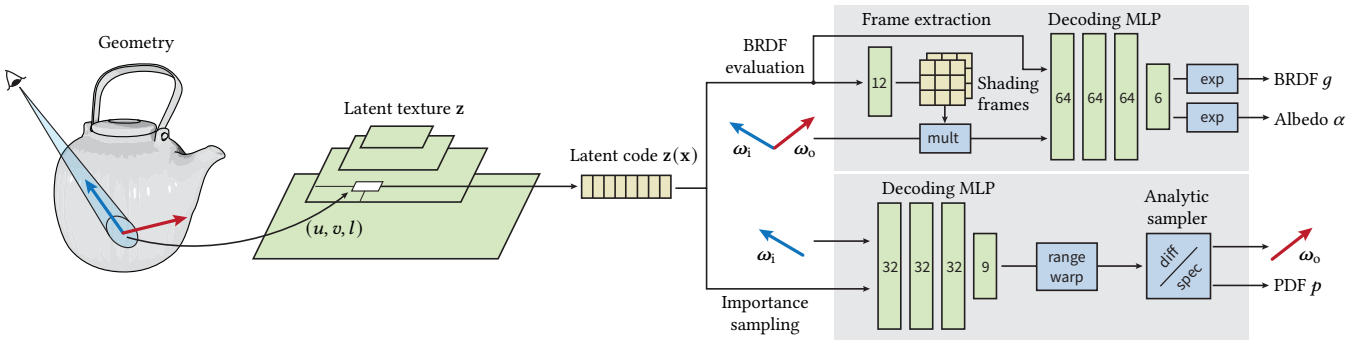



Fig. 4. We use our neural BRDFs in a renderer as follows: for each ray that hits a surface with a neural BRDF, we perform standard $(u, v)$ and MIP level $l$ computation, and query the latent texture of the neural material. Then we input the latent code $\mathbf{z}(\mathbf{x})$ into one or two neural decoders, depending on the needs of the rendering algorithm. The BRDF decoder (top box) first extracts two shading frames from $\mathbf{z}(\mathbf{x})$, transforms directions $\boldsymbol{\omega}_\mathrm{i}$ and $\boldsymbol{\omega}_\mathrm{o}$ into each of them, and passes the transformed directions and $\mathbf{z}(\mathbf{x})$ to an MLP that predicts the BRDF value (and optionally the directional albedo). The importance sampler (bottom box) extracts parameters of an analytical, two-lobe distribution, which is then sampled for an outgoing direction $\boldsymbol{\omega}_\mathrm{o}$, and/or evaluated for PDF $p(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})$.

## 2.2 Neural material filtering

Aliasing due to shading is commonly addressed with mipmapping, but requires special care for non-diffuse materials as their appearance can change significantly with linear filtering. Methods such as LEAN [Olano and Baker 2010], LEADR [Dupuy et al. 2013] and MIPNet [Gauthier et al. 2022] use statistical methods or neural downsampling to more closely match the prefiltered ground truth. While these approaches tune the parameters of traditional BRDFs, we instead train neural models and hierarchical textures to represent the filtered appearance directly, similarly to Kuznetsov et al. [2021] and Bako et al. [2023], albeit with a different interpolation scheme (see Section 4.1). However, we still leverage LEAN [Olano and Baker 2010] as a graphics prior to filter the inputs of our encoder.

## 2.3 Neural material importance sampling

Prior work on the importance sampling of neural materials can classified as: i) utilizing an analytical proxy distribution, ii) leveraging normalizing flows, and iii) warping samples with a network directly. See Xu et al. [2023] for an overview of neural materials samplers.

We utilize the first approach, in which a network parameterizes an analytical distribution. In contrast to Sztrajman et al. [2021] and Fan et al. [2022], who use the Phong-Blinn model or an isotropic Gaussian, we leverage a standard microfacet model [Trowbridge and Reitz 1975; Walter et al. 2007]. The microfacet model better handles anisotropy that is prevalent in (filtered) realistic materials.

Normalizing flows for importance sampling [Dinh et al. 2017; Müller et al. 2019] were first utilized for neural BRDFs by Zheng et al. [2021]. With sufficiently large networks, these can accurately match intricate distributions but we found it challenging to match the quality of the analytical proxy at comparable runtime performance.

The third approach, using the network directly to warp samples, has been recently explored by Bai et al. [2023] who aid training of the network with 2D optimal transport. This method has the drawback that the learned density only approximately matches the true Jacobian determinant of their warp. This leads to potentially unbounded bias, and we exclude this option to maintain compatibility with physically based renderers.

# 3 OVERVIEW

Our goal is to reproduce the appearance of real materials that stems from the interaction of light with matter. It can be described using the spatially varying bidirectional reflectance distribution function (SVBRDF) $f(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})$ that quantifies the amount of scattered differential radiance $\mathrm{d}L_\mathrm{o}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$ due to incident radiance $L_\mathrm{i}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i})$:

$$f(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) = \frac{\mathrm{d}L_\mathrm{o}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})}{L_\mathrm{i}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}) \cos\theta_i \mathrm{d}\boldsymbol{\omega}_\mathrm{i}}, \tag{1}$$

where $\mathbf{x}$ is a surface point, and $\boldsymbol{\omega}_\mathrm{i}$, $\boldsymbol{\omega}_\mathrm{o}$ are incident and outgoing directions, respectively. The SVBRDF can be integrated over the upper hemisphere $H^2$ to produce directional albedo $\alpha(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$:

$$\alpha(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o}) = \int_{H^2} f(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) \cos\theta_i \mathrm{d}\boldsymbol{\omega}_\mathrm{i}\,. \tag{2}$$

Our model represents both of these quantities; see Figure 4.

We design our model to serve as an optimized representation of existing (reference) SVBRDFs. That is, given a target material $f(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})$, we provide a function $g \approx f$ that closely approximates the reference material and can be evaluated in real time. To be useful, our system must satisfy a number of properties:

*Visual fidelity.* Our main goal is to faithfully reproduce a broad range of challenging materials, including multi-layer materials with low-roughness dielectric coatings, conductors with glints, stains, and anisotropy. We wish to go beyond fitting to spatially uniform measured material datasets [Dupuy and Jakob 2018; Matusik et al. 2003], and want to explicitly address materials with high resolution textures (4k and above) with detailed normal maps.

*Level of detail.* Unfiltered high-resolution materials tend to alias under minification and properly filtered reflectance can change significantly within a pixel footprint. We seek to support filtered lookups to enable level-of-detail rendering at low sample counts.

*Importance sampling.* In addition to representing the BRDF, we need an effective importance sampling strategy to permit deployment in Monte Carlo estimators, such as path tracing. This includes the traditionally challenging problem of importance sampling filtered versions of the material.

*Performance.* Our neural representation is geared towards real-time applications, where material evaluation may only use a small fraction of the total frame time. We require compatibility with path tracing, where materials are evaluated at random locations over many bounces. This precludes large networks and models relying on convolutions.

*Practicality.* While the optimization of our neural material happens in an offline process, training times have to remain reasonable even for high material resolutions (4k and beyond) for the system to remain practical. Days of training time are not acceptable.

Our main focus is on developing a system that fits the aforementioned criteria. Like prior works on neural materials, we forgo explicit constraints on energy conservation and reciprocity relying on the MLP learning these from data. We also set aside certain special cases, such as BRDFs with delta components, and (rough) refraction, although preliminary experiments show that our model can handle the latter.

In Sections 4 and 5, we describe the architecture of our neural model and its training procedure, following with a comparative analysis of individual components in Section 6. Since real-time performance is one of our main goals, we dedicate Section 7 to the task of efficiently evaluating the neural model from inside ray tracing shaders. We conclude by demonstrating the quality and runtime performance on a number of challenging scenes in Section 8.

## 4 NEURAL BRDF DECODER

In this section, we describe the architecture of our appearance model illustrated in Figure 4. The model consists of two main components: a *latent texture* and two *neural decoders*. All these components are jointly optimized to represent a specific material or a set of materials; details of the optimization procedure (e.g., encoding of the latent texture) follow in the next section.

The latent texture represents spatial variations of the material with a compact, eight-dimensional code denoted $\mathbf{z}$. Given a query location $\mathbf{x}$ and the corresponding latent code $\mathbf{z}(\mathbf{x})$, the BRDF value is inferred by a neural decoder $g$ with trainable parameters $\theta$:

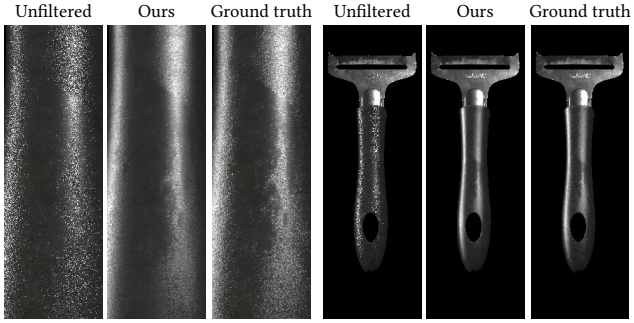
$$f(\mathbf{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) \approx g\left(\mathbf{z}(\mathbf{x}), T \cdot \boldsymbol{\omega}_\mathrm{i}, T \cdot \boldsymbol{\omega}; \theta\right) , \tag{3}$$

where $T$ represents a transformation of incident and outgoing directions to a number of learned shading frames. Next, we discuss the properties of the latent texture $\mathbf{z}$ and then describe the procedure of extracting $T$.

### 4.1 Latent texture

Similarly to prior works [Kuznetsov et al. 2021; Thies et al. 2019], we store latent codes in a UV-mapped, hierarchical texture, where each texel characterizes the appearance of the object at a given spatial location and scale. To maintain the fidelity of the original material, we set the resolution of the finest level to the texture resolution of the original material, and we leverage its UV-parametrization to preserve the original texel density.

Highly detailed materials may cause severe aliasing under minification (Figure 5, left columns in (a) and (b)). By default, our neural decoder would reproduce such aliasing. To avoid this, the hierarchical latent texture stores the latent codes in a texture pyramid [Kuznetsov et al. 2021; Thies et al. 2019]. Each level of the pyramid contains latent codes that characterize the original material filtered with a specific filter radius. The decoder is trained to infer the properly filtered BRDF value for all levels of the pyramid (Figure 5, middle columns in (a) and (b)).




(a) Cheese slicer, close (b) Cheese slicer, far

Fig. 5. Highly detailed materials will alias significantly when rendered without supersampling (left columns, unfiltered). Supersampling averages high frequency glints and produces a filtered material, but at impractical sample cost for real-time (right columns, ground truth at 512 SPP). Our neural material can render filtered materials without aliasing at any distance, without supersampling (middle columns, ours).

During rendering, we first determine the pixel footprint at the intersection point, and project it into UV space [Akenine-Möller et al. 2021]. We then determine the appropriate level of the texture pyramid to sample based on the area of the footprint.

The level index may be fractional and lie between two levels of the pyramid. We probabilistically select one of them using Russian roulette, and fetch the latent code via bilinear interpolation within the level. This introduces a small, but bounded amount of variance. We found this to yield higher quality than the more commonly used method of trilinearly interpolating the latent codes. This is likely because the latter strategy induces the additional constraint that the latent interpolation produce plausible BRDF values across levels, even though they may store very different content.

### 4.2 Transformation to learned shading frames

Our focus on real-time applications severely constrains the size of the decoder network. This makes it all the more important to incorporate graphics priors into the architecture to handle realistic materials, such as those exemplified in Figure 2. These layered materials produce intricate SVBRDFs, where reflection lobes shift in direction as we move over the surface. Such effects are readily modeled in classical materials via textured transformations, e.g., using normal maps, but are hard to achieve for a standard MLP.

A material may feature as many normal maps as scattering layers. We aim to compress the stack of layers, but still provide the model with enough room to represent multiple normal maps. We therefore incorporate a transformation module into the network, which transforms incident and outgoing directions into a number of learned shading frames (*mult* operation in Figure 4). Specifically, we use a single trainable layer to extract a fixed number $N$ of normals ($\mathbf{n}_1 \dots \mathbf{n}_N$) and tangent vectors ($\mathbf{t}_1 \dots \mathbf{t}_N$) from the latent code.

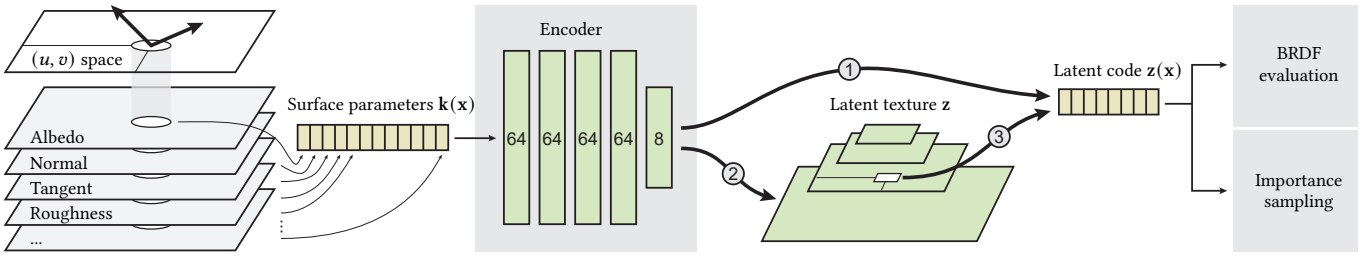



Fig. 6. We optimize our model by uniformly sampling the UV domain of the reference material. We start by fetching surface parameters (e.g., albedo) encoding them using an MLP to a latent code, and interpreting it as a BRDF value using the decoder (path marked with ①). Once the encoder is sufficiently trained, we construct the latent texture ② by processing all texels, and then drop the encoder. We continue "finetuning" the latent texture by sampling the UV space and MIP levels of the texture and optimizing the texels directly ③. We sample exponentially distributed filter footprints to optimize all levels of the latent texture, and train the decoder with prefiltered versions of the input material.

Then we construct a basis $(\mathbf{t}_i, \mathbf{b}_i, \mathbf{n}_i)$ for each $i$-th pair of normalized normals and tangents, and construct a combined transformation matrix $T$:

$$T = \begin{pmatrix} t_{1,x} & b_{1,x} & n_{1,x} & \dots & t_{N,x} & b_{N,x} & n_{N,x} \\ t_{1,y} & b_{1,y} & n_{1,y} & \dots & t_{N,y} & b_{N,y} & n_{N,y} \\ t_{1,z} & b_{1,z} & n_{1,z} & \dots & t_{N,z} & b_{N,z} & n_{N,z} \end{pmatrix}^{\mathsf{T}}. \tag{4}$$

The transformation layer then computes the product $T \cdot \boldsymbol{\omega}_i$ and $T \cdot \boldsymbol{\omega}_o$, resulting in $N$ new incident and outgoing vectors, one pair for each of the learned shading frames. The vectors are then fed to the decoder. The transformation allows the model to rotate the input directions into multiple, spatially varying shading frames in a single operation, improving the representational power of the network. We analyze the benefits in Section 6.

*Discussion.* It may not be immediately obvious why a vanilla MLP struggles with rotating directions. This is because, even though MLPs are built from matrix operations, they can only perform multiplicative transformations of the inputs with the (fixed) *network weights*. They cannot readily multiply the input dimensions with *each other*. In our case, a decoder with a vanilla MLP cannot easily multiply $\boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}$ with the latent code, which stores spatial variations of the material. The decoder is forced to approximate the multiplicative transform using its trainable layers, depleting its modeling capacity. Our approach is conceptually similar to (self-)attention models that augment neural networks with multiplicative transforms between activations [Rebain et al. 2023; Vaswani et al. 2017].

### 4.3 Importance sampling

We focus on samplers suitable for representation by a network: an invertible transform $W$ from random variates $\mathbf{u} \in [0,1)^2$ into outgoing directions $\boldsymbol{\omega}_\mathrm{o} = W(\mathbf{u}; \mathbf{x}, \boldsymbol{\omega}_\mathrm{i})$, and its associated probability density function (PDF) $p(\boldsymbol{\omega}_\mathrm{o}; \mathbf{x}, \boldsymbol{\omega}_\mathrm{i})$. Low variance results are achieved whenever the shape of $p$ closely matches $f$.

Optimizing an MLP to perform the sample transform $W$ does not guarantee invertibility of $W$ and tractable PDF evaluations. Importance sampling thus requires a different approach than BRDF evaluation. We draw inspiration from prior work and utilize a neural network to drive an existing analytic proxy distribution that is invertible in closed form. Like Sztrajman et al. [2021] and Fan et al. [2022], we use a linear blend between a cosine-weighted hemispherical density and a specular reflection component, but we differ in the choice of the specular component.

Instead of the isotropic models proposed earlier (e.g., Blinn-Phong model [Sztrajman et al. 2021] or a 2D Gaussian [Fan et al. 2022]) we use the more general, state-of-the-art microfacet model based on a Trowbridge-Reitz (GGX) NDF [Trowbridge and Reitz 1975; Walter et al. 2007] including elliptical anisotropy and non-centered mean surface slopes [Dupuy 2015]. This is well-suited both to the strongly normal-mapped materials represented in our target materials, as well as filtered BRDFs that naturally produce anisotropic distributions; we demonstrate the advantage in Section 6 and provide additional details of the sampler in Appendix A.

We train an additional *importance sampling decoder* MLP that infers parameters of the analytic model from the same latent code as used for the BRDF evaluation. This is conceptually similar to Sztrajman et al. [2021], though we additionally feed $\boldsymbol{\omega}_\mathrm{i}$ into the decoder to capture Fresnel-like effects where, e.g., the diffuse-specular mixing weights vary as a function of the incident angle.

## 5 TRAINING

We now discuss the training procedure for our decoder and latent texture (see Figure 6), and how our training data is generated.

One major challenge in training detailed materials is the sheer number of parameters to be optimized. Although the number of network weights is small, the resolution of the latent texture matches that of the source material and can be considerable: the ceramic body of the Teapot (Figure 2) is defined using 14 4k × 4k texture tiles totaling 235 million texels, or 2.5 billion latent parameters. Optimizing these parameters independently using backpropagation is impractical. Instead, we make use of an *encoder* in the first training phase to bootstrap latent codes, which we describe next.

### 5.1 Encoder

The encoder is a simple MLP that takes the parameters $\mathbf{k}(\mathbf{x})$ of the original material (albedo, roughness, normal maps, etc. for all material layers) at a given query location $\mathbf{x}$ as input, and outputs the corresponding latent vector $\mathbf{z}(\mathbf{x})$. To bootstrap the filtering, we prefilter the material parameters $\mathbf{k}(\mathbf{x})$ (using LEAN [Olano and Baker 2010]) for coarse MIP levels of the hierarchy.

In the first training phase, the model is trained end-to-end by forwarding the latent code from the encoder directly to the decoder, bypassing the latent texture.

After the decoder converges, we switch to the finetuning phase. The latent texture is initialized by evaluating the encoder for all texels, after which the encoder is dropped. The contents of the latent texture are then trained directly using backpropagation through the decoder. Because the encoder only participates in training, it has no impact on the evaluation cost during rendering.

The encoder also improves the structure of the latent space: it guarantees that similar material parameters are mapped to similar points in the latent space. This leads to better results under interpolation, and makes the job of the decoder easier. In contrast, direct optimization is prone to leaving portion of the random initialization noise in the latent texture, as analyzed in Section 6.2.

The encoder can be optimized to encode multiple materials, or even the full appearance space spanned by the reference BRDF (by sampling its parameters uniformly). Since our latent textures have a large memory footprint, in practice we train each one individually along with its own encoder, unless stated otherwise.

## 5.2 Data generation and optimization

We generate training data by uniformly sampling the UV space of the target (multi-layered) material. For each sample, we generate random directions $\boldsymbol{\omega}_\mathrm{i}$ and $\boldsymbol{\omega}_\mathrm{o}$ by uniformly sampling their half and difference vectors [Rusinkiewicz 1998; Sztrajman et al. 2021], and evaluate the reference BRDF value. Each sample additionally contains: normal, tangent, albedo, roughness, and layer weight, exported for each of the layers. Depending on the layer count a single sample may require over a hundred floating point numbers. We generate the samples on the GPU online during training.

*Filtering.* We discretely sample a pyramid level for each training sample from an exponential distribution, favoring finer levels. We average multiple sample points drawn from a Gaussian with appropriate footprint for the level, and choose the number of samples proportional to the filter area. This sampling process is fast enough that it does not significantly impact training time.

*Mollification.* Materials with very narrow peaks (e.g. the smooth glaze of the Teapot) lead to large training errors early in training and are challenging to learn for the network. To solve this, we initially blur the material directionally by averaging multiple samples from a small cone centered on $\boldsymbol{\omega}_\mathrm{o}$. The angle of the cone decreases during training, so that the network initially learns broad features of the material before converging to the reference.

*Optimization.* We train the BRDF decoder and the importance sampler simultaneously to establish a shared latent space. The BRDF prediction is optimized using the $L_1$ loss in log space [Zheng et al. 2021]. The PDF of samples $\boldsymbol{\omega}_\mathrm{o}$ drawn from the learned sampler is scored using the KL divergence against the current state of the learned BRDF. We found that training stability is improved when the latent code is detached from the KL loss computation. This way, the sampler MLP learns how to interpret the latents without interfering with the main BRDF evaluation decoder.

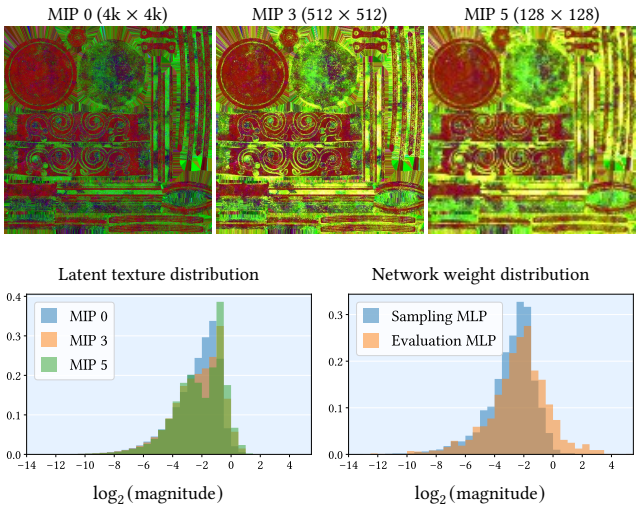



Fig. 7. Top row: Optimized latent textures (3 channels shown as RGB) for the neural Inkwell material at three levels of the MIP hierarchy. Bottom row: The corresponding distribution of latent (left) and network parameter magnitudes (right). All parameters lie comfortably within the $(2^{-14}, 2^{16})$ numerical range of FP16 normal numbers (excluding denorms), making quantization easy. The other materials show very similar distributions.

Albedo predictions, if enabled, are optimized using the $L_2$ loss against one-sample MC estimates of Equation (2).

We optimize our models using 300k iterations, processing two batches of 65k training samples in each iteration; one for optimizing the BRDF decoder and one for the sampler. This amounts to nearly 40 billion (online-generated) material samples in total, with training times lasting around 4–5 hours per material on a single NVIDIA GeForce RTX 4090. Further details of the training procedure are provided in the supplemental document.

*Precision.* We train master parameters for the BRDF decoder and sampler in 32-bit floating-point (FP32) precision. It is possible to make careful use of mixed precision training to further improve training performance without losing accuracy, but due to the small sizes of our MLPs we did not explore this option. For efficient inferencing, we use post-training quantization to convert the parameters to half precision (FP16) at load time. Figure 7 shows a representative example of the distribution of parameters for the evaluation and sampling models. In all our example configurations, the numerical range of network parameters lie within the normalized range of FP16. In future work, we plan to explore quantization aware training to further reduce runtime precision to INT8 or lower.

# 6 MODEL ANALYSIS AND ABLATION

Now that we have introduced our appearance model and its training procedure, we will analyze the main technical novelties: i) the transformation into learned shading frames, ii) the anisotropic importance sampler, and iii) the use of the encoder. We also demonstrate the filtering capabilities and the option of inferring albedo.

A number of neural appearance models have been published in the past, addressing various aspects of appearance modeling, e.g.,

geometric level of detail [Kuznetsov et al. 2021, 2022], interpretability of the latent space [Zheng et al. 2021], or layering of neural components [Fan et al. 2022]. These are complementary to our system and could be incorporated in the future. In this work, we focus on accommodating film-quality visuals and efficient execution on modern GPUs (presented in Section 7).

Due to the difference in focus, it is hard to compare our work to previous approaches *directly*. Instead, we compare to two ablated variants of our model in Table 2 and Figure 8, and relate them to corresponding components in prior work.

*Vanilla MLP decoder with latent texture.* The basic variant utilizes only a hierarchical latent texture and a vanilla MLP decoder. As such, there is no explicit rotation to shading frames in the decoder, and the texels of the texture are optimized *directly* via backpropagation. This variant can be viewed as the decoder of Sztrajman et al. [2021] extended to handle spatial variations using a hierarchical neural texture [Thies et al. 2019]. The model and the training procedure is also conceptually close to the NeuMIP model [Kuznetsov et al. 2021], except that NeuMIP additionally features a UV-offsetting module for handling displaced surfaces. The results of this variant (Figure 8, first column) fail to correctly reproduce the spatial details of the reference material due to the vast number of latent texels that need to be optimized. We further analyze the scaling of latent-texture optimization with increasing resolution in Section 6.2.

*Latent texture encoder.* The second column in Figure 8 shows the benefits of adding the encoder (Section 5.1). The texture detail is reproduced more faithfully due to two main reasons. First, the encoder prevents situations where multiple texels with identical BRDF end up with different latent codes after optimization. Such surjective mapping of latents to BRDF values often occurs in the basic model (first column) depleting the modeling capacity of the decoder. Second, the encoder amortizes each training record over many latent texels instead of optimizing a single latent texel.

While the spatial variations are captured well in this particular example, the decoder is unable to additionally capture the narrow reflection lobe of the Teapot ceramic even though it was correctly captured by the vanilla MLP decoder. This suggests that the model has insufficient modelling capacity to accurately reproduce both the spatial variations and the high-frequency reflections. This can be alleviated by increasing the size of the decoder.

Our encoder-decoder architecture is reminiscent of the auto-encoder used by Rainer et al. [2019] for compressing BTFs, with the key distinction that we chose to encode the material parameters (albedo, roughness, normal, etc.) instead of encoding the reflectance measurements. This allows our system to further improve scaling to very high-resolution textures, since the encoder can exploit the redundancy in the material parameterization.

*Transformation to learned shading frames.* In the third column of Figure 8, we prepend the MLP decoder with the transformation of directions to two learned shading frames, which are extracted from the latent code using an extra trainable layer with 12 neurons. This constitutes our complete model. As discussed in Section 4.2, performing a multiplicative operation on the inputs explicitly spares the MLP from approximating it using its non-linear layers. The quality of the results improves, including effects that are not necessarily related to normal mapping. This suggests that modeling capacity retained by the explicit shading frame transformation is "invested" in better capturing the shape and spatial variations of the BRDF.

Table 2. Image error metrics averaged over the four images in Figure 8 for each of the three compared variants. Material-specific statistics are included in the supplemental material.

| | Vanilla MLP | with encoder | with frame transform |
|---|---|---|---|
| Mean ꟻLIP | 0.2390 | 0.1956 | 0.0815 |
| Mean abs. error | 0.0769 | 0.0652 | 0.0183 |
| Mean sqr. error | 0.0682 | 10.1933 | 0.0057 |
| Mean rel. abs. error | 0.2177 | 0.3439 | 0.0656 |
| Mean rel. sqr. error | 0.0798 | 265.4018 | 0.0090 |
| SMAPE | 0.2670 | 0.2397 | 0.0713 |

## 6.1 Filtering

We evaluate the quality of our filtering in Figure 9 by comparing individual levels of the latent pyramid to ground truth rendered with supersampling. Our filtered model is a good match up close, but loses small details from a medium distance. This is because latent optimization does not work as well for coarser levels as it does for level 0 and slightly overblurs the result. This may be compensated by biasing our level selection towards finer MIP levels, at the cost of some aliasing. From afar, all levels have a similar appearance.

## 6.2 Latent texture optimization

We further analyze the benefits of using the encoder in Figure 10, in which we compare the latent textures of different configurations at MIP level 0. We visualize latent textures obtained via direct optimization (top row) and using the encoder at small ($512 \times 512$, left) and large ($4k \times 4k$, right) resolutions. The bottom insets show a close-up of the learned texture and the rendered appearance of this area. While direct optimization and the encoder perform comparably at small resolutions (as used for instance in NeuMIP [Kuznetsov et al. 2021]), the difference becomes apparent at high resolutions. At resolution $4k \times 4k$, the directly optimized texels receive roughly $64\times$ fewer gradient updates than texels of the $512 \times 512$ latent texture. This results in the decoder having to map vastly different latent codes (due to random initialization) to the same BRDF value, hindering its performance. Much of the initialization noise is still visible in the converged model. On the other hand, the encoder provides a more data- and compute-efficient approach, yielding high-fidelity visuals. All models were trained using the same amount of training data. Despite being computationally less intense during training, the models with direct optimization nearly doubled the training times (up to 10 hours) due to their higher memory requirements.

## 6.3 Importance sampling

We compare the importance sampler described in Section 4.3 against a simplified variant resembling that from Sztrajman et al. [2021] and Fan et al. [2022]. This variant is trained to only produce two outputs: an isotropic roughness parameter and a relative weight

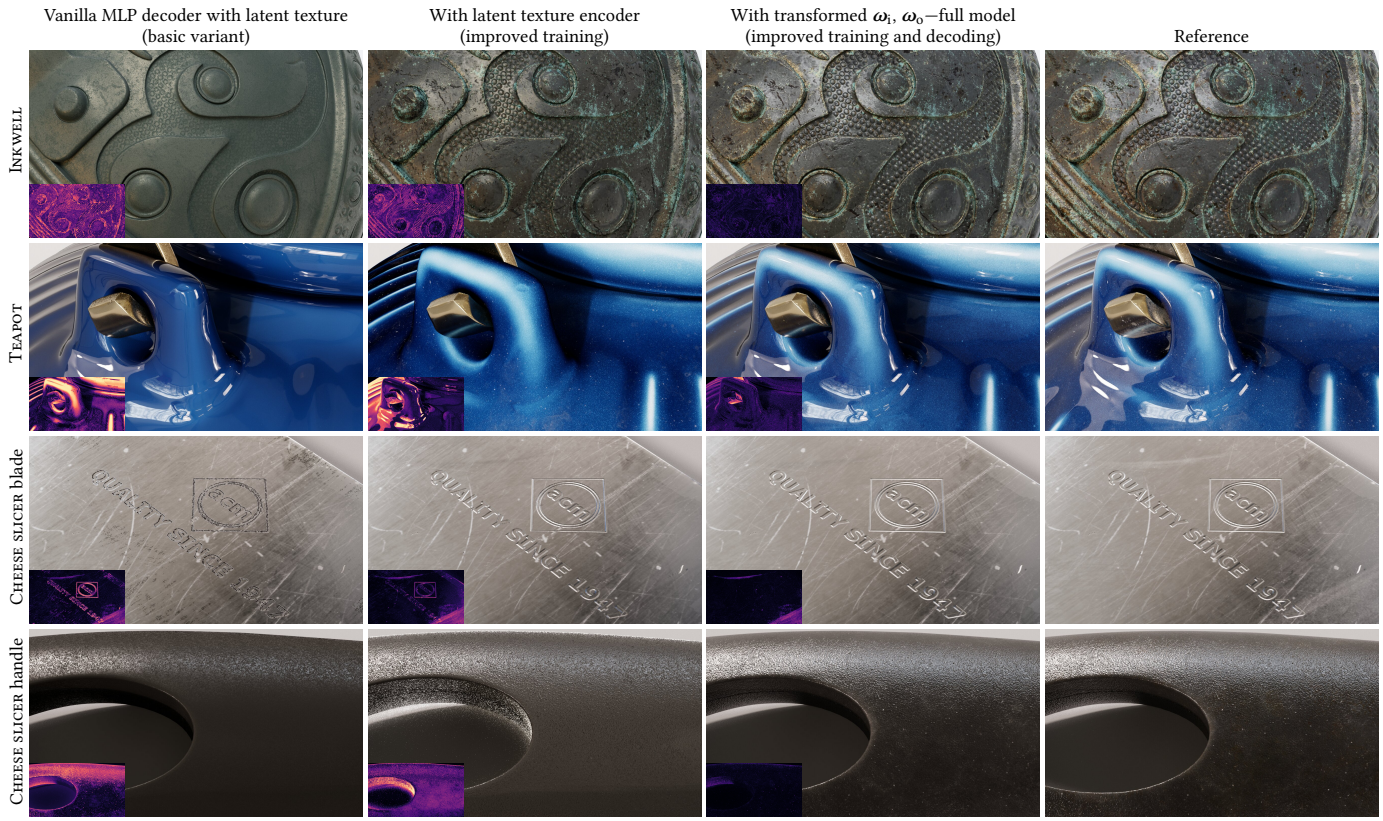



Fig. 8. A qualitative comparison of two ablated variants and our full model at equal amount of training iterations. A vanilla MLP decoder with directly optimized latent texture (first column) provides limited quality. Training an encoder to produce the latent texture (second column) ensures that texels with identical appearance feature identical latent codes, easing the decoding to BRDF values. Augmenting the MLP decoder with an explicit transformation of directions to learned shading frames—our full model (third column)—further improves the reproduction of the reference image (last column). The bottom left corners show images of the ꟻLIP difference metric. The models without the shading frame extractor (first two columns) were equipped with an extra first layer with 8 neurons to roughly match the number of parameters of the full model.

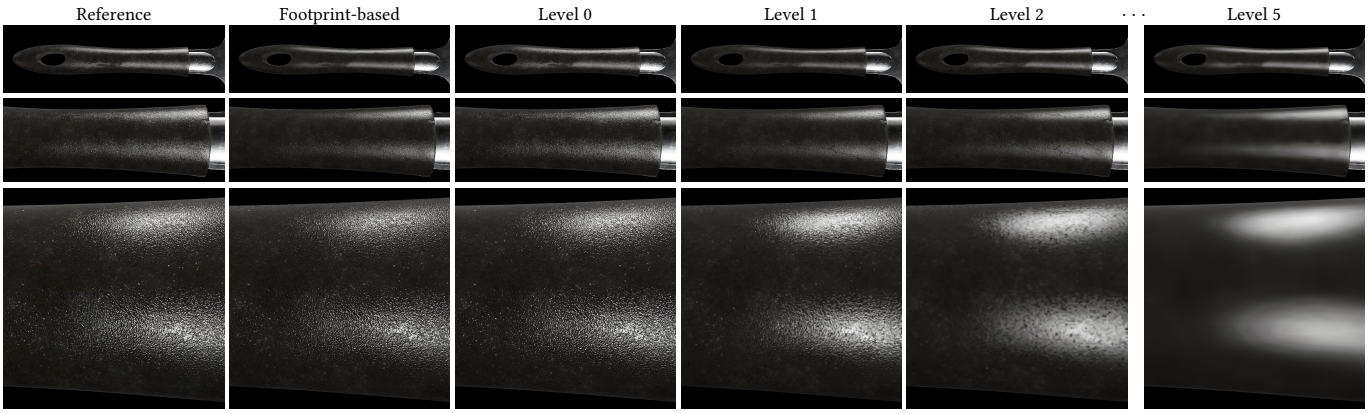



Fig. 9. We evaluate the quality of our filtering by comparing footprint-based level selection to fixed latent pyramid levels (rendered with supersampling) on the Cheese slicer asset at different distances. Up close, coarser levels show loss of small detail such as glints, which reflects in our filtered result. This is not the case for level 0, which is a near perfect match to the ground truth (at the cost of aliasing). From afar, all levels average to visually similar appearance.

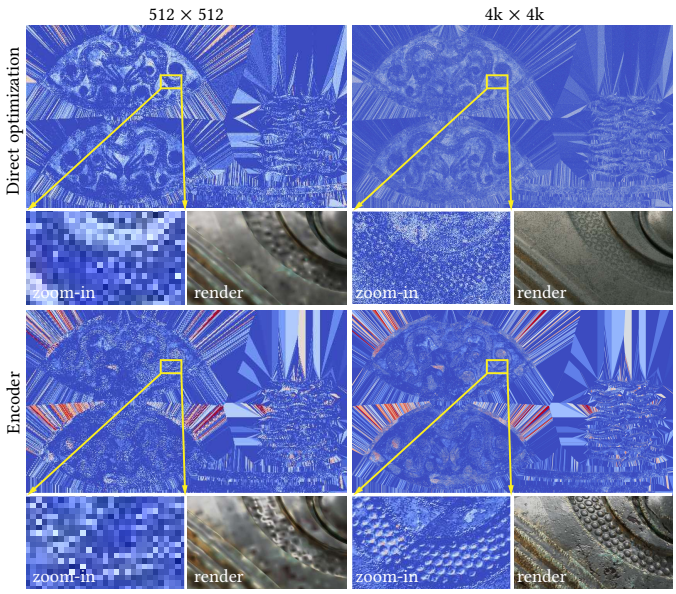



Fig. 10. Latent textures of the Inkwell asset. Direct optimization (top row) works well for small textures (left) but struggles with high resolutions (right) as independently optimizing texels is computationally inefficient; the latent texture still contains a large amount of initialization noise after many iterations. Therefore, we train an encoder (bottom row) that transforms PBR surface attributes into latent codes, and can be executed at any resolution. All analyzed configurations were optimized using the same amount of data. The left inset zooms-in on a small part of the texture that is partly visible in the rendered inset on the right.

for mixing the specular and diffuse components. Figure 11 shows the benefit of the more general approach in the context of level-of-detail rendering, where it is useful to sample both non-centered and anisotropic NDFs for normal mapped and filtered BRDFs.

We also considered using samplers based on normalizing flows [Dinh et al. 2017] in our system. In particular, the variant described by Zheng et al. [2021] where the distribution of half-vectors is represented by two piecewise quadratic warps [Müller et al. 2019], each parameterized by an MLP (3 layers with 16 neurons). We found this to yield comparable sampling quality to our chosen approach, but it increases the total frame render time by a factor of 2–3.8× (see Figure 12), making it less viable in our real-time context. This is explained by the additional overhead of the warps and the need to evaluate a larger number of MLPs at shading time. Normalizing flows generally run 4 MLPs at each hit: 2 when sampling an outgoing direction and 2 when evaluating the associated PDF, e.g. for computing multiple importance sampling (MIS) weights [Veach and Guibas 1995]. In contrast, our method only needs to query the sampling network once per hit and caches the resulting analytic proxy parameters for subsequent sampling and PDF evaluation steps.

### 6.4 Albedo inference

Figure 13 demonstrates the ability of a data-driven BRDF model to learn additional material characteristics. The BRDF decoder outputs an extra RGB triplet approximating the albedo of the multilayer material. We optimize the triplet against (one-sample) estimates of the true albedo during training using the $L_2$ loss, which ensures convergence towards the mean. The ability to predict albedo gives our approach an edge over complex materials composed of analytical models, that can only output texture values of *individual* components, since numerical albedo estimation is typically infeasible in a path tracer. The albedo value can be used, e.g., to guide a denoiser.

## 7 INLINE NEURAL MATERIALS

In this section, we describe the runtime system for inlining our neural appearance model in ray tracing shaders. Similar to recent work on real-time NeRFs [Müller et al. 2022], we implement fully fused neural networks from scratch on the GPU. Instead of hand-written kernels however, we use run-time code generation to evaluate the neural model *inline* with rendering code. This allows fine-grained execution of neural networks at every hit point in a ray tracing shader program, intermixed with hand-written code. There are several technical challenges in making this possible.

First, existing machine learning frameworks, such as PyTorch and TensorFlow, are built for coherent execution of neural networks in large batches. Tools for integrating neural networks in real-time shading languages such as GLSL or HLSL with potentially divergent execution, are largely non-existent. Second, we want to leverage hardware accelerated matrix multiply-accumulate (MMA) operations in recent GPU architectures by AMD,[1] Intel,[2] and NVIDIA,[3] but these instructions are not exposed in current shading languages. Last, the execution and data divergence in a renderer are challenging for neural networks, which load large amounts of parameter data from memory.

In the following, we discuss how we address each of these challenges in order to reach real-time performance.

### 7.1 Neural material shaders

Our neural model consists of several small MLPs, interconnected by blocks of non-neural operations. We train materials offline and export a description of the final model along with its learned hierarchical latent textures, stored as mipmapped 16-bit RGBA images. Texture compression of the latents is an interesting avenue for future work. In particular, neural texture compression [Vaidyanathan et al. 2023] may be very fruitful as the compression and neural material model could be trained end-to-end.

The runtime system compiles the neural material description into optimized shader code. We target the open source Slang shading language [He et al. 2018], which has backends for a variety of targets including Vulkan, Direct3D 12, and CUDA. Slang supports shader modules and interfaces for logically modularizing code. We generate one shader module per neural material, implementing the same interface as hand-written materials. In other words, neural materials are executed by the renderer no differently than classical ones. See the supplemental material for implementation details and pseudocode examples for functional reproducibility of our work.

[1] https://gpuopen.com/learn/wmma_on_rdna3
[2] https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-the-xe-hpg-architecture.html
[3] https://developer.nvidia.com/tensor-cores

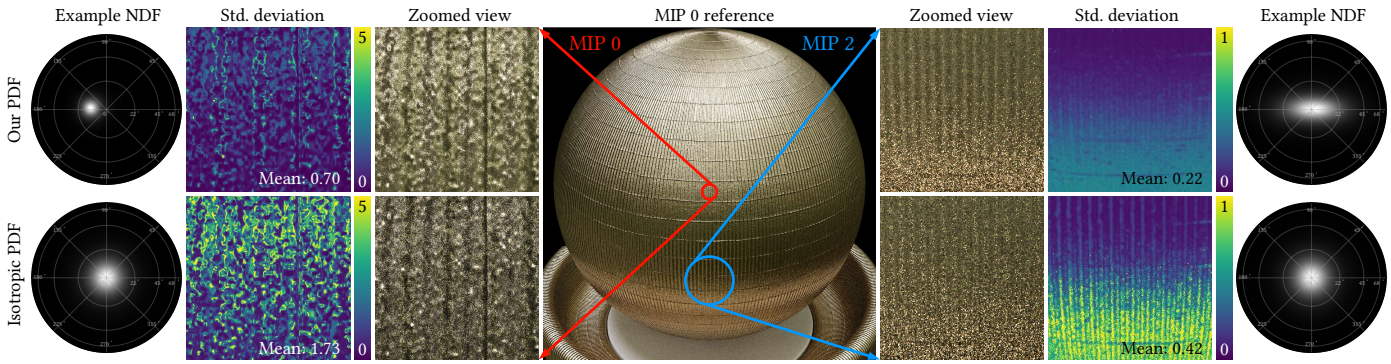



Fig. 11. The importance sampler (top row) reduces noise levels compared to a simpler variant only supporting isotropic specular reflections (bottom row), in the spirit of Sztrajman et al. [2021] and Fan et al. [2022]. Left: Fine details of a normal map are captured using a non-centered microfacet NDF. Right: At coarser MIP levels, the filtered distribution is strongly anisotropic. The zoomed views are rendered using 4 SPP. False-color images show the pixel-wise standard deviation and its mean across the entire inset.

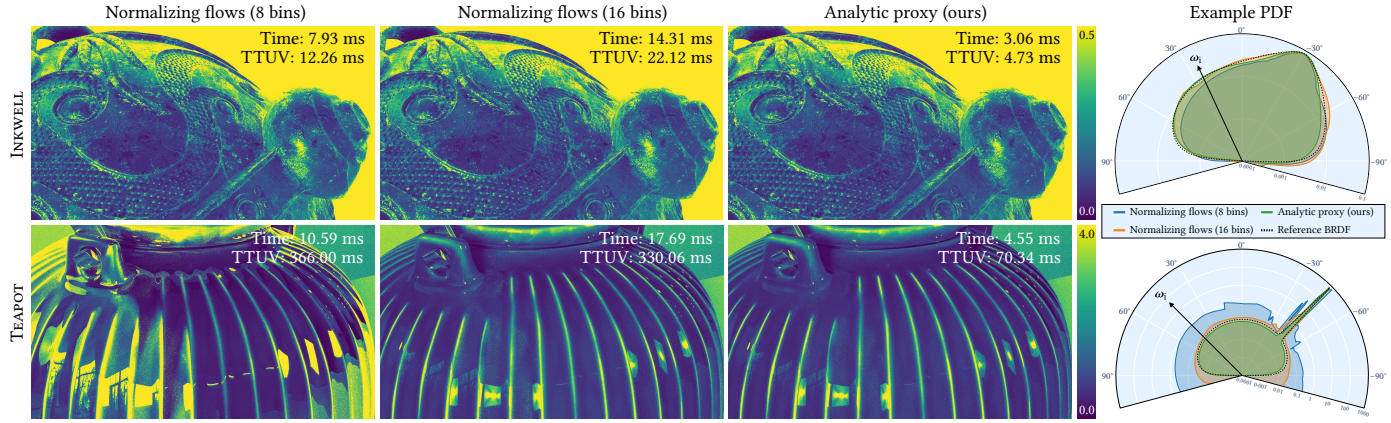



Fig. 12. Pixel-wise standard deviation images of our importance sampler against an alternative implementation based on normalizing flows. The sampler architecture in the first column (using warps with 8 bins, matching that of Zheng et al. [2021]), is adequate for the glossy Inkwell metal it struggles with the highly specular peak of the Teapot ceramic. The second column (using a higher-quality warp with 16 bins) captures the peak and roughly matches the variance of our sampler based on the analytic proxy (third column). The last column shows corresponding (log scale) polar plots of the learned densities. The overlaid numbers report rendering time (for the full frame at 1 SPP) and the *time to unit variance* (TTUV), i.e. the product of mean variance and render time. This reveals a significant runtime overhead of normalizing flows. The size of the evaluation network is fixed at 2 layers with 32 neurons in all cases.


Fig. 13. The BRDF decoder can be trained to additionally infer the albedo of the material by optimizing its additional RGB output against a Monte Carlo estimate of the albedo of the reference material.

*Code Generation.* GPUs use a *single instruction, multiple threads* (SIMT) execution model, where batches (*wavefronts* or *warps*) of threads execute in lockstep. Threads may be terminated or masked out due to control flow. Because each thread may process a different hit point and material, there is no guarantee that all threads in a warp evaluate the same network.

We handle this by generating two code paths, optimized for divergent and coherent execution respectively. The shader selects dynamically per warp which path to take. In the divergent case, we rely on the hardware SIMT model to handle divergence and generate an unrolled sequence of arithmetic and load instructions. A majority of the instructions evaluate the large matrix multiplies in the MLP feedforward layers. We use fused multiply-add (FMA) instructions to operate on two packed 16-bit weights at a time. The weights are laid out in memory in order of access, and special care is taken to generate 128-bit vectorized loads.

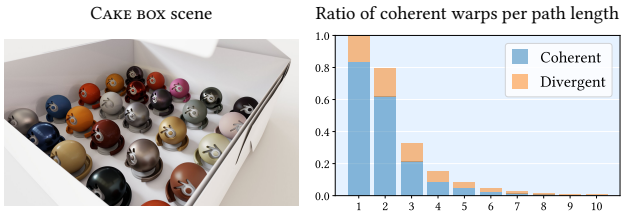



Fig. 14. This partially open Cake box is filled with 25 different neural materials. The statistics show that our megakernel path tracer achieves a high degree of shading coherency using shader execution reordering (SER) over all vertices along long light paths.

### 7.2 Tensor core acceleration

Some recent GPU architectures offer hardware units for accelerating general matrix multiplication. While implementation details vary, core functionality is similar. We focus on NVIDIA's *tensor cores* which provide many flavors of matrix multiply instructions, although the same idea applies to other architectures.

These instructions are currently limited to compute APIs and are not exposed in shaders. To address this, we modified an open source LLVM-based DirectX shader compiler[4] to add custom intrinsics for low-level access. This mechanism allows us to generate Slang shader code evaluating neural networks very efficiently using tensor cores, which operate on $16 \times 16$ blocks of the weight matrix simultaneously.

MMA instructions require cooperation across the warp, which limits this fast path to coherent warps where all threads evaluate the same material. Additionally, loading network parameters also benefits from coherent access, requiring careful consideration of how to construct coherent warps, which we discuss next.

### 7.3 Shading coherency

Neural materials allow us to reproduce a variety of materials using the same shader code, simply by swapping out network weights and latent textures. This improves warp utilization (and thus performance) even for workloads with traditionally high execution divergence, such as path tracing.

However, the increase in *data divergence* puts pressure on the memory system, and we can extract additional performance by increasing shading coherence. Classical coherent approaches like wavefront path tracing [Laine et al. 2013; van Antwerpen 2011] store hits to memory and globally reorder them after each bounce, but the high bandwidth requirements fundamentally limit their performance. Recent hardware features such as Intel's thread sorting unit (TSU)[5] and NVIDIA's shader execution reordering (SER),[6] instead reorder work *locally*. We use a megakernel path tracer to keep paths on-chip, and benefit from the increased data coherence provided by SER. Figure 14 shows that the majority of warps are fully coherent (shading the same material with all threads active) with our path tracing architecture.

[4]https://github.com/microsoft/DirectXShaderCompiler
[5]https://www.intel.com/content/www/us/en/developer/articles/guide/real-time-ray-tracing-in-games.html
[6]https://developer.nvidia.com/sites/default/files/akamai/gameworks/ser-whitepaper.pdf

## 8 RUNTIME ANALYSIS AND RESULTS

To study quality and performance, we implement our system for neural materials in a real-time path tracer [Clarberg et al. 2022a,b] built on the Falcor rendering framework [Kallweit et al. 2022]. The path tracer uses next-event estimation with MIS [Veach and Guibas 1995], and each path calls the *eval*, *sample*, and *evalPdf* material interface multiple times.

Our system is running on Direct3D 12 using hardware-accelerated ray tracing through DirectX Raytracing (DXR). All results are generated on an NVIDIA GeForce RTX 4090 GPU at resolution $1920 \times 1080$, unless otherwise noted. We focus on evaluating quality and performance for path tracing with neural materials, and therefore disable denoising and other features that can bias the results.

Performance is reported as total time in milliseconds (ms) for rendering a $1920 \times 1080$ image with *one* path sample per pixel (SPP). The timing in ms/SPP is representative for real-time path tracing, and can be scaled linearly to predict rendering time at higher SPP for applications such as high-quality preview rendering. Path length is capped at six path vertices (camera and light included) and Russian roulette is turned off for the purpose of these measurement.

*Reference materials.* In order to study rich materials, we added support for physically-based, layered material graphs expressed in the open standard MaterialX [Smythe and Stone 2021], a common interchange format for high-fidelity materials in VFX and movie production. This allows authoring complex layered materials (c.f., Figure 2) in Houdini and other tools. All materials consist of multiple BRDFs combined through mixing or coating operations. Nearly all parameters are textured, with resolutions of 4k-8k per texture. Some materials stitch multiple (up to 14) 4k texture tiles for even higher resolution. We programmatically converted the reference materials into an optimized Slang code that implements the shading graph as a weighted ($\omega_i$-dependent) combination of standard BRDF models. Each material comprises multiple layers, where each layer is driven by a number of textures; the statistics are provided in Table 1.

### 8.1 Visual accuracy

In Figure 3, we compare our proposed neural material parameterized by an 8-channel latent texture to a simple analytical model that combines a diffuse component with an isotropic Trowbridge-Reitz (GGX) lobe, which are driven by textures with 8 channels in total. We tested two variants for the analytical model: numerically optimized parameters obtained using our existing training pipeline (which was tuned for training neural materials), and parameters that were manually optimized by a specialist. Both variants fail to capture the complexity of the reference, multi-layered material. In particular, the diffuse albedo of the simple analytical model can only capture a slice of the view-dependent color of the ceramic glazing and is therefore accurate only for the specific view directions that match the chosen albedo. The neural material offers a more faithful reproduction, overall striking a balance between the speed and quality of the high-quality but slow reference, and the lower-quality but fast analytical approximation.

In Figures 15 and 16, we compare the visual quality and rendering performance of three configurations of the neural BRDF decoder (the importance sampler always comprises 3 hidden layers with

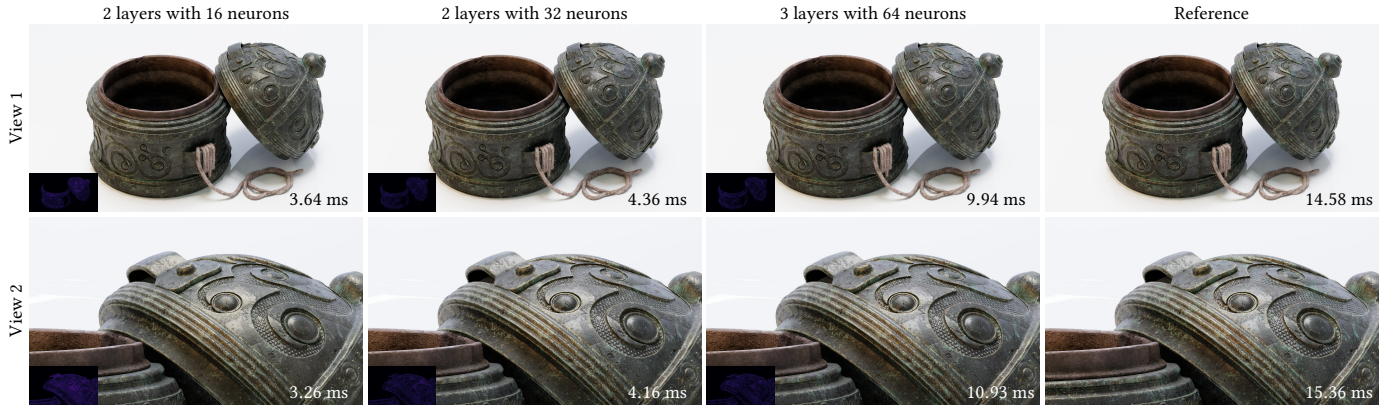



Fig. 15. The Inkwell scene where the metal uses the proposed neural BRDF. The remaining parts use analytical BRDFs. The first three columns show different sizes of the BRDF decoder, from fastest to the most accurate. In the corners we show a ꟻLIP error image and the rendering performance of an image with a *single path sample per pixel* (1 SPP) at 1920 × 1080 resolution using paths of up to length six. All images are rendered at 8192 SPP to suppress path tracing noise.

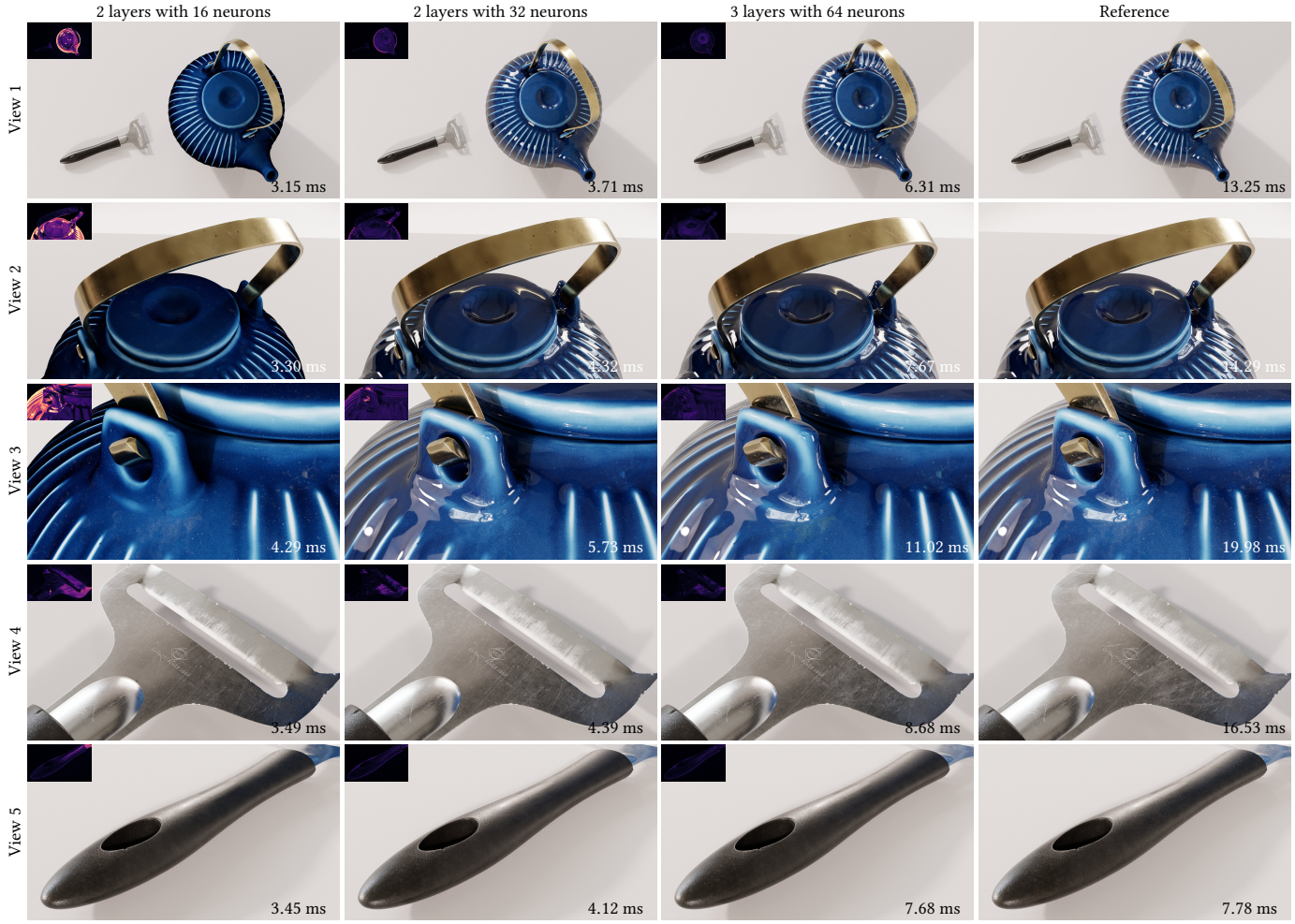



Fig. 16. The Stage scene with four materials that we approximate using the proposed neural BRDFs. We use a similar layout as in Figure 15. ꟻLIP error images are in the corners, timings quantify the cost of rendering a 1 SPP image of the scene at 1920×1080 resolution using paths of up to length six. All images are rendered at 8192 SPP to suppress path tracing noise. The rendering with neural BRDFs is **1.64**× to **4.14**× faster than the reference materials in full frame time (averaged over the views in Figure 15 and here). Please refer to the supplemental document for details on the scene and lighting setup.

Table 3. Image error metrics averaged over the converged renderings shown in Figures 15 and 16, each of which was produced using 8192 SPP. View-specific statistics are included in the supplemental material.

| | 2 × 16 | 2 × 32 | 3 × 64 |
|---|---|---|---|
| Mean ꟻLIP | 0.1087 | 0.0551 | 0.0444 |
| Mean abs. error | 0.0439 | 0.0145 | 0.0121 |
| Mean sqr. error | 1.3855 | 0.0107 | 0.0101 |
| Mean rel. abs. error | 0.1042 | 0.0429 | 0.0347 |
| Mean rel. sqr. error | 0.0353 | 0.0056 | 0.0035 |
| SMAPE | 0.1449 | 0.0468 | 0.0363 |

Table 4. Full frame performance in ms/SPP with three different BRDF decoder architectures (importance sampler is always 3 × 32). Column labels denote the number and width of hidden layers. Numbers in parenthesis show speed up over the reference material, reported in the last column.

| | 2 × 16 | 2 × 32 | 3 × 64 | Ref. |
|---|---|---|---|---|
| Inkwell, View 1 | 3.64 (4.01×) | 4.36 (3.34×) | 9.94 (1.47×) | 14.58 |
| Inkwell, View 2 | 3.26 (4.71×) | 4.16 (3.69×) | 10.93 (1.41×) | 15.36 |
| Stage, View 1 | 3.15 (4.21×) | 3.71 (3.57×) | 6.31 (2.10×) | 13.25 |
| Stage, View 2 | 3.30 (4.33×) | 4.32 (3.31×) | 7.67 (1.86×) | 14.29 |
| Stage, View 3 | 4.29 (4.66×) | 5.73 (3.49×) | 11.02 (1.81×) | 19.98 |
| Stage, View 4 | 3.49 (4.74×) | 4.39 (3.77×) | 8.68 (1.90×) | 16.53 |
| Stage, View 5 | 3.45 (2.26×) | 4.12 (1.89×) | 7.68 (1.01×) | 7.78 |
| Average | 3.51 (4.14×) | 4.40 (3.31×) | 8.89 (1.64×) | 14.54 |

32 neurons each). As expected, quality varies with the size of the decoder. The largest configuration, with 3 hidden layers and 64 neurons, reproduces the reference material well, with most details and colors captured accurately. The errors appear mostly at grazing angles of near-specular materials, e.g., the ceramic Teapot body near to the silhouette. We tested a number of hyper-parameter configurations, and while some successfully reduced the grazing angle artifacts (e.g., using $L_2$ loss), the quality elsewhere degraded, sometimes significantly. In order to escape this "zero-sum" game, we posit that another graphics prior is needed for handling Fresnel effects; we leave this to future work.

We include ꟻLIP [Andersson et al. 2020] false-color error images in corners to illustrate the perceived difference when toggling between the neural and reference BRDFs renders; all images are also provided as part of the supplemental material to facilitate such inspection. Table 3 lists average errors using a variety of standard image error metrics. The supplemental also includes polar plots for the learned materials with different decoder sizes.

## 8.2 Runtime performance

The smallest network yields the best rendering performance, albeit at reduced reconstruction accuracy. Table 4 lists the absolute performance in ms/SPP and the relative speed improvement over rendering a GPU-optimized implementation of the reference material (all running on NVIDIA GeForce RTX 4090 GPU). The full frame rendering times with the neural BRDFs are **1.64**× (3 × 64) to **4.14**× (2 × 16) faster than the reference material on average.

Table 5. Material shading performance in ms/SPP with two different BRDF decoder architectures (importance sampler is always 3 × 32). Column labels denote the number and width of hidden layers. Numbers in parenthesis show speed up over the reference material, reported in the last column.

| | 2 × 32 | 3 × 64 | Ref. |
|---|---|---|---|
| Stage, View 3 | 1.59 (10.19×) | 6.02 (2.69×) | 16.21 |
| Stage, View 4 | 1.23 (12.82×) | 5.06 (3.12×) | 15.77 |
| Inkwell, View 1 | 1.59 (6.99×) | 6.01 (1.85×) | 11.11 |
| Inkwell, View 2 | 1.74 (7.25×) | 7.15 (1.76×) | 12.61 |
| Average | 1.54 (9.06×) | 6.06 (2.30×) | 13.93 |

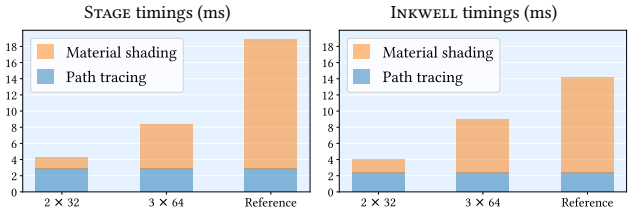



Fig. 17. Average path tracing and material shading time in ms, respectively, for rendering a 1 SPP image of the scene at 1920×1080 pixels resolution using paths up to six path vertices in length. Two different BRDF decoder architectures are profiled, and compared to the cost of shading using the reference materials.

The frame time includes both general path tracing operations (light sampling, ray tracing, and control logic) as well as material sampling and evaluation. To estimate how much time is spent in material shading, and thus the relative speedups of our neural materials over the reference materials, we setup a dedicated benchmark. Since all neural material shaders in our system are running inline in the renderer, not as separate kernels, this has to be done with care; we lock the path distribution to a simple cosine-weighted distribution, while ensuring that the compiler does not eliminate any of the material code. As a baseline, we measure the pure path tracing cost using a material with constant color.

Table 5 and Figure 17 summarize our findings for two representative views of the Inkwell scene (Figure 15, view 1 & 2) and Stage scene (Figure 16, view 3 & 4). The shading times with the neural BRDFs are **2.30**× (3 × 64) to **9.06**× (2 × 32) faster than the reference materials on average, with over an order of magnitude speedup for several views and the mid-sized BRDF decoder (2 × 32).

Overall, the performance and visual fidelity scale in a predictable manner as neural BRDFs accommodate trading quality for performance. Next, we analyze the scaling behavior in more detail.

## 8.3 Scalability

Figure 18 shows that performance scales favorably when increasing the number of neural materials. For this test we render the Cake box scene (Figure 14) and vary the number of (different) neural materials, while keeping geometry and path distribution identical. Paths up to ten vertices in length are traced and the scene also contains a small number of traditional materials, in order to introduce significant execution and data divergence.

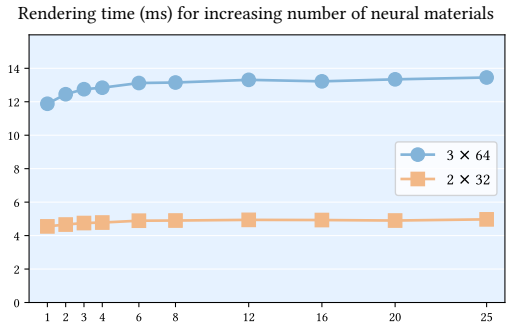



Fig. 18. Rendering times for path tracing a 1 SPP image of the Cake box scene with varying numbers of neural materials. The measurements show that our method is insensitive to the divergence introduced by path tracing scenes with many neural materials; rendering times stay near constant as material count increases. Two different BRDF decoder architectures are studied. The path distribution is kept fixed to isolate the effects on performance from scaling the number of materials.

For very small numbers of neural materials, the network parameters fit in caches close to the shader cores, whereas with more materials the parameters are increasingly streamed in from L2 or global memory. Our approach based on a megakernel path tracer with local work reordering manages to extract enough coherency to amortize the cost of memory loads well.

*Memory usage.* The memory footprint is dominated by the 8-channel, half-precision latent texture, requiring 256MB per 4k texture tile. The network weights are comparably small, requiring 37kB for the 3x64 network configuration and 9.3kB for the 2x16 configuration.

*Discussion.* It is difficult to do a direct comparison to previous work as our focus is different; we show that neural materials can run efficiently in real-time shaders even in divergent workloads such as path tracing. There are few examples of inferencing in traditional shaders. One exception is *deep shading* [Nalbach et al. 2017] that runs a forward pass in GLSL for traditional deferred shading. Research on neural appearance models have generally used CUDA kernels, either directly or via machine learning frameworks.

Fan et al. [2022] record all intersections to global memory and shade in a deferred manner, precluding adaptiveness and paying the cost of memory transfers. The authors report a single BRDF evaluation per pixel with resolution $1920 \times 1080$ costing 5 ms on an NVIDIA RTX 2080Ti. NeuMIP [Kuznetsov et al. 2021] implement an interactive CUDA/OptiX-based path tracer and report similar performance of 5 ms per evaluation at the same resolution/GPU. The paper is scarce on details; in personal communication it was stated that the reported 60 frames per second path tracing applies to relatively short paths in a simple scene with a single material. Scaling to multiple materials is not explored.

We believe the scalability, handling of divergent shaders, and integration in real-time shading languages are important contributions of our work for ease of adoption of neural materials more widely.

## 9 LIMITATIONS & FUTURE WORK

*Energy conservation and reciprocity.* Because the neural material is only an approximate fit of the input material, it is not guaranteed to be energy conserving. Although we have not observed this to be a problem in our tests, this could become an issue for high albedo materials with high orders of bounces (e.g. white fur). Enforcing energy conservation would require the network to output in a form that is analytically integrable, or integrates to a known value. The latter can be achieved with normalizing flows (as in [Müller et al. 2020]) at an increased evaluation cost. Our BRDF model is currently not reciprocal, but reciprocity could be enforced with the modified Rusinkiewicz encoding of directions [Zheng et al. 2021]. We opted for the Cartesian parameterization of directions that was more numerically stable in our experiments and yielded better visuals.

*Displacement.* We do not currently support effects that affect surface geometry, such as displacement mapping. We implemented the neural displacement approach of Kuznetsov et al. [2021], and tested several variations that include geometric priors, but we found that this approach is always outperformed by fixed-function ray marching, both in terms of bandwidth and runtime. None of these approaches were sufficiently fast to reach our performance goals, but we expect additional research to make them viable alternatives.

*Filtering.* Although neural prefiltering is effective at preventing aliasing, we report that, while the finest level is very accurate, the coarser levels of the latent pyramid tend to produce softer appearance than the supersampled reference BRDF. This is likely because the inputs to the encoder correlate strongly with the appearance *only* at the finest level. In case of coarser levels, the encoder consumes prefiltered material parameters, where the correlation is weaker and the auto-encoder thus performs worse. Finetuning improves the quality somewhat, but cannot escape the initial local minimum.

*Alternative geometric priors.* We tested a number of alternative implementations of the rotation prior (Section 4.2), ranging from unconstrained, high-dimensional affine transforms inspired by the generality of self-attention layers [Vaswani et al. 2017] to rotation-only matrices. Our final solution uses normalized (but not orthogonal) normal $\mathbf{n}$ and tangent $\mathbf{t}$ from the network output, with bitangent $\mathbf{b} = \mathbf{n} \times \mathbf{t}/\|\mathbf{n} \times \mathbf{t}\|$. Additionally, we tested explicitly supervising the extracted TBN frames against frames of the reference material, with an optional asymmetric loss [Vogels et al. 2018]. This occasionally improved the results (e.g., for glints), but the training requires extensive hyperparameter tuning; hence we excluded it from results.

*Training stability and time.* We occasionally found training to converge to local minima with large visual differences based on small perturbations of hyperparameters or weight initialization. For instance, the smallest network configuration could not reliably preserve the highly specular glazing of the Teapot so we chose to include a version without it in our results (Figure 16). We want to investigate robustness more closely, also while scaling to a larger target material diversity. At the same time, we would like to significantly reduce training times (ideally from *hours* to *minutes*) to improve iteration times when developing further enhancements and to make the current iteration of the system more practical.

*Refraction.* We evaluate our method only on purely reflective materials. Extending our model to transmissive materials poses the following challenge: physically based renderers require knowing the index of refraction of the material to maintain reciprocity after refracting. While the network could be trained to produce the index as an additional output, it is difficult to guarantee that this trained value matches the actual behavior of the BRDF; this topic deserves special attention in the future.

## 10 CONCLUSION

We present a complete real-time neural materials system. The model jointly addresses evaluation, sampling, and filtering of highly complex and detailed materials. We achieve this by combining ideas from prior works with new graphics priors and training strategies to achieve higher quality and faster training. A key contribution of our work is that such comprehensive solutions can be implemented efficiently on modern graphics hardware; we propose to deploy the neural network to the innermost rendering loop to reduce bandwidth requirements. In our tests, the neural BRDFs achieve state-of-the-art rendering performance, outperform optimized GPU implementations of reference multi-layered classical materials, and scale to multiple materials in a scene. We believe the presented neural BRDFs can serve as "baked" versions of complex materials; as well as increased performance and lower memory consumption, this enables easy interchange of arbitrarily complex materials between different workflows and tools, simply by exchanging a fixed set of latent textures and a small table of MLP weights. Lastly, we hope this article will stimulate adoption of small neural networks in real-time rendering.

## ACKNOWLEDGMENTS

We want to thank Toni Bratincevic, Davide Di Giannantonio Potente, and Kevin Margo for their help creating the reference objects, Yong He for evolving the Slang language to support this project, Craig Kolb for his help with the 3D asset importer, Justin Holewinski and Patrick Neill for low-level compiler and GPU driver support, and Karthik Vaidyanathan for providing the TensorCore support in Slang. We also thank Eugene d'Eon, Steve Marschner, Thomas Müller, Marco Salvi, and Bart Wronski for their valuable input. The material test blob in Figure 14 was created by Robin Marin and released under CC (https://creativecommons.org/licenses/by/3.0/).

## REFERENCES

Tomas Akenine-Möller, Cyril Crassin, Jakub Boksansky, Laurent Belcour, Alexey Panteleev, and Oli Wright. 2021. Improved Shader and Texture Level of Detail Using Ray Cones. *Journal of Computer Graphics Techniques (JCGT)* 10, 1 (January 2021), 1–24. http://jcgt.org/published/0010/01/01/

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. ꟻLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2, Article 15 (Aug 2020), 23 pages. https://doi.org/10.1145/3406183

Hendrik Baatz, Jonathan Granskog, Marios Papas, Fabrice Rousselle, and Jan Novák. 2022. NeRF-Tex: Neural Reflectance Field Textures. *Computer Graphics Forum* 41, 6, 287–301. https://doi.org/10.1111/cgf.14449

Yaoyi Bai, Songyin Wu, Zheng Zeng, Beibei Wang, and Ling-Qi Yan. 2023. BSDF Importance Baking: A Lightweight Neural Solution to Importance Sampling General Parametric BSDFs. arXiv:2210.13681

Steve Bako, Pradeep Sen, and Anton Kaplanyan. 2023. Deep Appearance Prefiltering. *ACM Transactions on Graphics* 42, 2, Article 23 (Jan 2023), 23 pages. https://doi.org/10.1145/3570327

Petrik Clarberg, Simon Kallweit, Craig Kolb, Pawel Kozlowski, Yong He, Lifan Wu, and Edward Liu. 2022a. Research Advances Toward Real-Time Path Tracing. Game Developers Conference (GDC).

Petrik Clarberg, Simon Kallweit, Craig Kolb, Pawel Kozlowski, Yong He, Lifan Wu, Edward Liu, Benedikt Bitterli, and Matt Pharr. 2022b. Real-Time Path Tracing and Beyond. HPG 2022 Keynote.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HkpbnH9lx

Jonathan Dupuy. 2015. *Photorealistic Surface Rendering with Microfacet Theory*. Ph. D. Dissertation. Université Claude Bernard - Lyon I ; Université de Montréal.

Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. 2013. Linear efficient antialiased displacement and reflectance mapping. *ACM Transactions on Graphics* 32, 6, Article 211 (Nov 2013), 11 pages. https://doi.org/10.1145/2508363.2508422

Jonathan Dupuy and Wenzel Jakob. 2018. An adaptive parameterization for efficient material acquisition and rendering. *ACM Transactions on Graphics* 37, 6, Article 274 (Dec 2018), 14 pages. https://doi.org/10.1145/3272127.3275059

Jiahui Fan, Beibei Wang, Miloš Hašan, Jian Yang, and Ling-Qi Yan. 2022. Neural Layered BRDFs. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada). Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3528233.3530732

Alban Gauthier, Robin Faury, Jérémy Levallois, Théo Thonat, Jean-Marc Thiery, and Tamy Boubekeur. 2022. MIPNet: Neural Normal-to-Anisotropic-Roughness MIP Mapping. *ACM Transactions on Graphics* 41, 6, Article 246 (Nov 2022), 12 pages. https://doi.org/10.1145/3550454.3555487

Yong He, Kayvon Fatahalian, and Theresa Foley. 2018. Slang: Language Mechanisms for Extensible Real-time Shading Systems. *ACM Transactions on Graphics* 37, 4, Article 141 (Jul 2018), 13 pages. https://doi.org/10.1145/3197517.3201380

Wenzel Jakob, Andrea Weidlich, Andrew Beddini, Rob Pieké, Hanzhi Tang, Luca Fascione, and Johannes Hanika. 2019. Path Tracing in Production: Part 2: Making Movies. In *ACM SIGGRAPH 2019 Courses* (Los Angeles, California). Association for Computing Machinery, New York, NY, USA, Article 20, 41 pages. https://doi.org/10.1145/3305366.3328085

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tomáš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty. 2022. The Falcor Rendering Framework (version 5.2). https://github.com/NVIDIAGameWorks/Falcor

Alexandr Kuznetsov, Miloš Hašan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. 2019. Learning generative models for rendering specular microgeometry. *ACM Transactions on Graphics* 38, 6, Article 225 (Nov 2019), 14 pages. https://doi.org/10.1145/3355089.3356525

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: multi-resolution neural materials. *ACM Transactions on Graphics* 40, 4, Article 175 (Jul 2021), 13 pages. https://doi.org/10.1145/3450626.3459795

Alexandr Kuznetsov, Xuezheng Wang, Krishna Mullia, Fujun Luan, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2022. Rendering Neural Materials on Curved Surfaces. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada). Association for Computing Machinery, New York, NY, USA, Article 9, 9 pages. https://doi.org/10.1145/3528233.3530721

Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels considered harmful: wavefront path tracing on GPUs. In *Proceedings of the 5th High-Performance Graphics Conference* (Anaheim, California). Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/2492045.2492060

Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. 2003. A data-driven reflectance model. *ACM Transactions on Graphics* 22, 3 (Jul 2003), 759–769. https://doi.org/10.1145/882262.882343

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Computer Vision – ECCV 2020*. Springer International Publishing, Cham, 405–421.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4, Article 102 (Jul 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

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

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural control variates. *ACM Transactions on Graphics* 39, 6, Article 243 (Nov 2020), 19 pages. https://doi.org/10.1145/3414685.3417804

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen Space Shading. *Computer Graphics Forum* 36, 4, 65–78. https://doi.org/10.1111/cgf.13225

Marc Olano and Dan Baker. 2010. LEAN mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.). Association for Computing Machinery, New York, NY, USA, 181–188. https://doi.org/10.1145/1730804.1730834

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering, Third Edition: From Theory to Implementation.* Morgan Kaufmann.

Gilles Rainer, Abhijeet Ghosh, Wenzel Jakob, and Tim Weyrich. 2020. Unified Neural Encoding of BTFs. *Computer Graphics Forum* 39, 2, 167–178. https://doi.org/10.1111/cgf.13921

Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. 2019. Neural BTF Compression and Interpolation. *Computer Graphics Forum* 38, 2, 235–244. https://doi.org/10.1111/cgf.13633

Daniel Rebain, Mark J. Matthews, Kwang Moo Yi, Gopal Sharma, Dmitry Lagun, and Andrea Tagliasacchi. 2023. Attention Beats Concatenation for Conditioning Neural Fields. *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=GzqdMrFQsE

Szymon Rusinkiewicz. 1998. A New Change of Variables for Efficient BRDF Representation. In *Rendering Techniques '98*. Springer Vienna, Vienna, 11–22.

Doug Smythe and Jonathan Stone. 2021. MaterialX: An Open Standard for Network-Based CG Object Looks, Version 1.38. https://materialx.org/assets/MaterialX.v1.38.Spec.pdf.

Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. 2021. Neural BRDF Representation and Importance Sampling. *Computer Graphics Forum* 40, 6, 332–346. https://doi.org/10.1111/cgf.14335

Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred neural rendering: image synthesis using neural textures. *ACM Transactions on Graphics* 38, 4, Article 66 (Jul 2019), 12 pages. https://doi.org/10.1145/3306346.3323035

T. S. Trowbridge and K. P. Reitz. 1975. Average Irregularity Representation of a Rough Surface for Ray Reflection. *Journal of the Optical Society of America* 65, 5 (1975), 531–536.

Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Moller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-Access Neural Compression of Material Textures. *ACM Transactions on Graphics* 42, 4, Article 88 (Jul 2023), 25 pages. https://doi.org/10.1145/3592407

Dietger van Antwerpen. 2011. Improving SIMD efficiency for parallel Monte Carlo light transport on the GPU. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (Vancouver, British Columbia, Canada). Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/2018323.2018330

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

Eric Veach and Leonidas J. Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 419–428. https://doi.org/10.1145/218380.218498

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics* 37, 4, Article 124 (Jul 2018), 15 pages. https://doi.org/10.1145/3197517.3201388

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Grenoble, France) *(EGSR'07)*. Eurographics Association, Goslar, DEU, 195–206.

Bing Xu, Liwen Wu, Miloš Hašan, Fujun Luan, Iliyan Georgiev, Zexiang Xu, and Ravi Ramamoorthi. 2023. NeuSample: Importance Sampling for Neural Materials. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA). Association for Computing Machinery, New York, NY, USA, Article 41, 10 pages. https://doi.org/10.1145/3588432.3591524

Chuankun Zheng, Ruzhang Zheng, Rui Wang, Shuang Zhao, and Hujun Bao. 2021. A Compact Representation of Measured BRDFs Using Neural Processes. *ACM Transactions on Graphics* 41, 2, Article 14 (Nov 2021), 15 pages. https://doi.org/10.1145/3490385

## A IMPORTANCE SAMPLING DETAILS

The following outlines the implementation details of our analytic proxy model used for importance sampling.

*Probability density.* Like prior work [Fan et al. 2022; Sztrajman et al. 2021] our sampling density is aS linear blend between a diffuse and specular term

$$p(\boldsymbol{\omega}_\mathrm{o}) = w_\mathrm{d} \cdot p_\mathrm{d}(\boldsymbol{\omega}_\mathrm{o}) + w_\mathrm{s} \cdot p_\mathrm{s}(\boldsymbol{\omega}_\mathrm{o}), \quad \text{with } w_\mathrm{d} + w_\mathrm{s} = 1. \tag{5}$$

The diffuse PDF $p_\mathrm{d}$ is a cosine-weighted distribution but tilted by a normal vector computed from a predicted 2D surface slope $(\mu_{\mathrm{d,x}}, \mu_{\mathrm{d,y}})$ as

$$\mathbf{n}_\mathrm{d} = \mathrm{Normalize}([-\mu_{\mathrm{d,x}}, -\mu_{\mathrm{d,y}}, 1]). \tag{6}$$

The specular PDF $p_\mathrm{s}$ is a standard microfacet density using a Trowbridge-Reitz (GGX) NDF [Trowbridge and Reitz 1975; Walter et al. 2007] with elliptical anisotropy and non-centered mean surface slopes [Dupuy 2015]:

$$p_\mathrm{s}(\boldsymbol{\omega}_\mathrm{o}) = D_\mathrm{std}\left(\frac{\mathbf{M}^{-1}\boldsymbol{\omega}_\mathrm{h}}{||\mathbf{M}^{-1}\boldsymbol{\omega}_\mathrm{h}||}\right)\frac{\det\left(\mathbf{M}^{-1}\right)}{||\mathbf{M}^{-1}\boldsymbol{\omega}_\mathrm{h}||^3}\frac{1}{4\left|\boldsymbol{\omega}_\mathrm{o}\cdot\boldsymbol{\omega}_\mathrm{h}\right|}, \tag{7}$$

where $\boldsymbol{\omega}_\mathrm{h} = \mathrm{Normalize}(\boldsymbol{\omega}_\mathrm{i} + \boldsymbol{\omega}_\mathrm{o})$ is the half vector and $D_\mathrm{std}$ is the isotropic NDF with unit roughness ($\alpha = 1$), transformed based on

$$\mathbf{M} = \begin{bmatrix} \alpha_\mathrm{x} & 0 & -\mu_{\mathrm{s,x}} \\ \alpha_\mathrm{y}\,\rho & \alpha_\mathrm{y}\sqrt{1-\rho^2} & -\mu_{\mathrm{s,y}} \\ 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

Here, the elliptical anisotropy is described by two orthogonal roughness values $\alpha_\mathrm{x}$, $\alpha_\mathrm{y}$ with correlation parameter $\rho$ and the mean of the NDF is offset by a 2D surface slope $(\mu_{\mathrm{s,x}}, \mu_{\mathrm{s,y}})$.

The last two terms in Equation (7) are the Jacobian determinants accounting for the transformation (and subsequent normalization) of $\boldsymbol{\omega}_\mathrm{h}$, as well as the change of variables between $\boldsymbol{\omega}_\mathrm{h}$ and $\boldsymbol{\omega}_\mathrm{o}$.

*Sampling.* The sample transform $W$ first selects one of the two PDF terms (Equation (5)) based on the relative weights $w_\mathrm{d}$ and $w_\mathrm{s}$. If the diffuse component is chosen we simply generate a cosine-weighted outgoing direction $\boldsymbol{\omega}_\mathrm{o}$ and tilt it based on $\mathbf{n}_\mathrm{d}$. Otherwise, we perform specular reflection along a sampled half-vector

$$\boldsymbol{\omega}_\mathrm{h} = \mathrm{Normalize}(\mathbf{M}\cdot W_\mathrm{std}(\mathbf{u})) \tag{9}$$

where $W_\mathrm{std}$ is the usual isotropic NDF sampling technique ($\alpha = 1$).

*Network prediction.* We dropped the explicit dependence of $p$ and $W$ on $\boldsymbol{\omega}_\mathrm{i}$ and $\mathbf{x}$ above for brevity, but our full set of 9 proxy parameters $\{w_\mathrm{d}, \mu_{\mathrm{d,x}}, \mu_{\mathrm{d,y}}, w_\mathrm{s}, \alpha_\mathrm{x}, \alpha_\mathrm{y}, \rho, \mu_{\mathrm{s,x}}, \mu_{\mathrm{s,y}}\}$ are the result of an MLP evaluation that takes these as input. To ensure that all inferred parameters lie in their respective valid ranges ($\alpha \in [0,1], \rho \in [-1,1], \mu \in [-\infty, +\infty]$) we append an appropriate final activation to each network output based on quadratic approximations of $\tanh(x)$ and $\sinh(x)$. Lastly, $w_\mathrm{d}$ and $w_\mathrm{s}$ are processed by the softmax function to form valid mixing weights that add up to one.