CHAPTER 4. NEURAL FIELDS FOR BTF ENCODING AND TRANSFER

Figure 4.1: On the left, renders generated with NeuBTF materials. On their right, a slice of the input training BTF and the guidance image onto which we propagate the BTF measurements.

Neural material representations are becoming a popular way to represent materials for rendering. They are more expressive than analytic models and occupy less memory than tabulated BTFs. However, existing neural materials are immutable, meaning that their output for a certain query of UVs, camera, and light vector is fixed once they are trained. While this is practical when there is no need to edit the material, it can become very limiting when the fragment of the material used for training is too small or not tileable, which frequently happens when the material has been captured with a gonioreflectometer. In this chapter, we propose a novel neural material representation that jointly tackles the problems of BTF compression, tiling, and extrapolation. At test time, our method uses a guidance image as input to condition the neural BTF to the structural features of this input image. Then, the neural BTF can be queried as a regular BTF using UVs, camera, and light vectors. Every component in our framework is purposefully designed to maximize BTF encoding quality at minimal parameter count and computational complexity, achieving competitive compression rates compared with previous work. We demonstrate the results of our method on a variety of synthetic and captured materials, showing its generality and capacity to learn to represent many optical properties. The contributions in this chapter led to the following publication, currently under review:

"NeuBTF: Neural Fields for BTF Encoding and Transfer"

Carlos Rodriguez-Pardo, Konstantinos Kazatzis, Jorge Lopez-Moreno, Elena Garces

Under Review (2023)

4.1 Introduction

A common approach to modeling real-world spatially-varying materials in computer graphics is through the use of Bidirectional Texture Functions (BTFs). This type of representation models the dense optical response of the material, and is more general than analytic representations such as a micro-facet SVBRDF. However, BTFs typically occupy large amounts of memory. Recently, neural material representations are being proposed as an learning-based alternative to tabulated BTFs, providing a more compact solution while keeping the flexibility and generality of BTFs.

Creating digital representations of real material samples requires using an optical capture device, such as a gonioreflectometer [Gar+23], a smartphone [Des+18; Des+19; Guo+20b; Hen+21], or a flatbed scanner [Rod+23a]. During the process, several choices must be made. First, it is important to select a patch of the material that contains enough spatial variability. Second, a process śautomatic or manualś must be found to produce a tileable material that can be used to create seamless 3D renders. Finally, resources must be allocated for storage as needed. Making these choices when dealing with implicit or tabulated representations, such as in BTFs or neural materials, is particularly crucial. Once these representations are trained or captured, they cannot be easily modified and it is only possible to query them using the UVs, light, and camera vectors.

In this chapter, we propose a novel neural material representation that addresses these issues. Unlike existing neural approaches that are immutable once trained [Rai+19; Kuz+21; Rai+20; Kuz+22], our model can be queried a test time with a guidance image that conditions the neural BTF to the structure provided by the guidance image. Our approach resembles synthesis by example and procedural processes, and can be used to extrapolate BTFs to large material samples, as well as to easily create tileable ones. Furthermore, our method achieve better compression rates than previous work on neural BTF representations.

To achieve this, we present a novel deep learning method that works at two steps. In the first step, we condition the neural BTF using a *guidance image*

as input. To this end, we use an *autoencoder* that outputs a high-dimensional latent representation of the material, a *neural texture*, which jointly encodes reflectance and structural properties. In the second step, the UV position of the latent representation, along with the camera and light vectors, are decoded by a fully-convolutional sinusoidal decoder [Sit+20], a neural *renderer* to obtain the RGB values. Using a single BTF as input, we train the network end-to-end using a custom training procedure, loss function, and data augmentation policy. This policy, inspired by recent work on attribute transfer [RG21] (Chapter 2), allows the autoencoder to encode the relationship between structural features and reflectance, enabling the propagation of the BTF to novel input guidances. Once trained, the novel input guidances may come from the same material, a different material, or a structural pattern. An input guidance of the same material can be used to extrapolate the BTF to larger samples or create tileable BTFs, provided the input guidance is tileable. If the input guidance is a structural pattern, the local features can be used to synthesize novel materials.

In summary, we propose the following contributions:

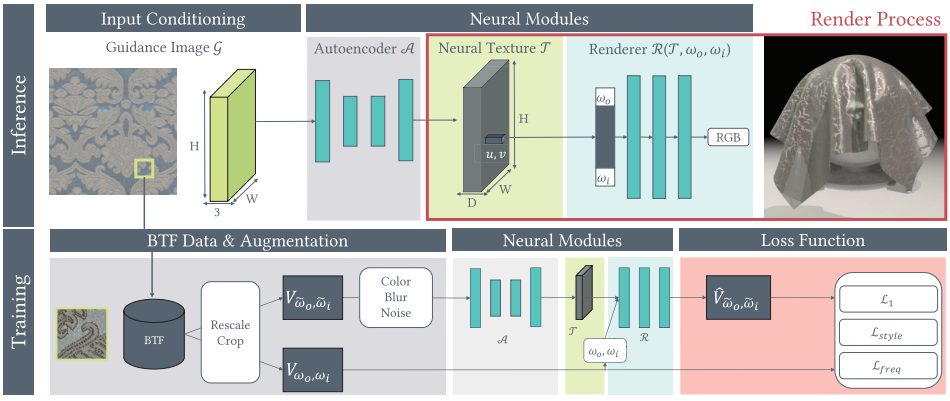
- The first neural BTF representation with conditional input that can be used to extrapolate BTF measurements, easily create tileable BTFs, and synthesize novel materials.
- We show how to leverage our system for rendering large-scale and tileable neural BTF generation using measurements captured with small portions of the material.
- We demonstrate that our method works with synthetic and captured materials of diverse optical properties, including colored specular or anisotropy.

4.2 Related Work

An accurate method for representing the optical properties of materials is through Bidirectional Texture Functions (BTFs) [Dan01]. BTFs are 6D functions that characterize all possible combinations of incoming and outgoing light and camera directions for the 2D spatial extent of a material. Although they are successful in representing materials, they have a major drawback in terms of memory requirements. Therefore, BTF compression has been a major research topic [FH08]. Non-neural approaches used dimensionality reduction techniques such as Principal Component Analysis (PCA) [Kou+03; MMK03; WGK14], vector quantization [HFM10], or clustering [Ton+02]. However, these approaches were recently surpassed by neural models [HS06] due to their flexibility and superior capacity to learn non-linear functions.

Neural BTFs Rainer *et al.* [Rai+19] proposed the first method to use deep autoencoders to compress BTFs, surpassing PCA [WGK14] on captured BTFs. However, this approach required training a single neural network per material. To address this limitation, a later work by the same authors [Rai+20] proposed a generalization of this idea in which a single network was able to generalize to a variety of materials. Although these methods were very effective for compressing flat materials, they had some limitations when it came to modeling materials with volume. In their work, Kuznetsov *et al.* [Kuz+21] improved the quality of neural materials by introducing a neural offset module that captures parallax effects. Further, their method also allowed for level-of-detail though MIP mapping by training a multi-resolution neural representation. However, grazing angles and silhouette effects remained a challenge for this approach. In a subsequent work, Kuznetsov *et al.* [Kuz+22] explicitly trained the network using queries that span surface curvatures, effectively handling these cases. Representing fur, fabrics, and grass with neural reflectance fields was explored by Baatz *et al.* [Baa+22] who proposed a representation that jointly models reflectance and geometry

Figure 4.2: An overview of our neural BTF inference and training processes. Top-Inference: Using a guidance image $\mathrm{g} \in \mathbb{R}^{H \times W \times 3,}$ we use our trained autoencoder to generate the Neural Texture $\mathrm{A(G)}=\mathrm{T_g} \in \mathbb{R}^{H \times W \times D,}$ which preserves the spatial resolution of the input image but represents a higher-dimensional learned representation. This Neural Texture $\mathrm{T_g}$, along with the trained renderer R, can be queried as a regular BTF, using UVs, and target camera and light positions for regular rendering. Bottom-Train: During training, we randomly select an input *view* $\mathrm{V}_{\tilde{\omega}_o, \tilde{\omega}_i}$, and a target $\mathrm{V}_{\omega_o, \omega_i}$. To both views, we apply random rescale and cropping. Then, only to $\mathrm{V}_{\tilde{\omega}_o, \tilde{\omega}_i}$, we randomly apply hue variations, gaussian blur, and noise, and feed it to the *autoencoder*, which returns a 2D *latent representation* of the material. A fully-convolutional *decoder* with sinusoidal activations receives both this latent space and the target ω_o, ω_i camera and light angles, and estimates $\hat{V}_{\omega_o, \omega_i}$. This output is compared with V_{ω_o, ω_i} using a multifaceted loss function.



All of these approaches share the idea of querying neural material using UVs, camera, and lighting vectors, but do not provide any functionality for modifying the material once the network is trained. In contrast, our approach can take a guidance image as input, which conditions the output to generate material variability.

Material Synthesis and Tiling Texture synthesis is a long-standing problem in the field of computer graphics. The goal is to reconstruct a larger image given a small sample, lever-aging the structural content and internal statistics of the input image. This concept has been used for synthesizing single images, BTFs, and full material models. For images, the most common strategies include PatchMatch [Dia+15], texture transport [AWL15], point processes [Gue+20; LH06], or neural networks [EM17; Zho+18; FAW19; RG22]. BTF synthesis, however, has received less attention. Steinhausen et al. [Ste+15a; Ste+15b] extrapolated

BTF captures to larger material samples using non-neural texture synthesis methods. For full materials, Li *et al.* [LPG19] captured the appearance of materials by first estimating their BRDF and then synthesizing the high-resolution micro-structure from a dataset of measured SVBRDFs. Nagano *et al.* [Nag+15] measured microscopic patches of the skin and used a convolutional filter to propagate the measurements to a spatially-varying texture. Deschaintre *et al.* [DDB20] used an autoencoder to propagate SVBRDFs to large material samples. Also recently, Rodriguez-Pardo and Garces [RG21] propagated any kind of visual attribute having a single image as guidance. Their approach shares some similarity to ours, although they transfer 2D image attributes, while we transfer the full BTF.

Procedural models [Hu+22b; Zho+23; Zho+22] are nowadays very successful for generating tileable materials. Thanks to the use of a tileable template, these methods adjust the generated image to the features available in the template. As we show, our approach can also work with a binary template as input. However, guaranteeing predictable outputs given this kind of input is out of the scope of our technique, which can transfer BTF measurements having as input a guidance image of the same material.

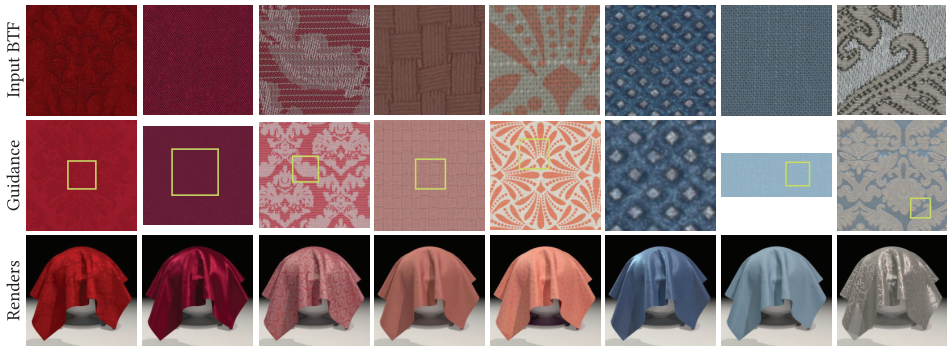
Other Neural Representations in Rendering Limited to BRDFs, neural networks trained with adaptive angular sampling have been explored to enable importance sampling [Szt+21], needed for Monte Carlo integration. Deep latent representations also allow for BRDF editions. For example, Hu *et al.* [Hu+20] demonstrate that autoencoders can outperform classic PCA for the purpose of editing. Other applications of neural encodings in rendering are numerous. For instance, they have been used for scene prefiltering [BSK23], where geometry and materials are simplified to accommodate the LoD of the scene using a voxel-based representation and trained latent encodings.

For anisotropic microfacets, Gauthier *et al.* [Gau+22] propose a cascaded architecture able to adjust the material parameters to the MIP mapping level. Encoding light transport using neural networks for real-time global illumination has also been explored [Rai+22; GMX22], showcasing promising results.

4.3 Method

We present an overview of our approach in Figure 4.2, where we show our inference and training pipelines. Our goal is twofold: First, find a compact representation for a BTF through the use of neural networks. Second, enable the extrapolation of the BTF according to guidance images used as input. In Section 4.3.1 we describe our inference pipeline and neural network, and in Section 4.3.2 our training process. Section 4.4 contains specific implementation and design details of the neural networks.

Figure 4.3: Some tileable neural materials achieved with our method. On the top row, we show a slice of the BTF used to train a NeuBTF representation. With the tileable guidance images shown on the middle row, we propagate the neural texture using our autoencoders. These neural textures can be rendered to generate realistic images (bottom). We highlight the training dataset surface area as a green inset.



4.3.1 Inference

Our neural network is composed of three modules: an *autoencoder* A, a *neural texture* $T \in \mathbb{R}^{H \times W \times D}$, and a *renderer* R. The renderer $R(T(u,v),\omega_o,\omega_i)$=RGB takes as input the feature vector at the (u, v) coordinates of the neural texture T, the view ω_o and light ω_i positions, and returns an RGB value. R acts as a conventional BTF and can be used as such in any render engine. An input *guidance* image, $G \in \mathbb{R}^{H \times W \times 3}$, is used during inference to condition the generation of the *neural texture* T. This conditioning allows us to propagate the

learned reflectance to novel guidance images that can be: a larger sample of the same material, a different material, or a structural image. In the simpler case, the guidance image comes from the BTF used for training, and our process is equivalent to previous work [Rai+19; Kuz+21].

The autoencoder A takes the guidance image $G \in \mathbb{R}^{H \times W \times 3}$ as input and outputs a neural texture $T \in \mathbb{R}^{H \times W \times D}$ with the same size $H \times W$ as the input guidance image but with more latent dimensions D. As a result, each pixel in the guidance image has a higher-dimensional neural representation in T. Because it is trained without explicit supervision, this latent representation can capture the reflectance and structural patterns automatically. Kuznetsov *et al.* [Kuz+21] also used a latent neural representation of the material, however, lacking the initial autoencoder their approach cannot synthesize novel BTFs without retraining, while our conditioning module allow us to generate novel BTFs during test time. The autoencoder and the renderer are neural networks trained jointly, using an end-to-end image-to-image approach we describe below.

4.3.2 Training

Figure 4.2 (bottom) illustrates our training process. It has two objectives: First, equivalent to regular BTF encoding, we aim to find the mapping between camera direction ω_i, light direction ω_i, and output slices of the BTF: $(\omega_o, \omega_i) \rightarrow V_{\omega_o, \omega_i} \in$ BTF. Second, we aim to condition the synthesis process with an input guidance image, G.

To this end, for training, we feed the model with images, $V_{\tilde{\omega}_0, \tilde{\omega}_i}$ which are randomly sampled from the BTF, and are subject to additional data augmentation processes.

This extensive augmentation process guarantees invariance to different input variations during test time, like camera or illumination conditions, while keeps consistency of the outputs.

Loss Function Design Our loss function, that compares ground truth slices $V\omega_0, \omega_i$ with generated ones $R(A(f(V_{\tilde{\omega}_0, \tilde{\omega}_i})), \omega_o, \omega_i)$, is a weighted sum of three terms: a pixel-wise loss, a style loss, and a frequency loss,

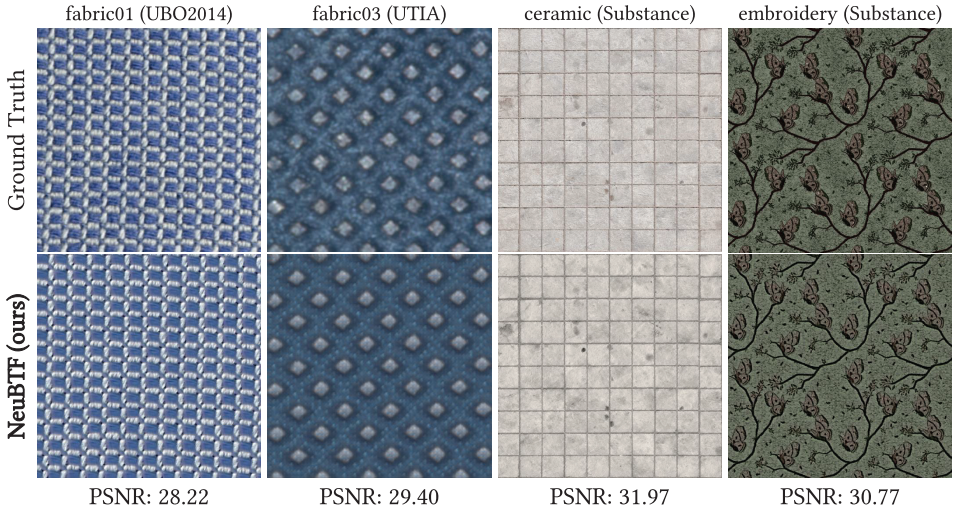
$$\mathcal{L} = \lambda_{\mathcal{L}_1} \mathcal{L}_1 + \lambda_{style} \mathcal{L}_{style} + \lambda_{freq} \mathcal{L}_{freq} \tag{4.1}$$

The main driver of our loss is the pixel-wise norm L_1. L_1 produces sharper results than higher-order alternatives, such as L_2 [Iso+17; RG21]. Following [Kuz+21], we apply a $log(x + 1)$ compression to improve the model results on high dynamic range. This compression is only done to the pixel-wise component of the loss function. Inspired by recent work on texture synthesis, capture

and transfer [Mar+20; AWL13; RG22; Rod+23a; Zho+22], we introduce a L_{style} loss to help the model generate higher quality and sharper results. Further, to mitigate the spectral bias of convolutional neural networks and help ameliorate the results further, we also introduce a *focal frequency loss* into our learning framework [Jia+21]. This combination of loss functions proves effective for our problem, without the need for complex adversarial losses which could reduce efficiency or destabilize training.

Data Augmentation We train our models using a comprehensive data augmentation policy aimed at achieving high quality reflectance propagation, increasing performance and generalization, and allowing for generation of multiple resolution materials at test time. We build upon recent work on material transfer [RG21] and use images of the material taken under different illumination and viewing conditions as inputs to our autoencoder. This helps it generalize to novel capture setups, which allows for multiple applications we describe on Section 4.6. In particular, we use every image available on the input BTF, selected uniformly at random for each element in each batch during training. As in [Tex+20b; RG21; Rod+23a], we also use random rescaling, which helps the model generalize to new scales, and, and build neural materials of multiple resolutions at test time, as we describe on Section 4.6.3 inspired by recent work on image synthesis [RG21; Tex+20b; RG22], we use random cropping, which helps generalization by effectively increasing the dataset size. Finally, we extend the color augmentation policy in [RG21] with

Figure 4.4: Qualitative results on a variety of BTFs, from different sources. From left to right, we show results on the UBO [WGK14] BTF dataset, the UTIA [Hai12] BTF dataset and two synthetic materials rendered from Substance SVBRDFs.



random hue changes across the entire color wheel, and introduce random Gaussian noise and blurs to the input images, to help it generalize further, as proposed in [Rod+23a] (Chapter 5).

4.4 Implementation details

Our model size, function weighting, optimizer and training configuration, and data augmentation hyperparameters were selected using a combination of manual tuning and Bayesian hyperparameter optimization using *Weights and Biases* [Bie20].

Autoencoder For the *autoencoder*, we use a lightweight U-Net [RFB15] with a few modi-fications to tailor it for our problem. Inspired by recent work on CNN design, we leverage ConvNext [Liu+22b] Blocks across our model, with depth-wise convolutions using 5 × 5 ker-nels. We empirically observe that ConvNext blocks achieve higher quality structural editions at a lower parameter count than vanilla U-Net blocks. To further help convergence and preserve details in the input images, we use residual connections [He+16; Dia+20; Rod+23a] in every convolutional block of the model. We use 1 × 1 convolutions on the skip connections and residual scaling [Kar+20a]. As in [Liu+22b], we use Layer Normalization [BKH16] and GELU non-linearities [HG16]. On the bottleneck of the model, we introduce an attention module [Woo+18] to help the model learn longer-range dependencies. To avoid checkerboard artifacts [ODO16], we use nearest neighbors interpolation for upsampling. Inspired by recent work on tileable material generation [Zho+22], we use *circular padding* throughout the model. We initialize its weights using *orthogonal initialization* [ACB19], which helps avoid exploding gradients.

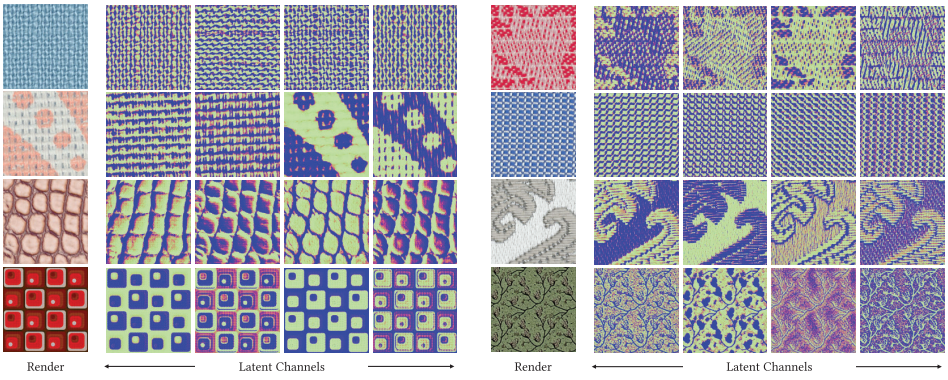
Renderer For the *renderer*, we build upon SIREN [Sit+20] MLPs, with additional modifi-cations to enhance its performance for our problem. We use 1 × 1 convolutions instead of vanilla linear layers, to allow for end-to-end training using 2D images. Further, we introduce Layer Normalization [BKH16] before each sinusoidal non-linearity, which stabilizes train-ing. Finally, inspired by [Meh+21], we use residual connections [He+16], to help preserve the information of the input vector across the decoder layers. Model weight initialization follows [Sit+20]. With sinusoidal activations, we observe significantly higher reconstruction quality and training dynamics than with ReLU [NH10] MLPs, which are common for BTF compression [Rai+19; Rai+20; Kuz+21]. Because the network is fully-convolutional, it can take as input feature vectors of any size. This is very convenient for our use cases when the input guidance image have a size different from the size of the original BTF used to train it.

Data Generation We generate synthetic ground truth BTF data using a proprietary CPU path-tracer based on the architectures of PBRT [PJH20],

Mitsuba [Nim+19] and Wave-front [LKA13]. The scene is set up with a single plane-mesh, and a directional light with unit irradiance rotating around the surface. We use a custom camera for skewed configurations to avoid post processing. We render a total of 81 × 81 = 6561 HDR images, changing the camera and lighting angles for each, following [SSK03]. The data are generated with 16 samples per BTF texel using a 10-core CPU, which takes approximately one hour for 512 × 512 resolution materials. For the rendered SVBRDFs, we use the material model in [Gar+23] (Chapter 2.C), without the transmittance and opacity maps. We use both procedurally and scanned SVBRDFs to generate our sythetic BTFs.

For acquired BTFs, we test on a variety of publicly available datasets, including *UBO 2003* [SSK03], *UBO 2014* [Rei+14], *Atrium*, *UTIA 2012* [Hai12], or *UTIA MAM 2014* [Fil+18]. Each of those has their own spatial and angular resolutions, we refer the reader to their individual specifications and to the Survey in [HS13] for additional details. We use *btf-extractor* to read the acquired BTF files. Our method works seamlessly with both measured and synthetically generated data without the need to change neither the neural module or the training procedure.

Figure 4.5: A selection of latent channels learned by NeuBTF for a variety of materials. We use a colorspace to help visualization. Without explicit supervision, the model internally learns semantically meaningful latent spaces. For instance, in the second example on the left, the two leftmost latent spaces encode geometry, while the other two encode the two distinct colors of the printed pattern over the yarns.



Optimization We use PyTorch [Pas+19] and Torchvision [MR10] for training. We train our models for 300 epochs, using a batch size of 80, and an input crop size of 128 × 128 pixels, as in [RG21; RG22; Rod+23a]. We set an initial learning rate of 0.0015, which we divide by 2 every 100 iterations using a step scheduler. We empirically observe training instabilities using Adam [KB15] on early iterations.

To mitigate this, we use RAdam [Liu+19a], which significantly stabilizes training. We also perform gradient norm clipping [Zha+20] (set to $maxnorm = 10$) and an $\epsilon = 10^{-6}$. We use a Dropout [Sri+14] rate of 0.1 on the autoencoder and a weight decay of 10^{-6}, to avoid overfitting. To further regularize the models, we leverage mixed precision training and automatic scaling [Mic+18]. This training process takes approximately 3 hours per material on a Nvidia RTX 3060 GPU. These training times are constant regardless of the material spatial resolution. For comparison, our model takes around four times longer to train than previous work [Kuz+21], because we need to optimize the autoencoder parameters and our loss function is multifaceted. Convergence is typically achieved on 100-150 iterations, longer training only helps resolve additional details. A subset of the training set is also leveraged as a validation set, which helps track the model performance on larger images. We use this validation dataset for early stopping regularization. During test time, we evaluate the model using *half precision*.

Loss Function We weight each part of the loss function as: $\lambda_{style} = 0.25$, $\lambda_{freq} = 1$, and $\lambda_{\mathcal{L}} = 5$. For $\lambda_{style,}$ we use the *AlexNet* variant of *LPIPS* [Zha+18], which allows for a lightweight yet powerful texture regularizer [RG22; Rod+23a]. This metric expects input on the [−1, 1] range, so we perform a simple $2x - 1$ transformation to the model target and output, which lie on the [0, 1] range before computing the loss.

Data Augmentation In order, we apply to both input and target images: random rescales in the range of [0.25, 1.5] and random cropping. Then, only to the input image: random hue changes, random grayscale (p = 0.2), random blurs (p = 0.2, kernel size = 5) and random Gaussian noise (p = 0.2). We empirically observe that higher probabilities, or more complex policies like random erasing [Zho+20] were detrimental to the model performance for some materials. We use Kornia [Rib+20] for these augmentations.

Model Sizes Our *autoencoder* has 4 hidden layers in the encoder, with a *width factor* of 5. All other model details follow [RFB15]. The outputs of the autoencoder (e.g. the 2D latent space) are regularized with Layer Normalization [BKH16]. We use D = 14 latent dimensions for our neural materials, following [Kuz+21]. For the *decoder*, we use 3 hidden layers with 40 neurons each, achieving similar model sizes as previous work on neural BRDF representations [Kuz+21; Szt+21]. Following [Kuz+21], introduce a ReLU non linearity [NH10] to the final layer of the decoder to avoid negative reflectance values.

4.5 Evaluation

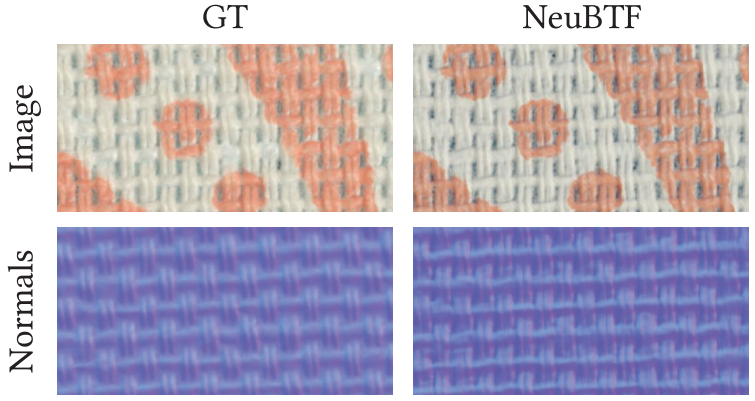
4.5.1 Qualitative Analysis

We evaluate NeuBTF on materials from different sources including acquired BTFs from [WGK14; Hai12], and rendered BTFs from procedurally generated and

scanned SVBRDFs. In Figure 4.3, we show examples of the results of NeuBTF for a variety of materials with highly complex structures and reflectance properties, like colored specular (first column) or anisotropy (last). We show some additional results in Figure 4.4 for materials of different datasets. As shown, our model achieves high quality reconstructions regardless on the type of data source.

In Figure 4.5, we show a colored visualization for a few channels of the latent neural texture T found for a variety of materials. Because the values for the neural texture are unbounded, to each channel $c \in T$, we standardize them to 0 mean and unit variance, and apply a $sigmoid(c) = \frac{1}{e^{1-c}+1}$ non-linearity to make the maps comparable. Without any explicit training, the models learn to separate distinct parts of the material. For example, the model finds distinct latent spaces for *warp* and *weft* yarns on woven fabrics, or separation between color and geometric patterns. This disentanglement provides clues on why the material propagation is possible, and suggests potential future research directions for fine-grained neural material edition.

Figure 4.6: Surface normals estimated using *Photometric Stereo* [Ike81] for ground truth materials and the materials reconstructed by NeuMat. As shown, the encoded materials preserve geometric information without explicit training.



Finally, we also evaluate whether the encoded materials preserve the geometric properties in their training datasets. We leverage *Photometric Stereo* [Ike81] to compute surface normals of encoded and ground truth materials. We provide results in Figure 4.6, where we show that the reconstructed surface geometry is accurate albeit noisier, and preserves the structure of the material.

4.5.2 Compression comparisons with previous work

In Table 4.1, we show the number of trainable parameters on the decoders of different neural BTF compression algorithms. As shown, our model is competitive with previous work in terms of trainable parameters. This is achieved as we use more complex loss functions than previous work, which help regularize the models, and because our sinusoidal MLP achieves higher quality reconstructions for natural signals than ReLU MLPs, as shown in [Sit+20]. NeuMIP [Kuz+21] uses smaller MLPs, however, they require an additional decoder for their *neural offset* module, which helps them encode parallax effects (See Figure 4.7), for which our model struggles. Our decoder has one order of magnitude fewer parameters than [Rai+19; Rai+20], however, the method in [Rai+20] provides the benefit of fast encoding of new materials, while ours requires a different model for each new material.

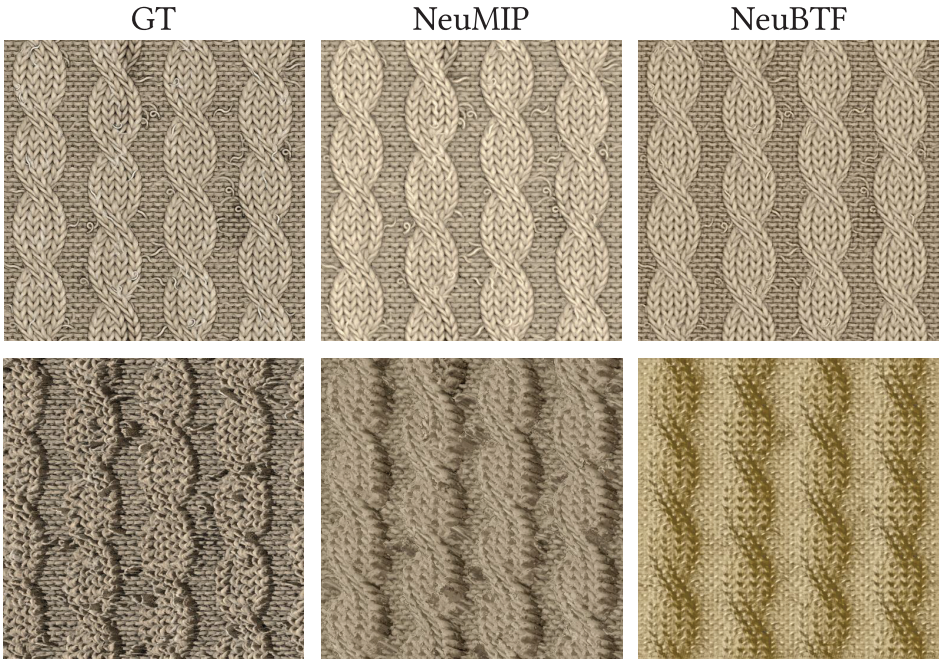
4.5.3 Limitations

As we show in Figure 4.7, our model struggles with materials with strong displacement. While our method provides accurate encodings on viewing angles close to the material surface, it cannot accurately encode grazing angles for such extreme cases. Displacement maps translate the geometric position of the points over the surface, breaking the underlying assumptions behind our neural texture. NeuMIP [Kuz+21] solves this issue by explicitly modelling parallax effects with a *neural offset* module. While we did not observe that such extension was needed for acquired BTF data, like the UBO2014 [Rei+14] dataset, introducing a similar module into our editable neural material framework is an interesting future research direction to increase its generality.

Table 4.1: Number of trainable parameters in the decoders and amount of latent texture channels for different neural BTF compression methods, measured using Torchinfo [Yep20]. Exact comparisons are challenging, as [Kuz+21] optimizes a multi-level texture pyramid and [Rai+20] learns a latent vector which can encode novel materials. For neither our method nor [Rai+19; Rai+20], we count the parameters in the encoders, as they are not needed for using the materials on rendering systems.

Method	**NeuBTF (ours)**	[Kuz+21]	[Rai+19]	[Rai+20]
Decoder Parameters	**3011**	3332	35725	38269
Texture Channels	**14**	**14**	**14**	38

Figure 4.7: A failure case of our method. Compared to NeuMIP [Kuz+21], which explictly models parallax effects, our model struggles to accurately encode materials with strong displace-ments, as this synthetic cable knit from Substance3D. For this type of materials, NeuBTF accurately encodes orthogonal viewing angles (top row), however, it struggles at grazing angles (bottom row)



4.6 Applications

4.6.1 Reflectance Propagation and Tileable BTFs

Many material reflectance acquisition devices are limited in the surface dimensions they can digitize. This hinders their applicability to many real-world materials, which exhibit variations that cannot be captured at such small scales. Further, in many applications like SVBRDF acquisition, obtaining larger samples of the material improves realism and helps tileable texture synthesis. In this context, previous work on BTF reflectance compression inherits the surface area limitations of the capture devices used to generate their training data. Our method can easily be applied for reflectance propagation. We build upon the work of Rodriguez-Pardo and Garces [RG21] and leverage our *encoder* to propagate the neural texture optimized using a small portion of the material (e.g. a 1 × 1 cm capture) to a larger portion of the same material,

represented with a *guidance image* captured using a commodity device like a flatbed scanner. Because our model is trained using a large amount of lighting conditions, as in [RG21], the propagation is invariant to how the images are illuminated. We show results of such pipeline in Figure 4.3. For instance, on the last row, we show an anisotropic and specular Silver Jacquard fabric, for which we generated a BTF by rendering a 1 × 1 cm SVBRDF. This small crop cannot represent the complex pattern in the fabric, which we show on the *guidance image*, which covers a 10 × 10 cm area. Using our encoder, we propagate the neural material to this guidance image, generating a new, high-resolution, latent space which we can render, enabling realistic material representations with a reduced digitization cost. This propagated neural material has a 2000 × 2000 texels resolution, and it requires no re-training during test time.

Relatedly, our propagation framework can also be easily leveraged for generating tileable BTFs. Given any *guidance image* of the material, we can generate a tileable version of it, either using manual editions by artists or automatic algorithms [Mor+17; Li+20; Rod+19; RG22]. With this tileable input guidance, we can use our autoencoder A to propagate the neural texture T , effectively generating tileable BTFs, as we show in Figure 4.3. This propagation algorithm can leverage state-of-the-art algorithms for tileable texture synthesis without any modification of our material model or training framework. Tileable BTFs were not achievable with previous approximations and this simple pipeline has the potential of enabling novel applications of this type of material representation in rendering scenarios.

4.6.2 Structural Material Edition

Besides propagating BTF measurements to larger portions of the material, NeuBTF allows for generating novel materials using structural editions. Given a trained NeuBTF and a guidance image representing some particular target structure, we can propagate the neural texture to this guidance image, generating high-quality neural materials which preserve the structure of the guidance image and the reflectance properties of the trained neural material. This pipeline allows for easily generating multiple different neural materials without the need for retraining. As we show in Figure 4.8, this propagation method works for many types of input guidances, including vector black and white images, procedurally generated textures, or real photographs of materials and textures. We show results on acquired BTFs and from synthetic BTFs, rendered from scanned and manually generated SVBRDFs. As shown, our propagation frameworks provides high-quality material editions, even for very challenging cases, like the *circles pattern*.

4.6.3 Multi Resolution Neural Materials

Another useful application enabled by our method is the generation of materials at different resolutions. Unlike previous work [Kuz+21], which explicitly optimizes a pyramid of levels of detail during training, we can generate materials at any resolution at test time without introducing any additional complexities to our material representation. Because we train our models using random rescales as a data augmentation policy, they are equivariant to rescales of its input guidance images $G{\downarrow}{:}R(A(G)){\downarrow} = R(A(G{\downarrow}))$. As such, we can generate any continuous resolution for a particular BTF by downsampling the guidance image to the target resolution and propagating its neural texture, as we illustrate in Figure 4.9. Note that this algorithm only guarantees accurate results for the rescaling ranges that we use during data augmentation.

4.7 Conclusions

We have presented a learning based representation for material reflectance which provides efficient encoding and powerful propagation capabilities. Our method introduces input conditioning into neural BTF representations. This allows for multiple applications which were not possible with previous neural models, including BTF extrapolation, tiling, and novel material synthesis through structure propagation. Our method builds upon recent work on neural fields, network design and data augmentation, showing competitive compression capabilities with previous work on neural BTF representation. Through multiple analyses, we have shown the capabilities of our method on a variety of materials with different reflectance properties, including anisotropy or specularity, as well as effectively handling either synthetic or acquired BTFs.

Our method can be extended in several ways. The most immediate extension is to allow for materials with strong parallax effects due to displacement mapping or curvature, as in [Kuz+21; Kuz+22]. Our representation is limited to opaque materials. Extending them to handle translucent or holed surfaces would increase their realism in materials like thin fabrics or meshes. Further, our method could be extended to allow for hyperspectral BTF data [RSK10], but captured data is scarce. Besides, recent work on neural BRDF representa-tions [Szt+21; Rai+22; Fan+22] and generative models [Mül+19] suggests a promising research direction: Learning to sample from neural BTFs, using invertible neural networks. While these may introduce challenging complexities to the models, they could provide efficient representations for Monte Carlo rendering using importance sampling. Further, building upon recent work on SVBRDF capture [Rod+23a; Zho+22], BRDF sampling [NJR15] and BTF compression

[Rai+20], it could be possible to learn a prior over neural BTFs with a generative model. This should help in capturing more efficiently the data needed for generating these assets, as well as generating new materials and interpolating between them. Finally, editing semantic and reflectance properties in neural fields is an active area of research [WTX22; Liu+21; Wan+22a; Ye+22; Haq+23]. While our method introduces structural edition into neural BTF representations, it is not capable of editing particular semantic properties, such as albedo or specularity. Extending our edition capabilities to more fine-grained parameters is an interesting research avenue. We hope our method inspires future research on neural material representations.

Figure 4.8: Examples of structural editions allowed by our method on a variety of materials. On the leftmost column, we show *guidance images*, which represent structures into which we transfer the neural BTF measurements illustrated on the top row. As shown, our method can effectively propagate BTF measurements into many material and structure types, using as guidances either synthetic or real images. The first three materials (*leather11*, *carpet07*, *fabric01*) are taken from *UBO 2014* [Rei+14], the *linen* material is rendered from a captured SVBRDF, while the *ceramic* material is rendered from an artistic material taken from *Substance3D*. We show renders generated using $\theta_v = \theta_l = 0$.

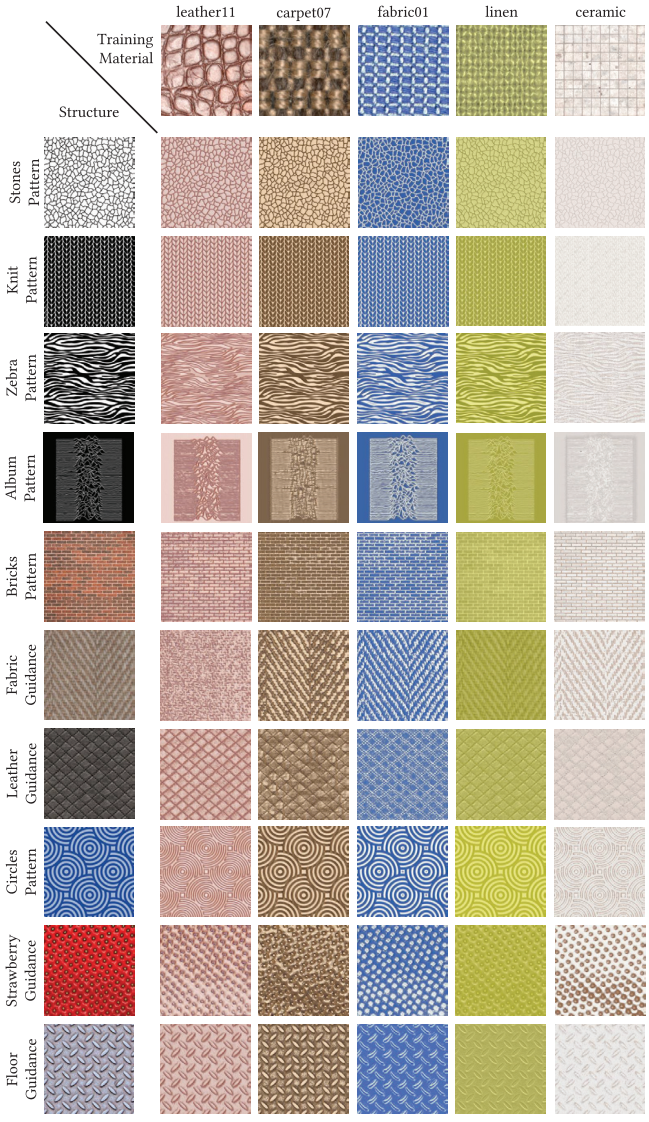
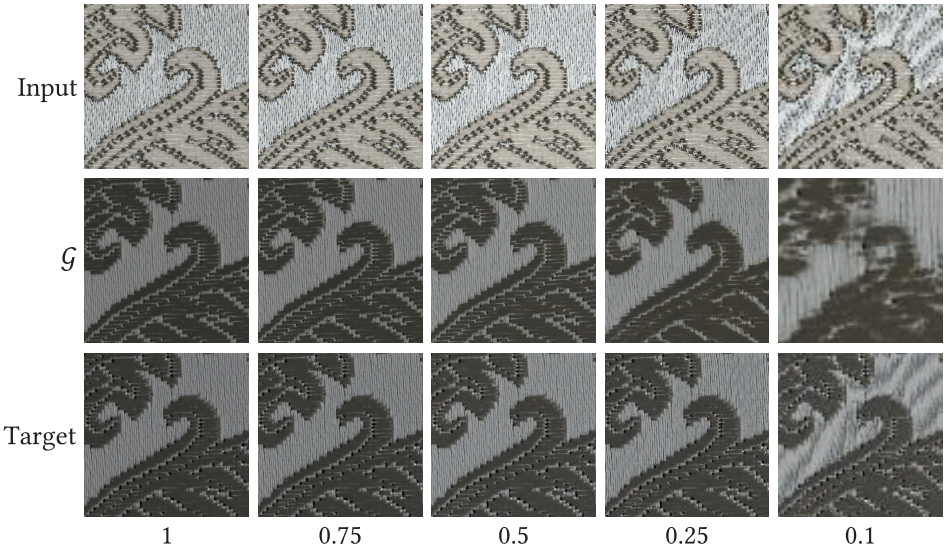


Figure 4.9: Our method naturally enables for the generation of different resolutions for the neural materials. We show the input to the autoencoder (top row), the rendered material at $\theta_c = 75, \theta_l = 60, \phi_c = \phi_l = 0$ (middle row), and the ground truth image at those posi-tions (bottom), at different resolutions (columns). We achieve this by downsampling the guidance image G fed into the autoencoder module, which returns an accurately down-sampled latent space, thanks to our data augmentation policy applied during training. The rightmost column lies beyond the ranges in which we train the model, however, the results are still somewhat plausible.



4.A Model Implementation Details

Figure 4.10: A full diagram of our U-Net autoencoder A. We use a CBA77777M [Woo+18] module on the bottleneck, Layer Normalization [BKH16], and residual connections in every convolutional block. Inspired by ConvNext Blocks [Liu+22b], we use 5×5 depthwise convolutions followed by 1×1 linear layers, with GeLU non-linearities [HG16]. We use 14 latent dimensions for our neural materials, following [Kuz+21]. We use *circular padding* throughout the model and initialize its weights using *orthogonal initialization* [ACB19], which helps avoid exploding gradients. In red, we show the input/output dimensions of each layer; in orange, we show attention modules; in blue, convolutional blocks and layers; in green, upsampling and concatenating operations; in yellow, normalization layers; and in purple, non-linearities.

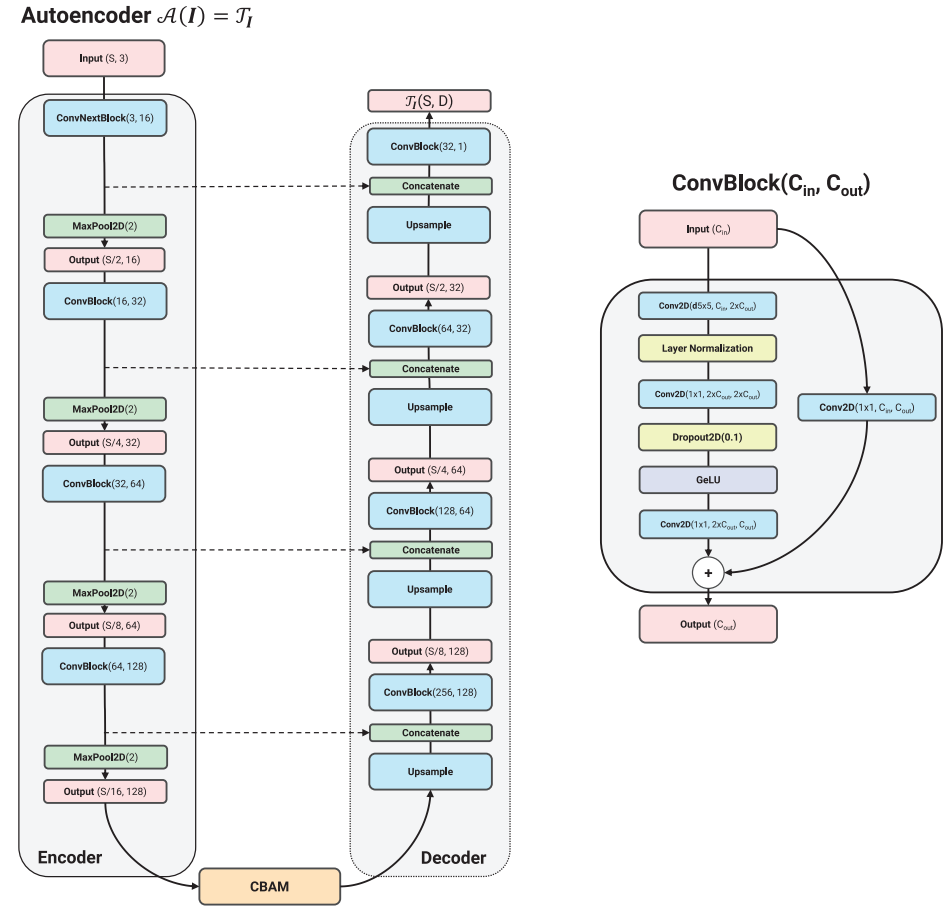


Figure 4.11: A full diagram of our sinusoidal renderer R. We build upon a SIREN architecture [Sit+20], which takes as input the latent vector, queried from the input texture, as well as target camera and light angles to re-render. We introduce Layer Normalization after each linear layer, which we implement using 1×1 convolutions to allow for 2-D inputs during training. The output of the model is clamped to avoid negative output radiance values, using a simple ReLU non-linearity [NH10], following [Kuz+21].