Real-time Neural Woven Fabric Rendering

Xiang Chen
School of Software, Shandong University
China
xiang_chen@mail.sdu.edu.cn

Lu Wang*
School of Software, Shandong University
China
luwang_hcivr@sdu.edu.cn

Beibei Wang*
School of Intelligence Science and Technology, Nanjing University
China
beibei.wang@nju.edu.cn

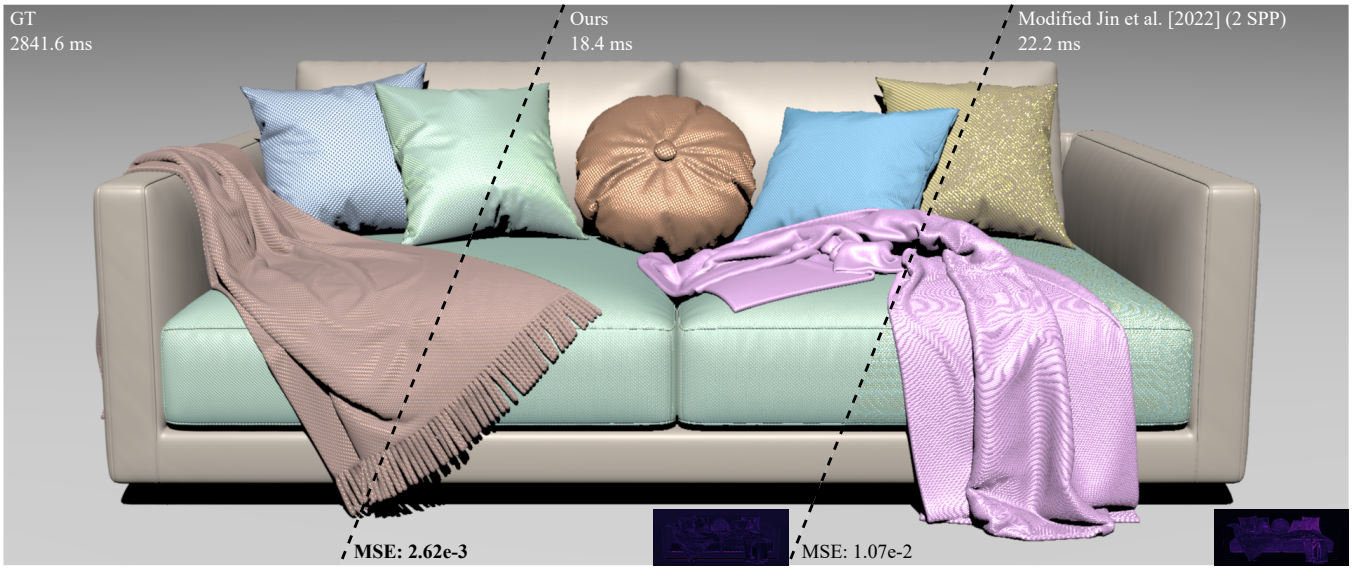

Figure 1: We present a neural network for real-time woven fabric rendering. In this Sofa scene, we provide eight different patterns of woven fabrics. Our method can represent several typical types of woven fabrics with a single neural network, which achieves a fast speed and a quality close to the ground truth

.

ABSTRACT

Woven fabrics are widely used in applications of realistic rendering, where real-time capability is also essential. However, rendering realistic woven fabrics in real time is challenging due to their complex structure and optical appearance, which cause aliasing and noise without many samples. The core of this issue is a multi-scale representation of the fabric shading model, which allows for a fast range query. Some previous neural methods deal with the issue at the cost of training on each material, which limits their practicality. In this paper, we propose a lightweight neural network to represent different types of woven fabrics at different scales. Thanks to the regularity and repetitiveness of woven fabric patterns, our network can encode fabric patterns and parameters as a small latent vector, which is later interpreted by a small decoder, enabling the representation of different types of fabrics. By applying the pixel's footprint as input, our network achieves multi-scale representation. Moreover, our network is fast and occupies little storage because of its lightweight structure. As a result, our method achieves rendering and editing woven fabrics at nearly 60 frames per second on an RTX 3090, showing a quality close to the ground truth and being free from visible aliasing and noise.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**.

KEYWORDS

fabric rendering, neural rendering

ACM Reference Format:
Xiang Chen, Lu Wang, and Beibei Wang. 2024. Real-time Neural Woven Fabric Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27-August 1, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3641519.3657496

*Corresponding authors.

SIGGRAPH Conference Papers '24, July 27-August 1, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07…$15.00
https://doi.org/10.1145/3641519.3657496

1 INTRODUCTION

Realistic woven fabric rendering is vital to many applications in the modern digital age, including virtual reality, video games, and digital humans. Besides the realism, these applications also require a real-time rendering performance. However, simultaneously achieving realism and low time costs is still challenging, as fabrics have complex microstructures and strong anisotropic appearances, leading to aliasing if under-sampled.

Existing approaches can produce high-fidelity rendering results by modeling woven fabrics with volume representations [Heitz et al. 2015; Zhao et al. 2011] or modeling the scattering function of each fiber [Aliaga et al. 2017; Khungurn et al. 2016]. Despite their high realism, they can not be applied to real-time rendering. On the other hand, it is more practical to model the woven fabrics with surface shading models [Irawan and Marschner 2012; Jin et al. 2022; Zhu et al. 2023a]. Nevertheless, these methods cannot be applied directly to real-time applications since they need hundreds of samples to avoid aliasing caused by detailed yarn structures. The key to this issue is a multi-scale representation of the fabric surface shading model, which can not be treated linearly as the mipmap of diffuse maps [Williams 1983]. The other line of work resorts to neural networks to create a multi-resolution representation for general materials [Gauthier et al. 2022; Kuznetsov et al. 2021], which are not specialized for fabrics. Despite their high performance, these methods either need per-material training [Kuznetsov et al. 2021] or have difficulties supporting fabric parameters [Gauthier et al. 2022].

In this paper, we propose to represent different types of woven fabrics at different scales with a single neural network. At the core of our method is a tiny neural network capable of representing different types of fabrics while simultaneously allowing real-time rendering and editing. The key to achieving these two goals is based on an important observation: woven fabric patterns have regularity, different from the general materials. More specifically, starting from the state-of-the-art fabric surface shading model [Zhu et al. 2023a], we design a lightweight neural network for the range query of fabric bidirectional scattering distribution functions (BSDFs). Our network first encodes the fabric patterns into features and then fuses the pattern feature with other fabric parameters into a small material latent vector. The material latent vector, together with the queries (position, range, and directions), can be interpreted by a decoder consisting of two multilayer perceptrons (MLPs) into BSDF values. Eventually, the lightweight network structure enables real-time rendering at a frame rate of almost 60 frames per second on an RTX 3090, with a quality close to the ground truth for several typical types of woven fabrics. The network only occupies 5 MB of storage. Furthermore, our network also supports real-time material editing by modifying the fabric patterns and parameters.

2 PREVIOUS WORK

In this section, we briefly review previous works related to fabric surface shading models and prefiltering methods for appearance models. As our method is less related to the models based on yarn [Zhu et al. 2023b], ply [Montazeri et al. 2020] or fiber [Aliaga et al. 2017; Khungurn et al. 2016] curves, we do not discuss these works in this section. The volume-based models [Heitz et al. 2015; Zhao et al. 2011] are also beyond our scope.

Fabric surface models. Fabric surface models [Adabala et al. 2003; Irawan and Marschner 2012; Sadeghi et al. 2013] have been introduced to fabric rendering for decades. The recent SpongeCake surface shading model by Wang et al. [2022] can model the fabric appearance by stacking different layers, where each layer is defined by a microflake distribution [Heitz et al. 2015; Jakob et al. 2010]. On top of the SpongeCake model, Jin et al. [2022] design extra components specialized for fabrics, including a blended Lambertian term and two types of noise. Later, their method is improved by Zhu et al. [2023a] to deal with transmission and shadowing-masking effects.

While the above surface shading models can render fabrics efficiently, they are unsuitable for real-time rendering due to the lack of a multi-scale representation. Thus, we propose a neural multi-scale representation that considers range queries and suits real-time rendering. Our method can exploit most of the above models in theory, and we use the modification of Jin et al. [2022] (including transmission and shadowing-masking effects) as an example.

Complex appearance prefiltering. Due to the limited time budget, prefiltering or downsampling complex appearance models is a practical solution for real-time rendering applications. It has been used for filtering normal maps [Dupuy et al. 2013; Olano and Baker 2010], filtering the normal distribution function in the slope domain [Kaplanyan et al. 2016], or joint prefiltering [Xu et al. 2017] the normal map together with bidirectional reflectance distribution function (BRDF). While these approaches produce reasonable results for surface-like distributions, they are unsuitable for fiber-like distributions. Unlike the above methods, Wu et al. [2019] introduce a more accurate way to filter displacement maps and BRDFs, using a scaling function for both surface location and direction, at the cost of expensive rendering time. Prefiltering has also been applied to microflake volumes [Heitz et al. 2015; Loubet and Neyret 2017; Zhao et al. 2016]. One related work by Heitz et al. [2015] performs mipmapping on the matrix which defines the microflake. Despite its efficiency, their method produces less satisfying results, especially for low-roughness fabrics.

Recently, neural-based approaches have been proposed for multi-scale appearance representation. NeuMIP [Kuznetsov et al. 2021] prefilters the neural textures with a latent texture mipmap. A similar prefiltering idea has been used by Zeltner et al. [2024], except they encode material parameters into latent vectors. These methods can provide high-quality rendering results and acceptable time costs due to their lightweight network structure. However, they must be trained per material, which costs extra time and restricts their practicality. Unlike over-fitted neural representations, MIPNet [Gauthier et al. 2022] produces mip-mapped textures of spatially varying BRDFs (SVBRDFs). MIPNet requires a tensor-based reformulation of specific shading models, making it less flexible to support other models.

3 BACKGROUND AND MOTIVATION

Woven fabrics are made of weft and warp yarns, which are interlaced in a particular pattern. Each yarn is built by twisting plies,

where each ply aggregates a set of fibers. In this paper, we focus on fabrics with a single ply for each yarn to demonstrate the effectiveness of our network, following previous work [Jin et al. 2022]. These different levels of structures can be modeled in several ways: volume, curve, or surface, where both volume and curve-based models are too heavy for real-time rendering. Hence, we opt for surface models. In this section, we recap the related fabric surface models that are close to our method. Later, we show the motivation and formulation of our method.

3.1 Preliminaries

Fabric surface model. Recently, Jin et al. [2022] proposed a surface model for woven fabrics, including the geometric and shading models. In their geometric model, each yarn is expressed as a curved cylinder, defined by the twist and inclination angle of the yarns, and the width and length of the yarns. These parameters with the underlying yarn pattern (Fig. 1 in the supplementary) establish the normal and tangent of yarns, forming microstructures on the cloth surface. Their shading model f_r consists of a specular term f_r^s and a diffuse term f_r^d, where the former is based on the SpongeCake model [Wang et al. 2022] with a fiber-like microflake phase function, and the latter is a blended Lambertian term considering both macro-surface normal n_s and the ply normal n_p:

$$\begin{aligned} f_\mathrm{r}(\omega_\mathrm{i},\omega_\mathrm{o}) &= f_\mathrm{r}^\mathrm{s}(\omega_\mathrm{i},\omega_\mathrm{o}) + f_\mathrm{r}^\mathrm{d}(\omega_\mathrm{i},\omega_\mathrm{o}),\\ f_\mathrm{r}^\mathrm{s}(\omega_\mathrm{i},\omega_\mathrm{o}) &= \frac{k_\mathrm{s} D(h) G(\omega_\mathrm{i},\omega_\mathrm{o})}{4\cos\omega_\mathrm{i}\cdot\cos\omega_\mathrm{o}},\\ f_\mathrm{r}^\mathrm{d}(\omega_\mathrm{i},\omega_\mathrm{o}) &= w\frac{k_\mathrm{d}\langle\omega_\mathrm{i}\cdot n_\mathrm{p}\rangle}{\pi\langle\omega_\mathrm{i}\cdot n_\mathrm{s}\rangle} + (1-w)\frac{k_\mathrm{d}}{\pi}, \end{aligned} \tag{1}$$

where ω_i and ω_o are the incoming and outgoing directions, respectively. k_d and k_s are the diffuse and specular albedo respectively, and w is the weight for blending the two Lambertian terms. The specular term includes the distribution D on the half vector h between ω_i and ω_o, and the attenuation G along the media with density ρ and thickness T. They are defined as:

$$\begin{aligned} D(h) &= \frac{1}{\pi\alpha q^2},\ \text{where } q = h^\top S^{-1} h,\\ G(\omega_\mathrm{i},\omega_\mathrm{o}) &= \frac{1-e^{-T\rho(\Lambda(\omega_\mathrm{i})+\Lambda(\omega_\mathrm{o}))}}{\Lambda(\omega_\mathrm{i})+\Lambda(\omega_\mathrm{o})},\\ \Lambda(\omega) &= \frac{\sigma(\omega)}{\cos\omega},\ \text{where } \sigma(\omega) = \sqrt{\omega^\top S\omega}, \end{aligned} \tag{2}$$

where α represents roughness and S is a symmetric, positive definite 3×3 matrix. In detail, $S = \mathrm{diag}(1, 1, \alpha^2)$ is for a microflake oriented along the ply direction, and other orientations can be achieved by defining $S' = R^\top S R$ for any 3×3 rotation matrix R. σ is the projected area, and Λ is the Smith shadowing-masking function. In practice, $T\rho$ is always assigned a value of 2 for fabrics. The height field scaling factor β is introduced to adjust the height field of plies, which affects both the normal n_p and orientation t_p. All the shading model parameters are summarized in Table 1.

Later, Zhu et al. [2023a] improve the model above by introducing a bidirectional transmission distribution function (BTDF) and the shadowing-masking effects between the yarns, detailed in the supplementary.

Table 1: Parameters in the shading model of fabric.

Parameter	Definition
n_p	ply normal
t_p	ply orientation
x_p	ply position
k_d ($k_\mathrm{d}^\mathrm{warp}$, $k_\mathrm{d}^\mathrm{weft}$)	diffuse albedo (for warp or weft yarns)
k_s ($k_\mathrm{s}^\mathrm{warp}$, $k_\mathrm{s}^\mathrm{weft}$)	specular albedo (for warp or weft yarns)
α (α^warp, α^weft)	roughness (for warp or weft yarns)
β (β^warp, β^weft)	height field scaling (for warp or weft yarns)

Fabric surface BSDF aggregation. The woven fabric surface is made of microstructures, which are smaller than a single pixel, leading to a sub-pixel appearance. These micro-structures are captured by sampling the rays within a pixel. Unfortunately, it is impossible to have a high sample rate in real-time rendering. Therefore, the core of real-time fabric rendering is a multi-scale representation which allows efficient aggregation. Zhu et al. [2023a] propose an aggregation BSDF defined on a patch $\mathcal{P}$ seen from a pixel, which includes visibility terms to enable shadowing-masking effects between yarns:

$$\begin{aligned} f_\mathcal{P}(\omega_\mathrm{i},\omega_\mathrm{o}) = \frac{1}{A_\mathcal{P}(\omega_\mathrm{o})}\int_\mathcal{P} & f_p(\omega_\mathrm{i},\omega_\mathrm{o})\langle\omega_\mathrm{i}\cdot n_\mathrm{p}(p)\rangle \\ & V(x_\mathrm{p}(p),\omega_\mathrm{i})A(p,\omega_\mathrm{o})k_\mathcal{P}(p)\mathrm{d}p, \end{aligned} \tag{3}$$

where p is a point within the patch $\mathcal{P}$. A kernel $k_\mathcal{P}$ is defined as $\int_\mathcal{P} k_\mathcal{P}(p)\mathrm{d}p = 1$ to normalize the area. $V(x_\mathrm{p},\omega)$ is the binary visibility function of position x_p in direction ω and $A(p,\omega_\mathrm{o})$ is the visible projected area along ω_o defined as:

$$A(p,\omega_\mathrm{o}) = \frac{\langle\omega_\mathrm{o}\cdot n_\mathrm{p}(p)\rangle}{\langle n_\mathrm{s}\cdot n_\mathrm{p}(p)\rangle}V(x_\mathrm{p}(p),\omega_\mathrm{o}), \tag{4}$$

where n_s is surface normal and $\frac{1}{\langle n_\mathrm{s}\cdot n_\mathrm{p}(p)\rangle}$ is the Jacobian $|\frac{\mathrm{d}x_\mathrm{p}(p)}{\mathrm{d}p}|$. $A_\mathcal{P}(\omega_\mathrm{o})$ is the total visible projected area in patch $\mathcal{P}$ along ω_o:

$$A_\mathcal{P}(\omega_\mathrm{o}) = \int_\mathcal{P}\frac{\langle\omega_\mathrm{o}\cdot n_\mathrm{p}(p)\rangle}{\langle n_\mathrm{s}\cdot n_\mathrm{p}(p)\rangle}V(x_\mathrm{p}(p),\omega_\mathrm{o})k_\mathcal{P}(p)\mathrm{d}p = \frac{\langle\omega_\mathrm{o}\cdot n_\mathrm{f}(\mathcal{P})\rangle}{\langle n_\mathrm{s}\cdot n_\mathrm{f}(\mathcal{P})\rangle}, \tag{5}$$

where $n_\mathrm{f}(\mathcal{P})$ is the average visible micro-scale normal over the patch $\mathcal{P}$.

3.2 Motivation

Zhu et al. [2023a] evaluate the integral of Eqn. (3) in the Monte-Carlo manner, leading to variance when under-sampled and preventing it from being used for real-time rendering. Therefore, the core problem is to solve Eqn. (3) efficiently on fabric materials to enable real-time rendering and editing. The function defined by Eqn. (3) is high-dimensional, which includes the geometry and appearance parameters. One straightforward way to represent such a high-dimension function is to use a neural network. This neural network has to meet the following requirements: *fast inference*, *capability of representing multiple woven fabrics*, and *editability*. The fast inference allows real-time rendering; the ability to represent multiple typical woven fabrics with a single neural network which

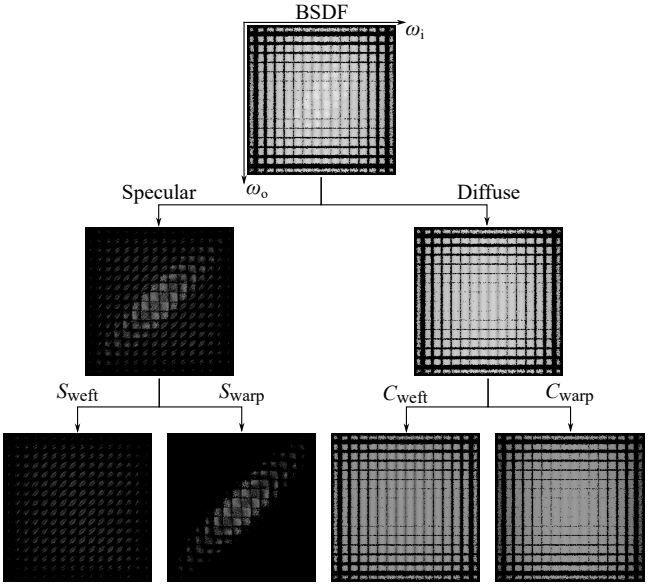

Figure 2: Separation of BSDF distributions. We separate the BSDF value into several components, considering specular/diffuse and yarn type (weft/warp). Thanks to this separation, the distributions become much simpler. The BSDF or components are visualized by mapping the incoming directions to the horizontal axis and the outgoing direction to the vertical axis. Here, we use a plain pattern with white color and compute its BSDF/component values by Monte-Carlo point sampling within a patch with the query size set as 205×205.

avoids per-material training; and the editability enables material editing.

The main challenge is designing a network to meet these requirements simultaneously. Fast inference requires a lightweight network; however, a lightweight network usually has limited representation ability. Most lightweight neural networks are used for per-material representation [Kuznetsov et al. 2021; Zeltner et al. 2024]. To this end, our key insight is the characteristics of woven fabrics, which have regular and repetitive patterns. For editability, we procedurally represent the materials with parameters rather than only using spatially-varying maps [Zhu et al. 2023a]. In this way, the material has more flexibility for editing.

4 METHOD

In this paper, we first reformulate the fabric aggregation shading model (Sec. 4.1). Then, we design a neural network to represent the shading model (Sec. 4.2). Finally, we apply the neural network for real-time rendering and material editing (Sec. 4.3).

4.1 Reformulation of fabric shading model

The aggregation form of the fabric shading model is shown in Eqn. (3), which is essentially a mapping from the geometry and appearance parameters to a BSDF value, given a query of incoming/outgoing directions and a patch $(\omega_i, \omega_o, \mathcal{P})$:

$$\{n_p, t_p, x_p, \alpha, \beta, k_d, k_s, \omega_i, \omega_o, \mathcal{P}\} \rightarrow f_{\mathcal{P}}(\omega_i, \omega_o), \tag{6}$$

where n_p, t_p, x_p are geometry parameters and α, β, k_d, k_s are appearance parameters defined in Table 1. A straightforward way to represent this mapping is using a neural network directly. However, this mapping mixes several distributions: the specular/diffuse distribution and the warp/weft distribution. As these different distributions have high variance, representing such mapping becomes difficult for a lightweight network. Therefore, we separate the mapping function into different components:

$$\{n_p, t_p, \alpha, \beta, \omega_i, \omega_o, \mathcal{P}\} \rightarrow \{C_{warp}, C_{weft}, S_{warp}, S_{weft}\}, \tag{7}$$

where $C_{warp}, C_{weft}, S_{warp}, S_{weft}$ represent the diffuse or specular term for warp or weft yarn respectively. We remove k_d and k_s from the input, as they depend on the yarn types (weft or warp) only. We also remove x_p, as it has been implicitly expressed by normal n_p and orientation t_p. The final shading function $f_{\mathcal{P}}$ is defined as:

$$f_{\mathcal{P}}(\omega_i, \omega_o) = k_d^{warp} C_{warp} + k_d^{weft} C_{weft} + k_s^{warp} S_{warp} + k_s^{weft} S_{weft}. \tag{8}$$

The final formulation is a mapping from geometry parameters (n_p, t_p), appearance parameters (α, β) and a query $(\omega_i, \omega_o, \mathcal{P})$ to four values $(C_{warp}, C_{weft}, S_{warp}, S_{weft})$. With this new formulation, the mapping has a much simpler distribution, as shown in Fig. 2.

4.2 Neural representation

With the reformulated aggregated shading model, we now design a neural network to represent the mapping. As we discussed previously, there are three types of inputs: geometry, appearance, and query parameters. The geometry and appearance parameters establish a material, and the query on the same material should be consistent. To this end, we leverage a typical encoder-decoder structure for the aggregated shading model, where the encoder compresses the geometry and appearance parameters into a material latent vector z and the decoder interprets the latent vector into the different components $(C_{warp}, C_{weft}, S_{warp}, S_{weft})$, given a query $(\omega_i, \omega_o, \mathcal{P})$. The network structure is shown in Fig. 3.

Encoder. The encoder compresses the geometry and appearance parameters into a latent vector z. The key to designing this encoder is fusing these two types of parameters. We take two types of inputs: geometry textures for the pattern and the procedural appearance parameters for the others. We first encode textures (normal map and orientation map) into features with ResNet [He et al. 2016]. In practice, we use a modified ResNet-18 without the last residual block, as it does not improve the quality while leading to longer training and inference time. Then, we feed the feature together with the appearance parameters into a residual block, which consists of two fully connected layers of size 128, and output a material latent vector of size 64.

Decoder. With the material latent vector, we decode it into four components given a query. In the query, we have the spatial inputs (shading position and query range of $\mathcal{P}$) and the angular inputs (light and view directions). The naive way to handle these inputs is concatenating all of them together and interpreting them by a decoder at the same time. However, it does not work well since

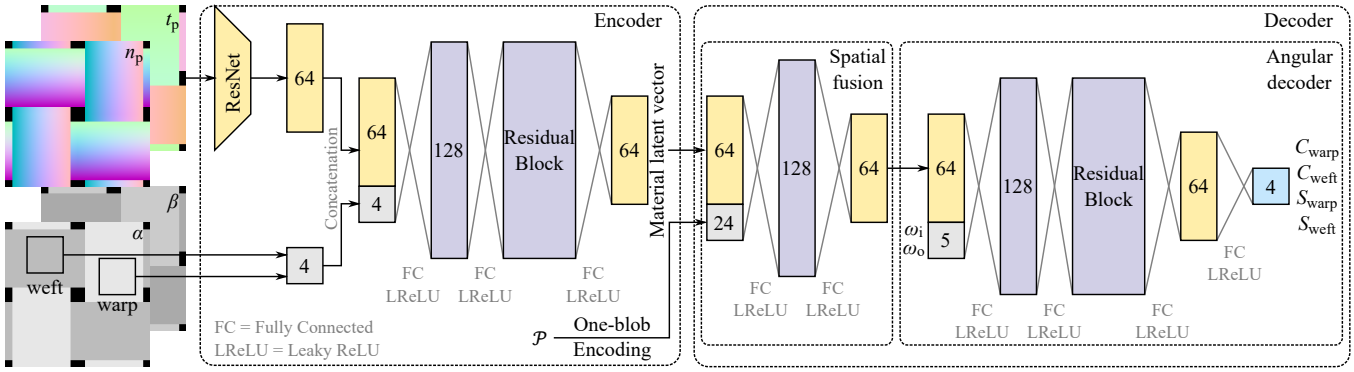

Figure 3: The structure of our neural network. Our network consists of an encoder and a decoder. The encoder takes the input of fabric patterns (normal and orientation textures) and parameters (roughness and height field scaling). These inputs are encoded into a material latent vector by the encoder, which consists of a modified ResNet [He et al. 2016] and an MLP. Then, the material latent vector is fused with the spatial query ($\mathcal{P}$) first, concatenated with the angular inputs (ω_i, ω_o) and then fed to the angular decoder to get four components. Both the encoder and decoder include a residual block composed of two fully connected layers and a skip connection before the last leaky ReLU function.

the spatial parameters dominate the main distribution. Also, the output value must be consistent with varying angular parameters given specific spatial parameters. Therefore, we fuse the material latent vector with the spatial parameters and then concatenate it with the angular parameters. In this way, a small decoder is enough to achieve multi-scale representation. In practice, we apply one-blob encoding [Müller et al. 2021] to the position and size of $\mathcal{P}$, extending them from 3 to 24 channels. This encoding aids the network in distinguishing the spatial inputs better. Then, we use two fully connected layers to fuse the material latent vector with the spatial parameters to get a spatial feature. Finally, the spatial feature, together with the angular inputs, is interpreted by a decoder—*angular decoder* to get the final four BSDF components, where the angular decoder consists of five fully connected layers with one skip connection. Note that the angular inputs consist of three channels (x, y, z) for the light direction and two channels (x, y) for the view direction. The z component for the light direction is needed, as it might be negative for transmission.

Loss functions. We compute the ground-truth values of the four terms ($C_{\text{warp}}, C_{\text{weft}}, S_{\text{warp}}, S_{\text{weft}}$) by sampling the integral in Eqn. (3), and then we introduce a specular loss and a diffuse loss.

Regarding the specular loss, we find that the distribution of specular has a wide range. Therefore, similar to NeuMIP [Kuznetsov et al. 2021], we perform a mapping $g(x) = \ln(kx+1)$ on the ground truth values, and then use the mean squared error (MSE) loss:

$$Loss_S(f_S^{\text{pred}}, f_S^{\text{gt}}) = \frac{1}{N}\sum_{i=1}^{N}\left(f_S^{\text{pred}} - g(f_S^{\text{gt}})\right)^2, \tag{9}$$

where f_S^{pred} and f_S^{gt} represent the network output and the target, respectively, for both S_{warp} and S_{weft}. N denotes the number of corresponding terms in a training batch. In practice, we find $k = 100$ for BRDF and $k = 1000$ for BTDF are good choices.

For the diffuse terms C_{warp} and C_{weft}, we also use the MSE loss:

$$Loss_C(f_C^{\text{pred}}, f_C^{\text{gt}}) = \frac{1}{N}\sum_{i=1}^{N}\left(f_C^{\text{pred}} - f_C^{\text{gt}}\right)^2. \tag{10}$$

Combining the specular and diffuse loss functions together leads to our final loss:

$$\begin{aligned} Loss(f^{\text{pred}}, f^{\text{gt}}) = &\sum_{I}\lambda_S Loss_S^i(f_{S_i}^{\text{pred}}, f_{S_i}^{\text{gt}}) \\ &+ \sum_{I}\lambda_C Loss_C^i(f_{C_i}^{\text{pred}}, f_{C_i}^{\text{gt}}), \end{aligned} \tag{11}$$

where i is one of the yarn types I (warp or weft). λ_S and λ_C are the hyperparameters, set as 0.4 and 0.1 in practice.

4.3 Rendering and editing

Now, we use our neural network for real-time rendering and material editing. In practice, we integrate our network into Falcor [Kallweit et al. 2022], and adopt NVIDIA TensorRT for inference.

Rendering. Before rendering, for all the fabric materials in the scene, we get their material latent vectors with our encoder. During rendering, we shoot a ray to get a shading point at each pixel. Then, we prepare the query ($\omega_i, \omega_o, \mathcal{P}$) and compute the lighting. Next, we infer the decoder with the query and the material latent vector to get C_{warp}, C_{weft}, S_{warp} and S_{weft}. Finally, we compute the BSDF value with these four components and multiply it by the lighting to get the final rendering result. Note that only the decoder needs to be evaluated during rendering.

Editing. Our method allows editing of the following parameters: albedos (k_d, k_s), roughness, height field scaling factor, and fabric patterns. To edit albedos, we only need to change color textures, and no encoder inference is needed, as they are not coupled with the network. To edit the other parameters, we need to infer the encoder to update the material latent vector, which costs about 0.5

Table 2: Distributions to sample the parameter space of our model. $\mathcal{U}(x, y)$ represents a continuous uniform distribution in the interval (x, y). $\mathcal{V}(X)$ is a discrete uniform random variable on a finite set X.

Parameter	Sampling Function
yarn pattern	$W = \mathcal{V}(\{0, 1, 2, 3, 4, 5, 6\})$
twist angle	$\psi = \mathcal{V}(\{-30, 0, 30\})$
inclination angle	$u = \mathcal{U}(15, 45)$
roughness	$\alpha = \mathcal{U}(0.1, 1)$
height field scaling	$\beta = \mathcal{U}(0, 2)$
footprint position	$p = (\mathcal{U}(0, 1), \mathcal{U}(0, 1))$
footprint size	$s = \mathcal{U}(\mathcal{V}(\{(0, 1), (1, 5)\}))$
incoming direction	$\omega_\mathrm{i} = (\mathcal{U}(-1, 1), \mathcal{U}(-1, 1))$
outgoing direction	$\omega_\mathrm{o} = (\mathcal{U}(-1, 1), \mathcal{U}(-1, 1))$

ms. After getting the material latent vector, the following steps are the same as rendering. Although the encoder needs to be inferred during editing, it costs negligible time. Thus, our method supports real-time editing of fabric materials.

5 IMPLEMENTATION DETAILS

5.1 Data preparation

To generate the synthetic training dataset, we modify the shading model by Jin et al. [2022], including the transmission and the shadowing-masking effects between yarns within a fixed-size patch. The main difference from Zhu et al. [2023a] is that their model includes a delta transmission and computes the shadowing-masking term within the pixel's footprint. Note that our network is compatible with most woven fabric surface shading model, and we choose the current shading model for simplicity.

We sample parameters to generate the dataset with the sampling functions shown in Table 2. Specifically, we choose seven fabric patterns, including plain, twill 3×3, twill 5×5, twill 8×8, satin 5×5, satin 8×8 and satin 5×10. The gap size is set as 0.2, which means the gap takes up 20% of the yarn. For each pattern, we generate different sets (9 for the plain pattern and 4 for others) of normal and orientation textures by sampling the twist angle ψ and the inclination angle u. Then, we uniformly sample the roughness and height field scaling factor, producing 9 sets for the plain pattern and 16 sets for other patterns. In total, we generate 81 plain materials, 192 twill materials, and 192 satin materials.

For each material, we generate queries by sampling the footprints and the incoming/outgoing directions. For the footprint center, we partition the pattern texture into 8×8 grids and perform uniform sampling at each grid. Then, for each footprint center, we generate different footprint sizes by uniformly locating 10 samples in the range $[0, L_t]$ and another 10 samples in the range $[L_t, 5L_t]$, where L_t is the pattern texture size. Then, we sample the ω_i and ω_o by stratified sampling the x and y components of both directions in a 8×8 grid and computing the z component for both BRDF and BTDF. As a result, we have around 2500 valid pairs of directions for each footprint, leading to about 3.2 million queries per material. After establishing the materials and queries, we compute their BSDF components (C_warp, C_weft, S_warp and S_weft) using 2048 samples to avoid noise. The time cost of data generation is about 22 minutes per material and 165 hours in total.

5.2 Training details

Our network is implemented in the PyTorch framework, using the SGD optimizer. In practice, we use a learning rate of 5×10^{-2} for all the weights, which decays by 0.2 at the sixth and ninth epoch, and we also add a regularization to all the weights. During training, materials are randomly chosen for each iteration. We uniformly choose about 160 million queries out of the entire data as the training dataset and organize 512 queries as one batch. We train the network for 10 epochs, which took about 4 days on a single NVIDIA RTX 3090 GPU.

6 RESULT

In this section, we evaluate our method and compare it against previous works. We take 256 samples per pixel (SPP) to get the converged results of the modified Jin et al. [2022] as the ground truth (GT). In our experiments, we only consider the direct illumination from a point or directional light source.

All the given timings are total time costs measured on a GeForce RTX 3090 GPU, except for the time costs of the BSDF evaluation in Table 2 (supplementary). We use MSE to measure the difference between each method and the ground truth, and visualize the differences with ꟻLIP [Andersson et al. 2021]. The resolutions of all the results are set as 1920×1080.

6.1 Comparison against previous work

In this section, we make comparisons with modified Jin et al. [2022] (taken as baseline) and NeuMIP [Kuznetsov et al. 2021]. For fairness, we use the same-sized decoder for NeuMIP as our angular decoder, and compare our method with NeuMIP only on seen BRDF materials, since NeuMIP must be trained per material and does not support BTDFs. In Fig. 4, we compare our method with the baseline (1 SPP), the baseline (equal time) and NeuMIP (1 SPP). By comparison, we find that the baseline (1 SPP) produces results with apparent aliasing, even with a higher sample rate. In contrast, our results are free from aliasing and closer to the ground truth. As for NeuMIP, it can produce results with a higher quality than ours in terms of MSE, at the cost of a higher storage and per-material training. In particular, the storage of our network is less than 5 MB, while the storage of NeuMIP is over 50 MB for each neural material. As NeuMIP is trained per material, its storage increases with the number of materials in the scene, while the storage of our method keeps constant. Furthermore, our network can be applied to three typical types of woven fabrics once trained, but NeuMIP have to be trained per material which takes a long time.

In Fig. 1, we compare our method with the baseline (2 SPP) on a complex scene with multiple fabrics. The result of the baseline has noticeable aliasing, while our result is smooth and closer to the ground truth. Meanwhile, our result has lower MSE than theirs. We provide more comparisons in Fig. 5 for BRDF and Fig. 6 for BTDF on unseen materials. In both scenes, our method performs better than the baseline with 1 SPP in terms of MSE. For equal-time comparison, our results have much less noise and aliasing for both BRDF and

Table 3: Time cost of our method for each scene in microsecond. "Others" refers to the non-neural steps (ray intersection, data transmission and scene shading).

Scene	Inference	Others
Single Cloth (Fig. 4)	14.3	2.9
Bowl Cloth (Fig. 5)	13.9	4.0
Lamp (Fig. 6)	14.4	2.9
Sofa (Fig. 1)	13.6	4.8

BTDF, although our results have higher MSE occasionally. More results of unseen materials are provided in the supplementary.

6.2 Ablation study

Encoding of footprint. We apply one-blob encoding [Müller et al. 2021] to the query footprint's position and size. We validate its impact in Fig. 7, by comparing the results with and without the one-blob encoding. By comparison, we find that the result with encoding has lower MSE, less artifacts, and a closer specular shape to the GT. This proves that the encoding improves performance.

Spatial fusion. In our decoder, we perform the spatial query by fusing the material latent vector with the footprint and then perform the angular query. We validate the impact of this spatial fusion in Fig. 7, by comparing the results of our network with and without the spatial fusion. For fairness, we increase the number of hidden layers in the angular decoder when removing the spatial fusion. We find that the result without spatial fusion shows obvious artifacts and higher MSE.

6.3 Interpolation and editing

Interpolation. The material latent vectors produced by the encoder form a material latent space. We study the behavior of the material latent space by interpolating or extrapolating between two given material latent vectors using parameters as the weights. In practice, we investigate the interpolation and extrapolation of two material latent vectors using the roughness (left) and height field scaling factor (right) as the weights in Fig. 8. Here, two given material latent vectors are obtained by performing the encoder with the parameter (roughness/height field scaling factor) set as 0.2 and 0.8. Then, the material latent vectors at 0.3 or 0.6 are computed by interpolation, while the material latent vector at 0.9 is computed by extrapolation. We compare the results of these interpolated/extrapolated material latent vectors (bottom) with the ones generated by the encoder directly (top). The results show that the interpolation/extrapolation of material latent vectors can produce a smooth transition and match the results rendered with the directly encoded material latent vectors.

Editing. In Fig. 9, we show the results of material parameter editing, including roughness, height field scaling factor, pattern and albedos. Our method can produce various rendering results when editing these parameters. In particular, our method is able to represent spatial-varying appearances when applying albedo maps, by computing k_d and k_s via mipmapping. In the supplementary video, we also show the efficiency of our material editing.

6.4 Performance analysis

In Table 3, we analyze the run-time performance of our method and show the breakdown cost of neural and non-neural steps. The neural step is the inference of our decoder, and the non-neural refers to the steps without the network (ray intersection, data transmission, and shading). We do not include the time cost of our encoder, as it is only inferred when initializing or editing materials, which takes only about 0.5 ms. As shown in the table, our method can achieve real-time rendering. Moreover, as shown in the supplementary video, our method keeps a stable frame rate at different scales when zooming in and out.

6.5 Discussion and limitations

Our real-time neural network can represent multi-scale appearances of multiple woven fabrics, which avoids training the network per material. However, our method still has some limitations.

Evaluation time. We use TensorRT to deploy our neural network, and the efficiency is optimized automatically. Other techniques could be used to accelerate its speed further. One solution is rewriting it by Slang as Zeltner et al. [2024]. Another option is using a small NeuMIP-style texture pyramid of latent codes to make the decoder smaller.

Network representing ability. Similar to NeuMIP [Kuznetsov et al. 2021], our representing ability is still limited by the network. For example, both NeuMIP and our method can not handle very sharp variations, as shown in the twill results of Fig. 4.

Importance sampling. Our paper mainly focuses on BSDF evaluation without considering the importance sampling. To enable importance sampling, one solution is predicting the outgoing direction and its probability density function with a neural network, following Zeltner et al. [2024].

Energy conservation. Similar to prior works [Zeltner et al. 2024], our method cannot guarantee energy conservation due to the bias of the neural representation. That is to say, more energy might be added, although we have not observed any noticeable issues.

Capability of our fabric model. We exclude the delta transmission and procedural perturbation in our model, which restricts the realism of the fabric appearance. It is possible to include the delta transmission by getting the gap ratio for a given query with the neural network and then compute the delta transmission through the gap ratio. For the procedural perturbation, combining our method with some real-time procedural glints approaches may be one solution.

7 CONCLUSION AND FUTURE WORK

In this paper, we present a lightweight neural network to represent multiple woven fabric materials at multiple scales. Our key observation is the regular and repetitive characteristics of woven fabric patterns, which enables the possibility of representing multiple materials with a compact latent space. Specifically, we introduce a simple encoder-decoder structure, where the encoder compresses the fabric pattern and other parameters into a material latent vector, and the decoder interprets the material latent vector with a

spatial fusion component and a small angular decoder. Thanks to the lightweight encoder and decoder, our network is able to achieve real-time rendering and editing. Meanwhile, our network only occupies a small storage of 5 MB, even for scenes with multiple fabric materials.

In the future, there are still many potential research directions. Currently, our network supports three typical woven fabric types. More types of woven fabrics and knitted fabrics are still missing from our representation. The other types of woven fabrics can be handled by expanding our dataset. However, representing knitted fabrics is difficult, as their structures significantly differ from woven fabrics. We leave it for future work. It is also worth exploring ways to enable rendering fabrics with spatially-varying patterns.

ACKNOWLEDGMENTS

We thank the reviewers for the valuable comments. This work has been partially supported by the National Science and Technology Major Project under grant No. 2022ZD0116305 and National Natural Science Foundation of China under grant No. 62272275 and 62172220.

REFERENCES

Neeharika Adabala, Nadia Magnenat-Thalmann, and Guangzheng Fei. 2003. Real-time rendering of woven clothes. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '03)*. Association for Computing Machinery, New York, NY, USA, 41–47. https://doi.org/10.1145/1008653.1008663

Carlos Aliaga, Carlos Castillo, Diego Gutierrez, Miguel A. Otaduy, Jorge Lopez-Moreno, and Adrian Jarabo. 2017. An Appearance Model for Textile Fibers. *Computer Graphics Forum* 36, 4 (2017), 35–45. https://doi.org/10.1111/cgf.13222

Pontus Andersson, Jim Nilsson, Peter Shirley, and Tomas Akenine-Möller. 2021. Visualizing Errors in Rendered High Dynamic Range Images. In *Eurographics Short Papers*. https://doi.org/10.2312/egs.20211015

Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. 2013. Linear Efficient Antialiased Displacement and Reflectance Mapping. *ACM Trans. Graph.* 32, 6, Article 211 (nov 2013), 11 pages. https://doi.org/10.1145/2508363.2508422

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. 2015. The SGGX microflake distribution. *ACM Trans. Graph.* 34, 4, Article 48 (jul 2015), 11 pages. https://doi.org/10.1145/2766988

Piti Irawan and Steve Marschner. 2012. Specular reflection from woven cloth. *ACM Trans. Graph.* 31, 1, Article 11 (feb 2012), 20 pages. https://doi.org/10.1145/2077341.2077352

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. 2010. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.* 29, 4, Article 53 (jul 2010), 13 pages. https://doi.org/10.1145/1778765.1778790

Wenhua Jin, Beibei Wang, Milos Hasan, Yu Guo, Steve Marschner, and Ling-Qi Yan. 2022. Woven Fabric Capture from a Single Photo. In *SIGGRAPH Asia 2022 Conference Papers (SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 33, 8 pages. https://doi.org/10.1145/3550469.3555380

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš 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. https://github.com/NVIDIAGameWorks/Falcor

Anton S. Kaplanyan, Stephen Hill, Anjul Patney, and Aaron Lefohn. 2016. Filtering Distributions of Normals for Shading Antialiasing. In *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*, Ulf Assarsson and Warren Hunt (Eds.). The Eurographics Association. https://doi.org/10.2312/hpg.20161201

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. 2016. Matching Real Fabrics with Micro-Appearance Models. *ACM Trans. Graph.* 35, 1, Article 1 (dec 2016), 26 pages. https://doi.org/10.1145/2818648

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: Multi-Resolution Neural Materials. *ACM Trans. Graph.* 40, 4, Article 175 (jul 2021), 13 pages. https://doi.org/10.1145/3450626.3459795

Guillaume Loubet and Fabrice Neyret. 2017. Hybrid mesh-volume LoDs for all-scale pre-filtering of complex 3D assets. *Computer Graphics Forum* 36, 2 (2017), 431–442. https://doi.org/10.1111/cgf.13138

Zahra Montazeri, Søren B. Gammelmark, Shuang Zhao, and Henrik Wann Jensen. 2020. A practical ply-based appearance model of woven fabrics. *ACM Trans. Graph.* 39, 6, Article 251 (nov 2020), 13 pages. https://doi.org/10.1145/3414685.3417777

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4, Article 36 (Aug. 2021), 36:1–36:16 pages. https://doi.org/10.1145/3450626.3459812

Marc Olano and Dan Baker. 2010. LEAN Mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)*. Association for Computing Machinery, New York, NY, USA, 181–188. https://doi.org/10.1145/1730804.1730834

Iman Sadeghi, Oleg Bisker, Joachim De Deken, and Henrik Wann Jensen. 2013. A practical microcylinder appearance model for cloth rendering. *ACM Trans. Graph.* 32, 2, Article 14 (apr 2013), 12 pages. https://doi.org/10.1145/2451236.2451240

Beibei Wang, Wenhua Jin, Miloš Hašan, and Ling-Qi Yan. 2022. SpongeCake: A Layered Microflake Surface Appearance Model. *ACM Trans. Graph.* 42, 1, Article 8 (sep 2022), 16 pages. https://doi.org/10.1145/3546940

Lance Williams. 1983. Pyramidal parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '83)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/800059.801126

Lifan Wu, Shuang Zhao, Ling-Qi Yan, and Ravi Ramamoorthi. 2019. Accurate appearance preserving prefiltering for rendering displacement-mapped surfaces. *ACM Trans. Graph.* 38, 4, Article 137 (jul 2019), 14 pages. https://doi.org/10.1145/3306346.3322936

Chao Xu, Rui Wang, Shuang Zhao, and Hujun Bao. 2017. Real-Time Linear BRDF MIP-Mapping. *Computer Graphics Forum* 36, 4 (2017), 27–34. https://doi.org/10.1111/cgf.13221

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.* (apr 2024). https://doi.org/10.1145/3659577 Just Accepted.

Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. 2011. Building volumetric appearance models of fabric using micro CT imaging. *ACM Trans. Graph.* 30, 4, Article 44 (jul 2011), 10 pages. https://doi.org/10.1145/2010324.1964939

Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. 2016. Downsampling scattering parameters for rendering anisotropic media. *ACM Trans. Graph.* 35, 6, Article 166 (dec 2016), 11 pages. https://doi.org/10.1145/2980179.2980228

Junqiu Zhu, Adrian Jarabo, Carlos Aliaga, Ling-Qi Yan, and Matt Jen-Yuan Chiang. 2023a. A Realistic Surface-Based Cloth Rendering Model. In *ACM SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/3588432.3591554

Junqiu Zhu, Zahra Montazeri, Jean-Marie Aubry, Ling-Qi Yan, and Andrea Weidlich. 2023b. A Practical and Hierarchical Yarn-based Shading Model for Cloth. *Computer Graphics Forum* 42, 4 (2023), 2–11. https://doi.org/10.1111/cgf.14894

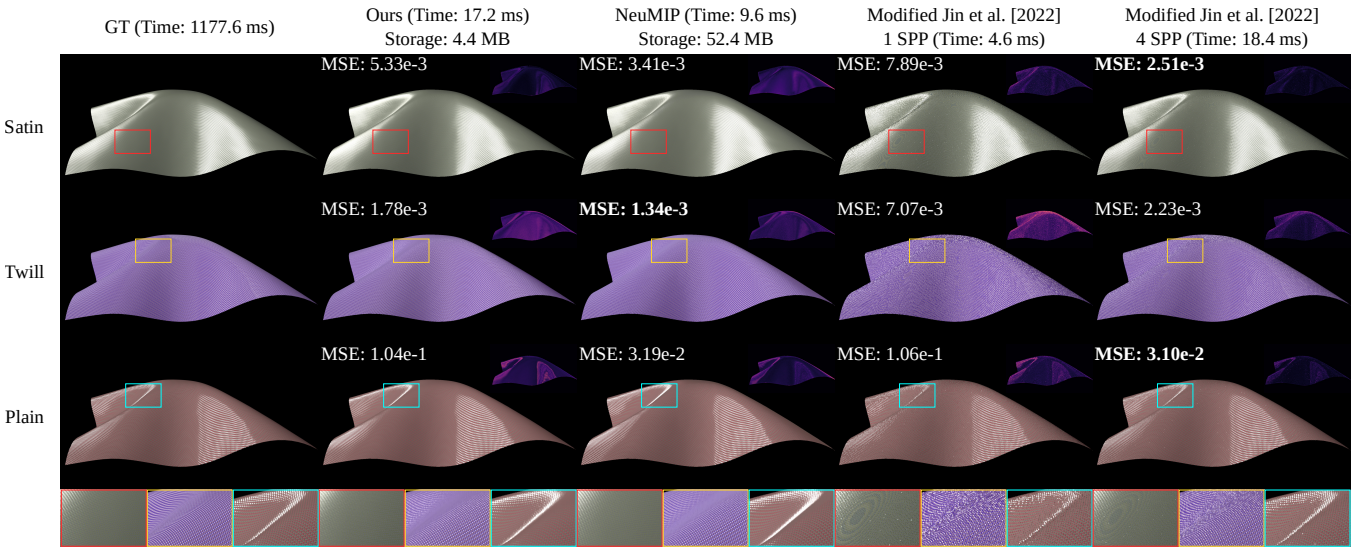

Figure 4: Comparison among our method, NeuMIP [Kuznetsov et al. 2021] and modified Jin et al. [2022] on seen materials. While NeuMIP produces the lowest MSE occasionally, it has to be trained per material and does not support editing. The results by modified Jin et al. [2022] show noticeable aliasing, even with more samples. In contrast, our results are much smoother and free from aliasing.

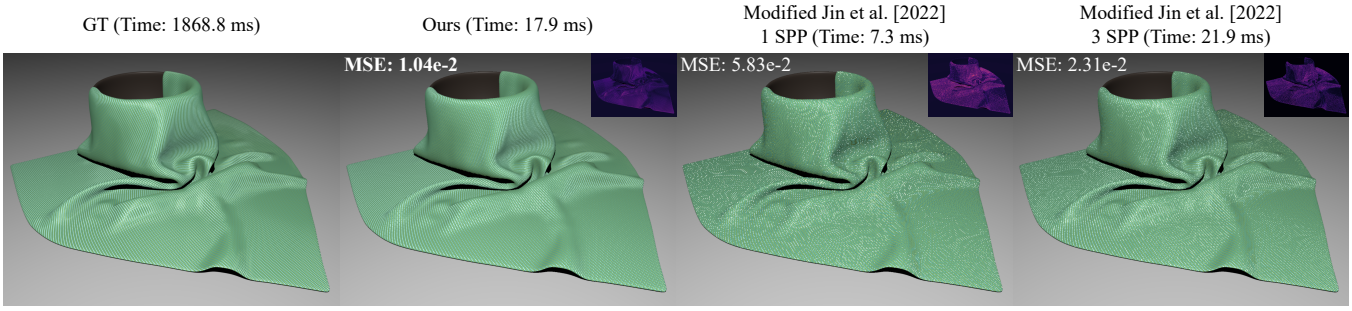

Figure 5: Comparison between our method and modified Jin et al. [2022] on an unseen material (a twill 4×4 pattern). Our method produces results with lower MSE and less aliasing than both rendering results (1 SPP and 3 SPP) by modified Jin et al. [2022].

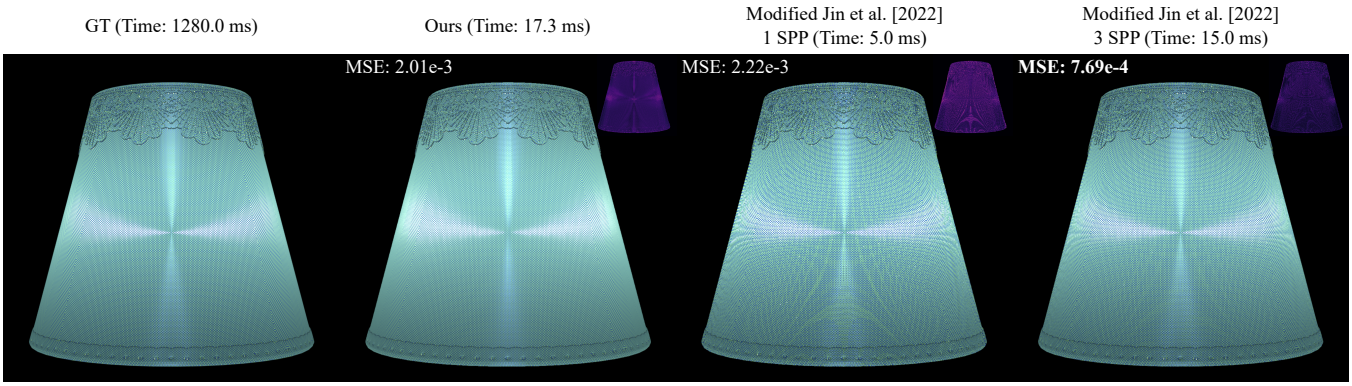

Figure 6: Comparison between our method and modified Jin et al. [2022] on an unseen BTDF material (a plain pattern). Both rendering results (with 1 SPP and 3 SPP) by modified Jin et al. [2022] have apparent aliasing, while our results are closer to the ground truth.

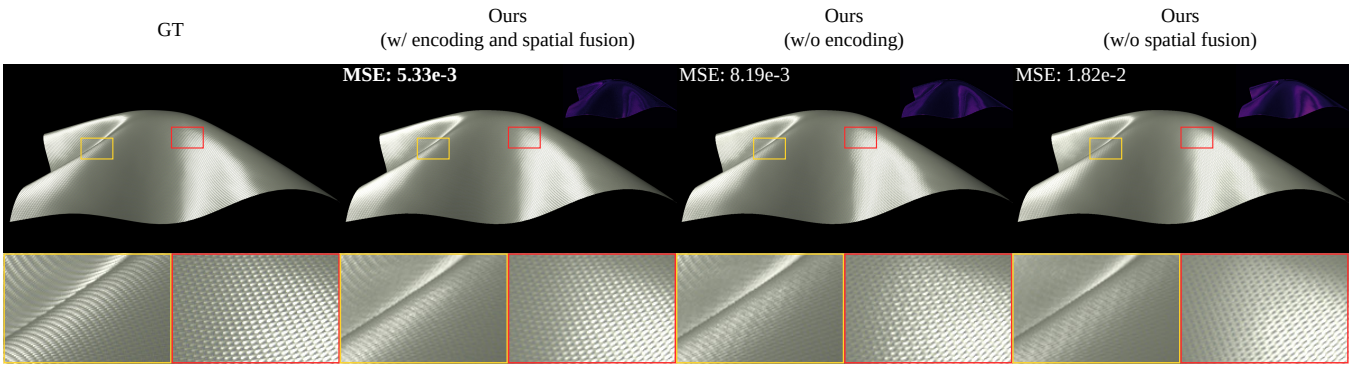

Figure 7: The influence of two designs (the one-blob encoding and the spatial fusion) in our network. With both components, our network has a more powerful representation ability, leading to a higher quality of rendering results.

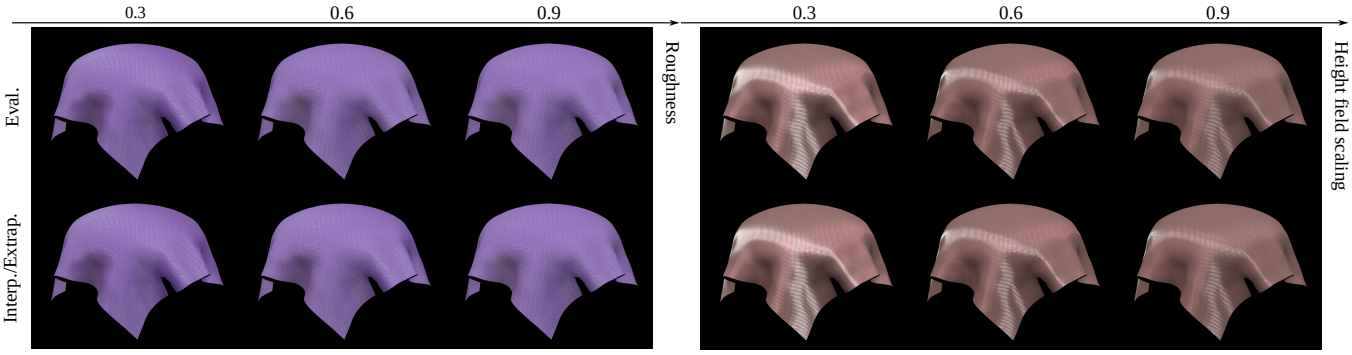

Figure 8: Interpolation and extrapolation of material latent vectors with roughness (left) and height field scaling factor (right) as the weights. The interpolated/extrapolated material latent vectors can produce rendered results similar to those obtained by the directly encoded material latent vectors.

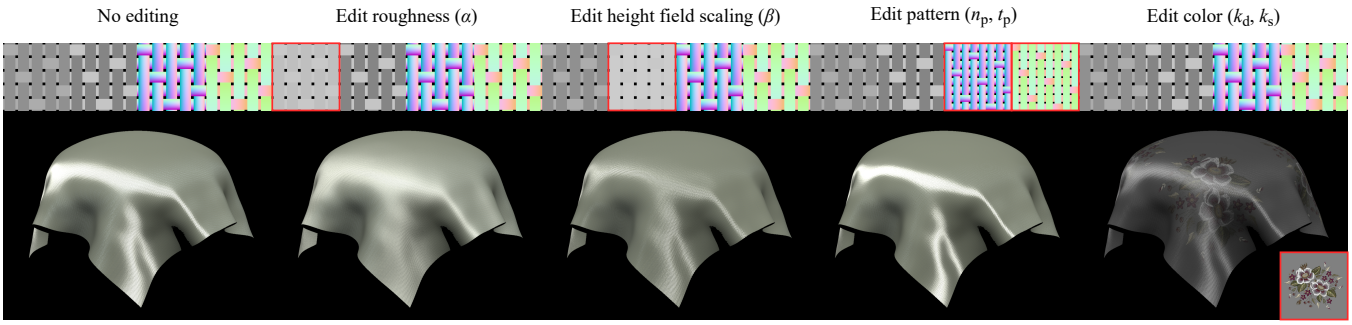

Figure 9: Our method supports real-time editing of several parameters, including roughness, height field scaling factor, pattern (normal and orientation textures), and albedos, producing various rendering results.