Eurographics Symposium on Rendering (2023)
T. Ritschel and A. Weidlich (Editors)

SparseBTF: Sparse Representation Learning for Bidirectional Texture Functions

Behnaz Kavoosighafi[1], Jeppe Revall Frisvad[2], Saghi Hajisharif[1], Jonas Unger[1], and Ehsan Miandji[1]

[1]Linköping University, Sweden
[2]Technical University of Denmark

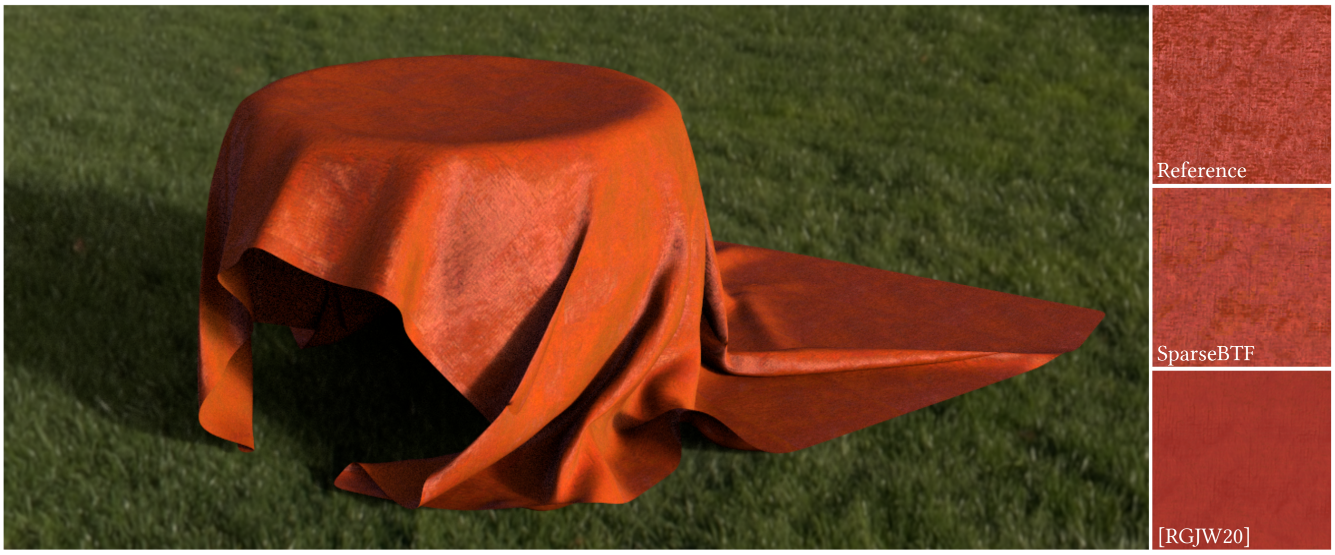

Figure 1: *Our SparseBTF representing* Fabric12 *[WGK14]. Left: Illuminated by a high dynamic range environment map (Pixar Campus) and rendered for 6 seconds with 158 frames per second (GeForce RTX 3080) using a progressive path tracer that we built upon the NVIDIA OptiX framework [PBD*10] (v. 7.5). Right: Comparison of reconstructed images using our method and [RGJW20] with the same number of coefficients. Incident azimuth and elevation angles:* 0°, 45°. *Observation azimuth and elevation angles:* 0°, 90°.

Abstract

We propose a novel dictionary-based representation learning model for Bidirectional Texture Functions (BTFs) aiming at compact storage, real-time rendering performance, and high image quality. Our model is trained once, using a small training set, and then used to obtain a sparse tensor containing the model parameters. Our technique exploits redundancies in the data across all dimensions simultaneously, as opposed to existing methods that use only angular information and ignore correlations in the spatial domain. We show that our model admits efficient angular interpolation directly in the model space, rather than the BTF space, leading to a notably higher rendering speed than in previous work. Additionally, the high quality-storage cost tradeoff enabled by our method facilitates controlling the image quality, storage cost, and rendering speed using a single parameter, the number of coefficients. Previous methods rely on a fixed number of latent variables for training and testing, hence limiting the potential for achieving a favorable quality-storage cost tradeoff and scalability. Our experimental results demonstrate that our method outperforms existing methods both quantitatively and qualitatively, as well as achieving a higher compression ratio and rendering speed.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Rendering; Reflectance modeling; Machine learning approaches;***

DOI: 10.2312/sr.20231123

1. Introduction

Accurate representation and simulation of material appearance are key components in photo-realistic rendering. In the strive for realism, many applications render images using captured appearance of real world objects. However, accurately modeling the reflectance properties of their materials, especially those with complex spatial variation can be a challenge. To address this challenge, Dana et al. [DvGNK99] proposed the *Bidirectional Texture Function* (BTF) and captured the first BTF dataset (CUReT) from 60 materials with 205 different combinations of light and camera directions.

When using spherical coordinates for angular information and 2D parameterized surfaces for spatial information, a BTF is a 7-dimensional function $\mathcal{B}(\mathbf{x}, \omega_i, \omega_o, \lambda)$ of a 2D surface position $\mathbf{x}$, an incoming direction ω_i, an outgoing direction ω_o, and a wavelength parameter λ. The BTF describes the appearance of a texture for different light and view directions. Due to the inadequacy of the angular information in the CUReT BTF dataset, research has produced several BTF datasets with denser sampling over the directions of incidence and observation [SSK03; KMBK03; HFV12; WGK14; FKH*18]. Although BTFs enable good measurement and representation of complex materials, they suffer from the fact that densely sampled BTFs exhibit a very large memory footprint making them difficult to use and, in many cases, even unsuitable for practical rendering purposes.

Due to the large memory footprint, BTF data is most often transformed into a *representation* space where the redundancies in the data are exploited to reduce storage requirements. Decomposition-based methods, using e.g. matrix or tensor factorization, are commonly used for this purpose. Principal Component Analysis (PCA) can, for instance, decompose the data into the multiplication of a basis matrix, holding the principal components as columns, and a coefficient matrix, containing the principal values for each BTF as columns [CD01; WGK14]. Regardless of the method, a data representation is achieved by deriving, optimizing, or training a model that represents the original data using a set of parameters, also known as coefficients, where the number of parameters is much smaller than the number of data elements.

For the practical use of a BTF data representation model, we find that the representation should ideally fulfill four criteria (a–d) that we describe in the following. (a) *Low memory footprint*. The memory footprint often refers to the number of coefficients needed to represent a BTF in a compact form. The storage cost of the model itself is also a key factor. (b) *Quality-storage cost tradeoff*. An efficient data representation model provides a tradeoff between quality and storage cost. As the number of coefficients increases, we expect a steady reduction in the representation error. (c) *Fast local reconstruction*. Fast access to single elements within the 7D BTF function is a requirement for efficient offline and real-time rendering. (d) *One-time model training*. It is important to train a rich and robust model such that it can be directly utilized on many unseen BTFs, even those that differ significantly from the training set. Moreover, this property eliminates the need for storing multiple models, i.e. one for each BTF.

We present a novel data-driven BTF representation enabling a low memory footprint and real-time rendering performance while maintaining the high visual quality and numerical accuracy required for physically-based photo-realistic rendering. Our main contributions are in summary:

- A novel unsupervised data-driven model for BTF representation. The model is trained once, using a small training set, and is able to exploit redundancies in the data across all BTF dimensions simultaneously, providing a distinct advantage over previous methods.
- Real-time BTF rendering, enabled by a sparse set of coefficients for each BTF that admits interpolation directly in the coefficient space.
- Significantly higher image quality, quality-storage cost tradeoff, and rendering speed compared to the state-of-the-art methods.

In contrast to recent work on BTF representation [RJGW19; KMX*21], our model is trained once utilizing a small training set, with the capability to accurately represent a diverse range of materials and appearances. Since the number of coefficients is user-defined, and changing this parameter does not require a retraining of the model, our method provides an intuitive trade-off between quality and storage cost. This characteristic highlights a high quality-storage cost tradeoff of SparseBTF. In Table 1, we compare our method (SparseBTF) with state-of-the-art methods in terms of fulfilling the four criteria (a–d). As exemplified in Figure 1, for an equal storage cost, our method represents the reference more faithfully when compared to the most recent method featuring a one-time model training [RGJW20]. In addition, the improved representation quality achieved by SparseBTF is accomplished while maintaining the real-time performance, in contrast to the baseline method that reports offline rendering performance [RGJW20]. The code and data for this paper are available at `https://github.com/behnazkavoosi/SparseBTF`.

2. Previous Work

Decomposition-based methods. Most decomposition-based methods on BTF modeling rely on matrix factorization and tensor decomposition due to the multi-dimensional nature of BTF data [MMS*05; FH09; HF13]. Koudelka et al. [KMBK03] apply a full matrix factorization to a matrix with Apparent BRDFs (ABRDF) as columns to take the correlation between angular domains into account. Müller et al. [MMK03] propose a per-cluster factorization based on K-means, where PCA is computed for each cluster rather than the entire matrix to reduce memory consumption. Cula and Dana [CD01], on the other hand, use PCA to reduce the dimensionality of the space of feature histograms obtained from the statistical distribution of features in the angular images. Liu et al. [LHZ*04] employ Singular Value Decomposition (SVD) to factorize a BTF matrix to generate geometry maps and point appearance functions, resulting in efficient synthesis and fast rendering of BTFs on arbitrary surfaces. In a more recent work, Weinmann et al. [WGK14] encode their measured BTFs using truncated PCA in order to compress the data set. While these factorization-based methods can achieve an acceptable reconstruction quality, they cannot utilize the spatial coherence between ABRDFs, resulting in relatively low compression ratios.

Tensor decomposition-based methods, on the other hand, attain

	decomposition-based	[RJGW19]	[RGJW20]	[KMX*21]	SparseBTF
Low memory footprint		✓	✓	✓	✓
Quality-storage cost tradeoff	✓				✓
Fast local reconstruction	✓			✓	✓
One-time model training			✓		✓

Table 1: *Support of criteria for practical use and broad model applicability in graphics. Comparison of our method (SparseBTF) with state of the art.*

a more compact representation of BTFs compared to matrix factorization approaches. In particular, an N-mode SVD can be applied to the 7-dimensional BTF tensor directly to take advantage of correlations among different modes of the tensor, i.e., spatial, angular, and temporal domains [VT04; WWS*05]. Tsai et al. [TS12; Tsa15] introduced K-clustered tensor approximation to partition high-dimensional data, e.g. BTF, into clusters to harness the intra- and inter-cluster dependencies. Havran et al. [HFM10] parameterize the BTF data as a set of multi-dimensional conditional probability density functions compressed by a multi-level vector quantization algorithm. In other work [TFLS11], a decomposition of the reflectance field in multivariate spherical radial basis functions (SRBFs) along with a hierarchical BTF fitting model is proposed to model the existing non-linearity in complex materials while preserving the spatial dependencies. Wu et al. [WDR11] present a sparse parametric model for BTFs, where a stagewise Lasso-based fitting algorithm is used to decompose a BTF into a dictionary and a set of sparse coefficients.

Decomposition-based methods require solving a computationally demanding optimization problem for each BTF individually to obtain a high-capacity model and its coefficients. Once the decomposition is performed, reconstruction performance is typically adequate for real-time rendering applications. Moreover, since both the coefficients and the model has to be saved for each BTF, decomposition-based methods impose a higher memory footprint.

Learning-based methods. Ruiters and Klein [RK09] use the K-SVD algorithm [AEB06] to split a massive BTF tensor into one dictionary and two sparse tensors. This lowers the memory footprint while enhancing the reconstruction quality. Den Brok et al. [dBWK15] harness the linear bases obtained from rank-512 approximations of BTFs to reduce the dimensionality of BTFs. However, the model requires a large training set to achieve an acceptable reconstruction error. In a later study [dBWK18], minimization of a relative error metric is utilized to compute a linear basis for BTF representation, which allows for sparse and multiplexed data acquisition. Tongbuasirilai et al. [TUGM22] propose sparse modeling of Bidirectional Reflectance Distribution Functions (BRDFs) by training multi-dimensional dictionaries for a set of BRDFs. While we draw inspiration from this model, it is important to acknowledge that adapting this technique to BTFs is a non-trivial task due to the high dimensionality and intricate nature of the data. Learning a set of dictionaries capable of accurately representing the wide range of variability found in BTFs poses a particularly difficult challenge as compared to BRDF modeling. To overcome these challenges, we develop novel approaches that efficiently capture the richness of BTF data. Furthermore, SparseBTF achieves real-time rendering of BTFs and real-time interpolation in the angular domain in the coefficient space, which has not been explored in previous work on BRDFs [TUGM22].

Recently, several studies have focused on neural material representation models, with various architectures utilized to encode material as a set of low-dimensional latent vectors [HGC*20; SRRW21; ZZW*21; FWH*22]. Rainer et al. [RJGW19] propose a per-material encoder-decoder architecture, where the auto-encoder generates 8 latent coefficients per pixel from the input ABRDFs for each BTF. The RGB color of each pixel corresponding to one particular set of the incident and observed directions is then recovered using the stored latent vectors. However, since the network has to be re-trained for each BTF, they later generalized this architecture by adding a Multi-Layer Perceptron (MLP) to the auto-encoder to encode various materials using a common latent space with 32 coefficients per spatial position [RGJW20]. While their model offers a low memory footprint and supports one-time training, it does not exploit correlations in the spatial domain since the model is trained on ABRDFs. In Section 4, we demonstrate that SparseBTF exhibits a substantially superior quality-storage cost tradeoff compared to the methods proposed in [RJGW19] and [RGJW20].

In a recent study, Kuznetsov et al. [KMX*21] combine two MLP networks for mipmapping and parallax mapping to represent a multi-resolution BTF, referred to as MBTF. However, this approach has been found to suffer from overfitting to a specific material, as in earlier work [RJGW19], demanding re-training for every individual BTF. The authors extend their work by incorporating opacity and reflectance prediction modules to handle silhouette effects on curved surfaces [KWM*22]. More recently, two changes in the sampling strategy and activation layer of NeuMIP [KMX*21] were proposed, leading to improvements in quality, storage cost, and performance [QP22].

3. SparseBTF

In our SparseBTF method, we represent a BTF by a set of sparse coefficients obtained using a learned 4D dictionary ensemble. We first divide a BTF into a set of *data points* to enable efficient processing of the data in the remaining steps (Section 3.1). The 4D dictionary ensemble is then trained once using our proposed algorithm (Section 3.2). Given the trained model and a BTF in the test set, we obtain the sparse coefficients through a simple and fast greedy algorithm (Section 3.3). A discussion of the reasons behind the improvements of our method over the prior art (Table 1) is included in Section 3.2.

Each individual point from a BTF data point is recovered using the sparse coefficients and the 4D dictionary ensemble (Section 3.4). We also propose a method for interpolation in the angular

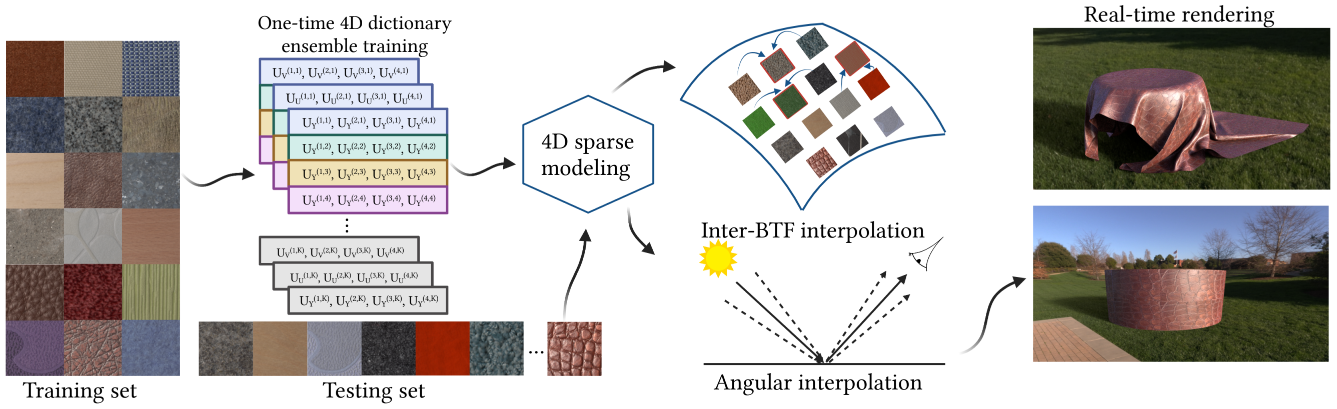
One-time 4D dictionary ensemble training
$U_V^{(1,1)}, U_V^{(2,1)}, U_V^{(3,1)}, U_V^{(4,1)}$
$U_U^{(1,1)}, U_U^{(2,1)}, U_U^{(3,1)}, U_U^{(4,1)}$
$U_Y^{(1,1)}, U_Y^{(2,1)}, U_Y^{(3,1)}, U_Y^{(4,1)}$
$U_Y^{(1,2)}, U_Y^{(2,2)}, U_Y^{(3,2)}, U_Y^{(4,2)}$
$U_Y^{(1,3)}, U_Y^{(2,3)}, U_Y^{(3,3)}, U_Y^{(4,3)}$
$U_Y^{(1,4)}, U_Y^{(2,4)}, U_Y^{(3,4)}, U_Y^{(4,4)}$
$U_V^{(1,K)}, U_V^{(2,K)}, U_V^{(3,K)}, U_V^{(4,K)}$
$U_U^{(1,K)}, U_U^{(2,K)}, U_U^{(3,K)}, U_U^{(4,K)}$
$U_Y^{(1,K)}, U_Y^{(2,K)}, U_Y^{(3,K)}, U_Y^{(4,K)}$
4D sparse modeling
Real-time rendering
Inter-BTF interpolation
Angular interpolation
Training set
Testing set

Figure 2: *Overview of the proposed framework, SparseBTF, for learning-based sparse representation of BTFs. A training set containing a small number of BTFs is used for learning an ensemble of 4D dictionaries, which is performed only once. Given a set of BTFs in the test set and the dictionary ensemble, we obtain 4D sparse coefficients. The coefficient set, defining the model space, admits Inter-BTF interpolation directly using the coefficients since the model space defined by the coefficients and the dictionary ensemble is a smooth manifold. Moreover, SparseBTF benefits from fast angular interpolation in the model space, as opposed to the BTF space, which enables real-time rendering. The rendered images are illuminated by a high dynamic range environment map (Pixar Campus).*

domain, i.e., for incident and view directions, directly in the coefficient space and without the need for multiple reconstructions. Fast local reconstruction of a single BTF element, together with interpolation in the coefficient space, enables SparseBTF to achieve high frame rates on a single consumer-level GPU. Finally, we describe our method for interpolating between two or more BTFs in Section 3.5. We show that, given a set of BTFs, SparseBTF admits constructing a smooth manifold over this set, which enables the synthesis of novel BTFs directly in the model space using sparse coefficients. The full pipeline of our method is shown in Figure 2.

Notation. We use boldface lower-case letters for vectors ($\mathbf{a}$), boldface upper-case for matrices ($\mathbf{A}$), and calligraphic letters for tensors ($\mathcal{A}$). A finite set of objects is indexed by superscripts, e.g. $\{\mathbf{A}^{(i)}\}_{i=1}^{N}$, whereas individual elements of $\mathbf{a}$, $\mathbf{A}$ and $\mathcal{A}$ are denoted $\mathbf{a}_i$, $\mathbf{A}_{i_1,i_2}$ and $\mathcal{A}_{i_1,\ldots,i_n}$, respectively. The ℓ_p norm of a vector $\mathbf{s}$, for $1 \leq p \leq \infty$, is denoted by $||\mathbf{s}||_p$, and we use $||\mathbf{s}||_F$ for the Frobenius norm. The ℓ_0 pseudo-norm of a vector, denoted $||\mathbf{s}||_0$, defines the number of non-zero elements in the vector. The n-mode product of a tensor $\mathcal{S}$ and a matrix $\mathbf{A}$ is denoted $\mathcal{S} \times_n \mathbf{A}$. This operation involves multiplying each mode-n fiber in tensor $\mathcal{S}$ with matrix $\mathbf{A}$.

3.1. BTF data points

A BTF is a seven-dimensional (7D) function $\mathcal{B}(\mathbf{x}, \omega_i, \omega_o, \lambda)$ parameterized by a 2D surface position $\mathbf{x}$, a direction of incidence light ω_i, a direction of observation ω_o, both in 2D spherical coordinates, and a wavelength parameter λ. By vectorizing angular dimensions corresponding to incoming and outgoing directions, we obtain a 5D parameterization. In this paper, we process each color channel (λ) of a BTF individually. Since we consider RGB data, this leads to a triplet of 4D functions for each BTF. For efficient processing in training, testing, and rendering, we divide each BTF into a set of data points. Data points can be obtained using a 1D to a 4D non-overlapping sliding window over the discrete 4D BTF tensor.

To enable efficient BTF representations, we define 4D data points that include all the angular information of the BTF as well as a small spatial patch, e.g., 10×10 pixels. Our motivation for this approach is that the spatial domain of a BTF often contains high-frequency information, while the angular domain is relatively smooth. As a result, the representation model can exploit correlations in both the angular and spatial domains without the need for a large number of parameters. Moreover, having all the angular information within a data point facilitates efficient angular interpolation in the coefficient space, see Section 3.4. Another advantage of incorporating small spatial patches within a data point is that BTFs are often used for modeling the appearance of textures, which often exhibit repeated patterns in the spatial domain. By creating small spatial patches, we can exploit such non-local spatial correlations. This approach, known as patch-based processing, is well-established in the fields of image processing and machine learning.

After extracting data points from one or multiple BTFs, we apply a log transformation to reduce the dynamic range of the measurements. Subsequently, a color transformation is applied to convert the RGB values (as existing BTF data sets are mostly in RGB color space) to the YUV color space [RKAJ08; XSJT07; SWRK11; KWK17]. In each BTF image, which represents spatial information of a specific light and view direction, we apply a normalization process by subtracting the mean and dividing by the standard deviation. The transformation to YUV color space decorrelates the reflectance values, allowing them to be treated independently without introducing artifacts. More importantly, as the luminance channel contains the majority of the signal's information, it allows us to use far fewer coefficients for the U and V channels. This reduction in model complexity and cost leads to an increase in rendering performance, as the model becomes much more compact.

3.2. Learning an ensemble of 4D dictionaries

High-dimensional visual data, such as BRDFs and BTFs, can be represented using two main approaches. Either the high-dimensional data points are converted to 1D or 2D signals to facilitate data processing using existing tools within linear algebra and machine learning [NJR15; SESM22], or the data points are defined in such a way that they only contain one or two dimensions from the signal [RGJW20; KMX*21]. Unlike these two well-established approaches, the key observation in designing the SparseBTF algorithm is the simultaneous exploitation of correlations present in all the dimensions of a 4D BTF.

Let $\{\mathcal{X}^{(i)}\}_{i=1}^{N_l}$ be a set of N_l BTF data points with dimensions $\mathcal{X}^{(i)} \in \mathbb{R}^{m_1 \times m_2 \times m_3 \times m_4}$ where (m_1, m_2) represent the spatial patch size, and (m_3, m_4) define the incoming and outgoing directions, respectively. We assume that all the data points have the same dimensionality and that each color channel can be processed independently. Since SparseBTF is a learning-based approach, we define a set of data points, $\{\mathcal{X}^{(i)}\}_{i=1}^{N_l}$, as the training set, and our testing data is denoted by $\{\mathcal{Y}^{(i)}\}_{i=1}^{N_t}$. In our experiments, we used the Bonn BTF data set [WGK14], which features an angular resolution of 151×151. Choosing a spatial patch size of 10×10 leads to data points of dimensionality $m_1 \times m_2 \times m_3 \times m_4 = 10 \times 10 \times 151 \times 151$. Importantly, the training phase of SparseBTF is a one-time process that is performed on a small number of BTFs, as opposed to previous approaches [RJGW19; KMX*21], where a separate model is trained for each individual BTF. Note that the notation utilized henceforth is formally introduced in the Notation subsection of Section 3.

Following the criteria listed in Section 1 and Table 1, we use a 4D dictionary ensemble model to represent BTF data. The ensemble is learned from a training set, but since the representation is based on sparse tensor decompositions, our method benefits from the advantages of both learning-based and decomposition-based approaches. At its core, SparseBTF represents a data point $\mathcal{X}^{(i)}$ as

$$\mathcal{X}^{(i)} = \mathcal{S}^{(i)} \times_1 \mathbf{U}^{(1,k)} \times_2 \mathbf{U}^{(2,k)} \times_3 \mathbf{U}^{(3,k)} \times_4 \mathbf{U}^{(4,k)}, \tag{1}$$

where the set $\{\mathbf{U}^{(1,k)}, \mathbf{U}^{(2,k)}, \mathbf{U}^{(3,k)}, \mathbf{U}^{(4,k)}\}_{k=1}^{K}$ describes an ensemble of K dictionaries, each containing four basis matrices corresponding to the four dimensions of a BTF data point. In particular, we have $\mathbf{U}^{(1,k)} \in \mathbb{R}^{m_1 \times m_1}$, $\mathbf{U}^{(2,k)} \in \mathbb{R}^{m_2 \times m_2}$, $\mathbf{U}^{(3,k)} \in \mathbb{R}^{m_3 \times m_3}$, and $\mathbf{U}^{(4,k)} \in \mathbb{R}^{m_4 \times m_4}$. Thus, each basis matrix $\mathbf{U}^{(n,k)}$ learns a transformation from the nth dimension of the data point to the sparse model space. The tensor $\mathcal{S}^{(i)} \in \mathbb{R}^{m_1 \times m_2 \times m_3 \times m_4}$ is sparse and contains the set of coefficients for the data point $\mathcal{X}^{(i)}$. The sparsity of a tensor is typically measured using the ℓ_0 norm.

A similar model, albeit with different model dimensionality, is used in existing work for light field compression [MHU19] and measured BRDF representation [TUGM22]. However, the direct application of such previous approaches to the case of BTF representation poses several challenges making it infeasible. First, BTF data sets are orders of magnitude larger than measured BRDFs or light fields, thus requiring a higher compression ratio while maintaining the rendering quality. Second, BTF rendering requires fast local reconstruction of a single element of the BTF given a spatial location, and incoming and outgoing directions. Third, given the low angular resolution of existing acquisition systems and data sets, efficient interpolation is crucial for rendering. Previous methods using a model similar to the one in Eq. (1) have not considered the interpolation of elements within a data point in their sparse form. Finally, in our initial experiments, we found that including color information in a data point leads to color artifacts because of the complexity of BTF data along the spatial dimensions, which dominates the learning algorithm. On the other hand, using the RGB color space also introduces color artifacts since small errors in each channel accumulate into perceivable artifacts. We use the YUV space to separate the luminance and chroma components. This allows us to employ a varying number of model parameters for each channel and reduce color artifacts.

	Fabric10	Wallpaper06	Wood10
SparseBTF	**2.1**	**2.8**	**1.1**
[MHU19]	7.3	8.2	6.9

Table 2: *Mean squared reconstruction error ($\times 10^{-4}$) for three BTFs from the data set presented in [WGK14]. The storage costs for both methods are equal.*

We compared the reconstruction error of SparseBTF to the method proposed by Miandji et al. [MHU19], which does not include any BTF-specific transformations or processing. Table 2 reveals that SparseBTF achieves an average fourfold reduction in error as compared to Miandji et al., highlighting the importance of domain-specific data processing. The results show that the processing plays a crucial role in reducing complexity and extracting the most relevant features, facilitating the learning process and enhancing the overall efficiency of SparseBTF.

SparseBTF trains an ensemble of $K \ll N_l$ 4D dictionaries, given a training sparsity parameter, τ_l, by optimizing the following objective function over a training set of BTF data points $\{\mathcal{X}^{(i)}\}_{i=1}^{N_l}$

$$\min_{\mathbf{U}^{(j,k)}, \mathcal{S}^{(i,k)}, \mathbf{M}_{i,k}} \sum_{i=1}^{N_l} \sum_{k=1}^{K} \mathbf{M}_{i,k} \left\| \mathcal{X}^{(i)} - \mathcal{S}^{(i,k)} \times_1 \mathbf{U}^{(1,k)} \times_2 \mathbf{U}^{(2,k)} \times_3 \mathbf{U}^{(3,k)} \times_4 \mathbf{U}^{(4,k)} \right\|_F^2 \tag{2a}$$

subject to

$$\left(\mathbf{U}^{(j,k)}\right)^T \mathbf{U}^{(j,k)} = \mathbf{I}, \ \forall k = 1, \dots, K, \ \forall j = 1, \dots, 4, \tag{2b}$$

$$\left\| \mathcal{S}^{(i,k)} \right\|_0 \leq \tau_l, \tag{2c}$$

$$\sum_{k=1}^{K} \mathbf{M}_{i,k} = 1, \ \forall i = 1, \dots, N_l, \tag{2d}$$

where (2b) guarantees the orthonormality of each dictionary and (2c) enforces sparsity of the representation. The orthonormality of each basis matrix is crucial for achieving fast and local reconstructions. More importantly, we exploit this property in deriving a simple approach for the interpolation of values within a BTF data point, as well as interpolating distinct BTFs for novel BTF synthesis. It is the orthogonality of each basis that admits a locally linear representation in the model space, hence constituting a smooth manifold that can be explored by the manipulation of sparse coefficients. Equation (2d) associates each data point to one 4D dictionary using the membership matrix $\mathbf{M} \in \mathbb{R}^{N_l \times K}$. The matrix $\mathbf{M}$ is thus a binary clustering matrix where only one component is nonzero for each row. The entire training algorithm is available in the supplementary material.

Since we process each YUV color channel independently, we train three ensembles. Due to the sensitivity of the human visual system to brightness variations, we train a dictionary with lower sparsity for the Y channel, i.e., with a higher number of coefficients. The chroma channels, i.e., UV, require fewer coefficients since they carry less information compared to luminance.

3.3. Sparse representation of BTFs

As discussed in Section 3.2, the training algorithm clusters the set of BTF data points in the training set so that each 4D dictionary best approximates its corresponding cluster of data points. Once the ensemble is learned on the training set, the test set $\{\mathcal{Y}^{(i)}\}_{i=1}^{N_t}$, containing N_t BTF data points, together with the ensemble are used to compute the sparse coefficient tensors. This amounts to determining a dictionary in the ensemble for each data point that results in the sparsest representation with the least representation error. Recall that each BTF data point is represented by one dictionary. We, therefore, project each data point onto all 4D dictionaries in the ensemble and pick the dictionary that produces the sparsest solution with the smallest error. The projection is done by evaluating

$$\hat{\mathcal{S}}^{(i,k)} = \mathcal{Y}^i \times_1 \left(\mathbf{U}^{(1,k)}\right)^T \times_2 \left(\mathbf{U}^{(2,k)}\right)^T \times_3 \left(\mathbf{U}^{(3,k)}\right)^T \times_4 \left(\mathbf{U}^{(4,k)}\right)^T, \quad (3)$$

where $\hat{\mathcal{S}}^{(i,k)}$ is the sparse coefficient tensor of the ith data point using the kth dictionary. Let τ_t be the user-defined sparsity parameter used for the test set, i.e., the testing sparsity. We nullify the smallest absolute values in $\hat{\mathcal{S}}^{(i,k)}$ until the desired sparsity is achieved or when the error becomes larger than a user-defined threshold. Note that if the number of non-zero values is equal to τ_t for all dictionaries, then the dictionary with the least error is picked. We require $\tau_t \geq \tau_l$ to avoid underfitting to the model. Similar to the training sparsity, the testing sparsity is set differently for luminance and chroma channels to minimize the number of parameters while maintaining the representation error. Apart from the sparse coefficient tensors, we require to store a membership vector $\mathbf{m}$ associating each data point in the testing set to its representative dictionary and a vector $\mathbf{z}$ containing the number of non-zero coefficients for each data point.

A unique aspect of SparseBTF is that the number of model parameters for training, τ_l, is distinct from that of testing, τ_t. This is the main reason for achieving a superior quality-storage cost tradeoff as compared to previous work. Once the ensemble is trained using τ_l, any testing BTF can be represented with $\tau_t \in [\tau_l, m_1 m_2 m_3 m_4]$ coefficients, where the maximum is the total number of elements in $\hat{\mathcal{S}}^{(i,k)}$. Moreover, SparseBTF provides a steady increase in reconstruction quality with increasing τ_t, see Table 3. This is in contrast to previous work [RJGW19; RGJW20; KMX*21], where the number of model parameters cannot be controlled during testing or reconstruction. Indeed, existing methods may increase the number of parameters, e.g., by adding layers to the encoder/decoder, as well as increasing the number of latent variables; however, this comes at the cost of re-training the model with a possibly larger training set. Furthermore, depending on the number of parameters, distinct networks should be trained and stored. Even when the number of parameters that are learned is increased, the quality-storage cost tradeoff is still low since the number of parameters is fixed during training and testing, and it may only be optimal for a certain class of BTFs. For instance, using a large number of latent variables for smooth BTFs leads to overfitting, while a small number of latent variables for high-frequency BTFs leads to underfitting. Indeed, SparseBTF can benefit from training based on different τ_l since the sparse representation step described above can pick the most suitable dictionary. We leave this last point for future work.

3.4. Real-time rendering using sparse coefficients

Once the sparse coefficient tensors $\hat{\mathcal{S}}^{(i)}$ for all the data points within the test set and the membership vector $\mathbf{m}$ are obtained, the reconstruction is performed by evaluating

$$\hat{\mathcal{Y}}^{(i)} = \hat{\mathcal{S}}^{(i)} \times_1 \mathbf{U}^{(1,\mathbf{m}_i)} \times_2 \mathbf{U}^{(2,\mathbf{m}_i)} \times_3 \mathbf{U}^{(3,\mathbf{m}_i)} \times_4 \mathbf{U}^{(4,\mathbf{m}_i)}, \quad (4)$$

where $\mathbf{m}_i \leq K$ is the index of the chosen dictionary for the i-th data points and $\hat{\mathcal{Y}}^{(i)} \in \mathbb{R}^{m_1 \times m_2 \times m_3 \times m_4}$ is the reconstructed BTF data point. With a slight abuse of notation, $\hat{\mathcal{S}}^{(i)} \in \mathbb{R}^{m_1 \times m_2 \times m_3 \times m_4}$ denotes the sparse coefficient corresponding to the dictionary $\mathbf{m}_i$. To reduce the storage cost, we store the sparse coefficient tensor $\hat{\mathcal{S}}^{(i)}$ as a set of nonzero values and their corresponding location within the tensor, e.g. using an index tuple $\left(l_1^{\tau_t}, l_2^{\tau_t}, l_3^{\tau_t}, l_4^{\tau_t}\right)$. Note that, without loss of generality, here we assume that all the data points have τ_t coefficients, i.e. a fixed sparsity parameter. This choice was made to facilitate comparisons to previous work; i.e. by fixing the storage cost with respect to the number of coefficients and comparing the image quality. In practice, as described in Section 3.3, the number of coefficients may vary among data points by using an error threshold parameter rather than a fixed sparsity parameter during the computation of the sparse representation. We formulate the reconstruction of a single element within a data point $\mathcal{Y}^{(i)}$ located at (x_1, x_2, x_3, x_4), which we denote by $\mathcal{Y}^{(i)}_{x_1,x_2,x_3,x_4}$, as follows

$$\hat{\mathcal{Y}}^{(i)}_{x_1,x_2,x_3,x_4} = \sum_{z=1}^{\tau_t} \hat{\mathcal{S}}^{(i)}_{l_1^z,l_2^z,l_3^z,l_4^z} \mathbf{U}^{(1,\mathbf{m}_i)}_{x_1,l_1^z} \mathbf{U}^{(2,\mathbf{m}_i)}_{x_2,l_2^z} \mathbf{U}^{(3,\mathbf{m}_i)}_{x_3,l_3^z} \mathbf{U}^{(4,\mathbf{m}_i)}_{x_4,l_4^z}, \quad (5)$$

where l_1^z and l_2^z store the spatial locations and l_3^z and l_4^z denote the angular locations of non-zero coefficients within $\hat{\mathcal{S}}$. The non-zero values in $\hat{\mathcal{S}}^{(i)}$, together with their locations, the membership vector, and the 4D dictionary ensemble, are uploaded as textures to the GPU. The ensemble is uploaded once, since it is fixed, while the coefficients may need to be uploaded progressively if the scene contains many BTFs.

Since BTF data sets are often coarse in the angular domain, we perform interpolation in the angular domain to compute BTF values for novel light and view directions during rendering. First, the k-Nearest Neighbors (kNN) algorithm is utilized to find the two

	Carpet12	Fabric12	Felt12	Leather12	Stone12	Wallpaper12	Wood12
[RJGW19] (16)	1.5	0.8	1.5	14	1.5	1.1	0.9
[RGJW20] (16)	3.3	6.7	2.7	101	4.9	1.9	4.5
SparseBTF (16)	1.8	3.7	1.6	83	6.0	2.2	2.1
[RJGW19] (32)	0.9	0.5	1.0	11	1.1	0.9	0.8
[RGJW20] (32)	2.4	4.5	2.0	90	3.7	1.5	4.0
SparseBTF (32)	1.0	3.0	1.0	71	4.3	1.6	1.6
[RJGW19] (64)	0.5	0.4	0.7	11	0.9	0.7	0.7
[RGJW20] (64)	2.2	4.0	1.8	98	3.7	1.5	4.1
SparseBTF (64)	0.7	2.7	0.7	47	3.2	1.2	1.2
[RJGW19] (128)	0.5	0.4	0.6	10	0.9	0.7	0.7
[RGJW20] (128)	2.2	4.3	1.8	96	3.7	1.4	4.5
SparseBTF (128)	0.5	2.3	0.5	32	2.4	1.0	1.0

Table 3: *Mean squared reconstruction error ($\times 10^{-4}$) of BTFs in the test set with 100×100 texels. SparseBTF (16), SparseBTF (32), SparseBTF (64), and SparseBTF (128) imply that our compression ratio is equal to [RGJW20] and [RJGW19] with 16, 32, 64, and 128 latent coefficients, respectively.*

light/view configurations with solid angle bin centers that are closest to the direction of incidence/observation. We then use Eq. 5 to obtain the reflectance value at four different combinations of the best and second-best bins for the two directions, and perform bilinear interpolation to return the final appearance value. Because each data point includes all the angular information, and the 4D dictionaries are orthonormal, the interpolation with respect to the incoming and outgoing directions can be performed in the coefficient space. Without loss of generality, we perform linear interpolation of two viewing directions, denoted x_s and x_t, using

$$\hat{\mathcal{Y}}^{(i)}_{x_1,x_2,x_3,x_4} = \sum_{z=1}^{\tau_t} \hat{\mathcal{S}}^{(i)}_{l_1^z,l_2^z,l_3^z,l_4^z} \mathbf{U}^{(1,\mathbf{m}_i)}_{x_1,l_1^z} \mathbf{U}^{(2,\mathbf{m}_i)}_{x_2,l_2^z} \mathbf{U}^{(3,\mathbf{m}_i)}_{x_3,l_3^z} \left((1-\alpha)\mathbf{U}^{(4,\mathbf{m}_i)}_{x_s,l_4^z} + \alpha \mathbf{U}^{(4,\mathbf{m}_i)}_{x_t,l_4^z} \right), \quad (6)$$

Interpolation along light directions can be carried out in a similar way. It is noteworthy that our method is not restricted to a particular interpolation technique. Alternative methods for interpolation within the sparse space of coefficients, such as kernel density estimation utilizing various kernel functions [Sil86] can also be applied. Further details and illustrative examples can be found in the supplementary material.

3.5. Inter-BTF interpolation

To interpolate between two or more distinct 4D BTFs in the coefficient space, we adapt an existing method [TUGM22] to work for our SparseBTF model. For brevity, we illustrate this using linear interpolation in the angular and spatial domains. More sophisticated BTF interpolation methods have been proposed, e.g., [RSK13; GK18], and we see that the SparseBTF representation supports extensions to using more advanced filtering as part of future work. Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two distinct BTFs that are represented using a common dictionary $\{\mathbf{U}^{(1,k)}, \mathbf{U}^{(2,k)}, \mathbf{U}^{(3,k)}, \mathbf{U}^{(4,k)}\}$. Moreover, let $\mathcal{B}_3$ be the result of the interpolation between $\mathcal{B}_1$ and $\mathcal{B}_2$. Then,

$$\mathcal{B}_3 = ((1-\alpha)\mathcal{S}_1 + \alpha\mathcal{S}_2) \times_1 \mathbf{U}^{(1,1)} \times_2 \mathbf{U}^{(2,1)} \times_3 \mathbf{U}^{(3,1)} \times_4 \mathbf{U}^{(4,1)}, \quad (7)$$

where α is the interpolation parameter, $\mathcal{S}_1$ and $\mathcal{S}_2$ are the sparse coefficients of $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. Indeed, $\mathcal{B}_3$ can be obtained using bi-linear, cubic, barycentric, or other variants of interpolation based on the number of BTFs to be interpolated and the smoothness of the interpolation. Moreover, it is possible to interpolate between two or more BTFs that use distinct dictionaries. Assume that $\mathcal{B}_1$ uses the kth dictionary and $\mathcal{B}_2$ uses the pth dictionary. For this case, we have

$$\left((1-\alpha)\mathcal{S}_1 + \alpha\tilde{\mathcal{S}}_2\right) \times_1 \mathbf{U}^{(1,k)} \times_2 \mathbf{U}^{(2,k)} \times_3 \mathbf{U}^{(3,k)} \times_4 \mathbf{U}^{(4,k)}, \quad (8)$$

where

$$\tilde{\mathcal{S}}_2 = \mathcal{S}_2 \times_1 \mathbf{R}^{(1,k)\to(1,p)} \times_2 \mathbf{R}^{(2,k)\to(2,p)} \times_3 \mathbf{R}^{(3,k)\to(3,p)} \times_4 \mathbf{R}^{(4,k)\to(4,p)}, \quad (9)$$

and $\mathbf{R}^{(n,k)\to(n,p)}$ defines a transformation matrix from the orthogonal matrix $\mathbf{U}^{(n,k)}$ to $\mathbf{U}^{(n,p)}$.

4. Results

In this section, we evaluate SparseBTF and compare it with the work of Rainer et al. [RGJW20; RJGW19] as well as NeuMIP [KMX*21]. It is important to note that two of these methods [RJGW19; KMX*21] require re-training for each BTF, hence overfitting the model to the data. Thus, a direct comparison between these two methods is not feasible. However, we have included the results of [RJGW19] as a baseline in Table 3. Due to the fact that NeuMIP [KMX*21] involves additional steps, such as creating a level-of-detail pyramid and performing displacement mapping, which are outside the scope of this paper, our comparison with NeuMIP is limited to their model without the neural offset

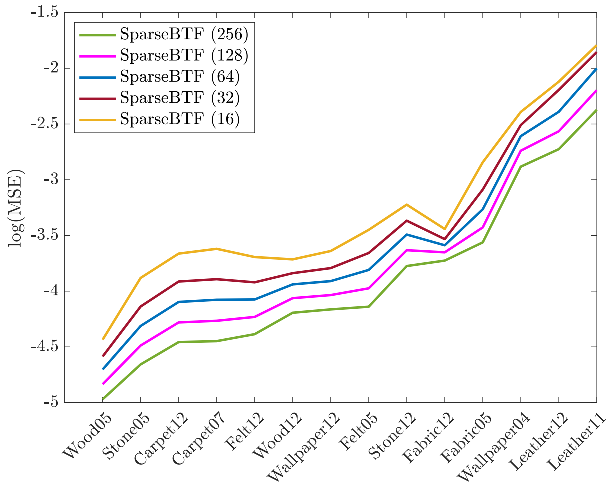

Figure 3: *Logarithm of MSE of 14 reconstructed BTFs using our method with different testing sparsities, i.e., the number of non-zero coefficients.*

	Compression ratio	Absolute size
SparseBTF (16)	4218	4.94
SparseBTF (32)	2122	9.84
SparseBTF (64)	1065	19.60
SparseBTF (128)	506	41.24
SparseBTF (256)	260	80.31
Full BTF	-	20875

Table 4: *Compression ratio and absolute size (MB) of different variations of SparseBTF. The absolute size includes the dictionary and sparse coefficients.*

	Carpet12	Felt05	Stone05
SparseBTF (32)	**1.0**	**2.0**	**0.6**
PCA (64)	2.2	3.5	1.0

Table 5: *Mean squared reconstruction error ($\times 10^{-4}$) of three BTFs in the test set with 100×100 texels. Note that the compression ratio for SparseBTF is twice that of the PCA.*

module. Applying displacement mapping to SparseBTF is left for future work.

Data set and parameters. For training the 4D dictionary ensemble, we chose 18 BTFs out of 84 from the UBO2014 data set [WGK14] as our training set. The training BTFs were selected to cover the variation over the different BTF classes in the UBO2014 data set as well as inter-class variations. We provide more details regarding the materials included in the training set in the supplementary material.

The UBO2014 data set has a spatial resolution of 400×400 and an angular resolution of 151×151, with 3 color channels (RGB). To enable faster processing and to exploit angular correlations, we create 4D data points from each BTF, consisting of all the 151×151 angular samples with a spatial patch size of 10×10 pixels. Hence, every single BTF in UBO2014 is represented by $40 \times 40 = 1600$ data points. We trained 8 dictionaries with training sparsities of $\tau_{l_Y} = 256$, $\tau_{l_U} = 32$, and $\tau_{l_V} = 32$ for the Y, U, and V color channels, respectively. To reduce memory cost during training, we randomly sampled 10% of the training data points. Hence the total number of data points used for training was $(1600 \times 18 \times 0.1) = 2880$. The one-time training takes approximately 101 hours on a machine with 128 cores operating at 2.4GHz. Improvement of the training time is possible, as pre-clustering has been shown to significantly improve the training time by 10× with the same number of dictionaries [MHU19]. The time required for sparse representation of each BTF, as described in Section 3.3, is about 1060 seconds.

To evaluate our method in comparison to [RGJW20], the 12th BTF of each material class, namely *Carpet12, Fabric12, Felt12, Leather12, Stone12, Wallpaper12, Wood12*, are chosen as the test set. Additionally, we have set the sparsity parameter of our method so that the storage cost of both methods exactly match. Throughout this section, we use SparseBTF (16), SparseBTF (32), SparseBTF (64), SparseBTF (128), and SparseBTF (256) to refer to the storage cost of SparseBTF with respect to [RGJW20] with 16, 32, 64, 128, and 256 latent variables, respectively. The compression ratio and the total storage cost for SparseBTF is outlined in Table 4. Note that since we have to account for the location of nonzero coefficients, the membership vector, and the 4D dictionary ensemble, the number of coefficients for SparseBTF is different from that of [RGJW20]. We elaborate on the calculation of our sparsity parameter, τ_t, to best match the storage cost of the baseline method in the supplementary document.

Full BTF Reconstruction. Since Rainer et al. [RGJW20] evaluated their method with 100×100 BTF texels, for a fair comparison, we also crop the BTFs for the results reported in Table 3. We evaluated SparseBTF, in comparison to previous work, using a test set with seven unseen BTFs (materials not in the training data). Table 3 reports the reconstruction error in terms of mean squared error (MSE) with respect to the number of coefficients. The average reconstruction error of SparseBTF reduces consistently with an increasing number of coefficients. This is in contrast to the existing methods [RGJW20; RJGW19], where the error does not decrease considerably for 64 and 128 latent variables; in some cases, the error even increases as the number of coefficients is doubled. Moreover, it should be noted that the encoder of Rainer et al. [RGJW20] is trained on 77 BTFs, whereas our training set includes only 18 BTFs. According to Table 3, SparseBTF performs significantly better in the vast majority of test cases when compared to the method of Rainer et al. [RGJW20]. Table 5 compares SparseBTF to PCA for three BTFs from the UBO2014 dataset [WGK14]. SparseBTF achieves lower MSE values, indicating better reconstruction accuracy than PCA. Note that the SparseBTF compression ratio is twice that of the PCA, showcasing its higher compression efficiency.

Figure 3 confirms the flexibility of SparseBTF, showing a trade-off between memory footprint and reconstruction quality can be obtained through the user-defined sparsity parameter τ_t. The plot includes seven more BTFs in addition to the test set used for Table 3. As demonstrated, SparseBTF shows a consistent increase in

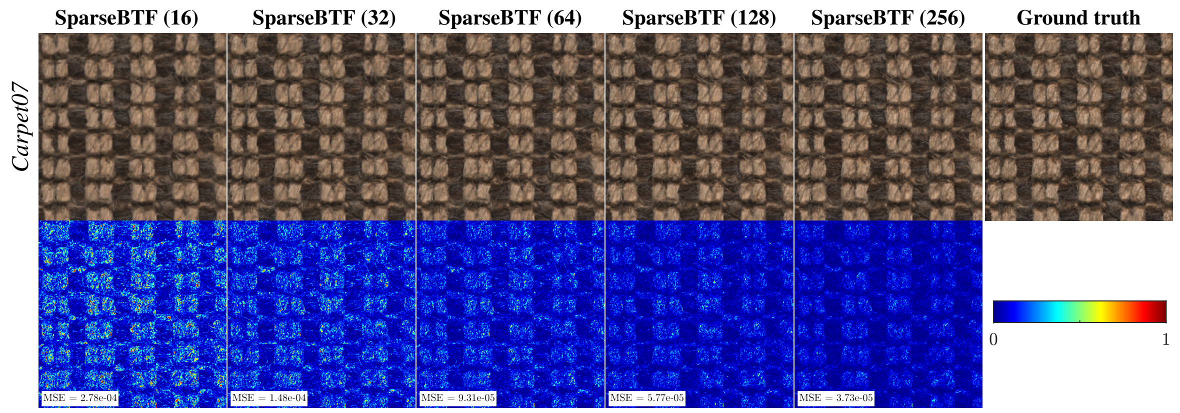

Figure 4: *Demonstration of quality-storage cost tradeoff using our method with different testing sparsities, i.e., the number of non-zero coefficients. Incident azimuth and elevation angles:* 270°, 66.5°. *Observation azimuth and elevation angles:* 15°, 79°.

quality as the number of parameters is increased. Visual comparisons for the effect of sparsity on the image quality are illustrated in Figure 4. To facilitate comparison, all false-color absolute error images in the paper are normalized and scaled by a factor of 2.

Figure 5 shows the reconstructed images of a specific pair of light and view angles produced by SparseBTF and [RGJW20], along with error images. SparseBTF consistently outperforms [RGJW20] in visual quality by preserving high-frequency details while the baseline method tends to smooth images and fails to recover fine texture details, see e.g., *Fabric12* and *Leather12*. For display purposes, the images in figures 4, 5, and 6 are gamma-mapped using $I_{out} = aI_{in}^{-\gamma}$, where $a = 1.3$ and $\gamma = 1.8$.

Table 6 presents the average peak signal-to-noise ratio (PSNR) values for three reconstructed BTFs using SparseBTF and the method of Wu et al. [WDR11], which utilizes analytical dictionaries. For a fair comparison, we utilize the same data set used therein, i.e. the UBO2003 data set [SSK03] with $81 \times 81 = 6561$ angular images and a spatial resolution of 256×256. To establish a baseline, we processed all color channels together to obtain a 5D representation of BTFs. The resulting 5D data points were of size $8 \times 8 \times 81 \times 81 \times 3$, where $m_1 = m_2 = 8$, $m_3 = m_4 = 81$, $m_5 = 3$; the last dimension, i.e. m_5, denotes the number of color channels. In line with the methodology of Wu et al. [WDR11], we train one dictionary per material with $\tau_l = 128$. Since $\tau_t \geq \tau_l$, we set $\tau_t = 965$ for each data point. The results, as shown in Table 6, indicate that SparseBTF significantly outperforms Wu et al. [WDR11] in terms of both reconstruction quality and storage cost. The proposed method by Wu et al. [WDR11] can be adopted for real-time rendering, but we note that this may not be straightforward as the residuals must be also compressed individually to obtain low representation error.

Inter-BTFs blending. Our method is capable of interpolations between two or more BTFs in their sparse representation as explained in Section 3.5. Figure 6 presents our interpolation results for three BTF pairs, with an interpolation parameter ranging from 0.0 to 1.0 with steps of 0.1. Our interpolation results show smooth transitions between BTFs, which is suitable for BTF synthesis using existing data sets. For simplicity, we have shown only linear interpolation

	Wool	Corduroy	Pulli
SparseBTF PSNR (dB)	**20.97**	**21.15**	**21.90**
[WDR11] PSNR (dB)	20.27	19.60	17.86
SparseBTF Size (MB)	**4.71**	**4.71**	**4.71**
[WDR11] Size (MB)	14.93	19.62	11.28

Table 6: *Comparing SparseBTF and [WDR11] on [SSK03] data set. The SparseBTF size includes the dictionary and sparse coefficients.*

between two BTFs; however, our algorithm allows utilizing more than two BTFs and different interpolation methods (e.g. spline or cubic) as described in Section 3.5.

Rendering. We implemented SparseBTF in a progressive path tracer [Kaj86; PJH16] built upon the NVIDIA OptiX framework [PBD*10] and CUDA textures. The framework calls the shader for each ray intersection and uses Eq. (5) to reconstruct each pixel locally in real-time resulting in 200 frames per second (fps) on a GeForce RTX 3080 for SparseBTF (32). We observed that interpolation in the coefficient space using Eq. (6) improves frame rates by a factor of 4. Additionally, since SparseBTF provides local and random access to each texel in the BTF, it enables real-time adjustment of compression ratio for optimal speed and reconstruction quality. Our patch design to incorporate all angular information is shown to be effective in mitigating any flickering artifacts when the light or view direction changes. Moreover, the application of a Gaussian kernel for interpolation further enhances the overall visual fidelity by ensuring that no artifacts are perceptible to the viewer. The real-time rendering of multiple materials is provided in the supplementary video.

We used the Mitsuba renderer [Jak10; NVZJ19] to generate the renderings in Figures 7 and 8 with resolutions of 400×400 and 960×540, respectively. Figure 7 presents the renderings of SparseBTF and NeuMIP at the same storage cost, along with the corresponding MSE and structural similarity (SSIM) values. It is important to note that we trained NeuMIP without the neural offset module to ensure a fair comparison. As NeuMIP employs only

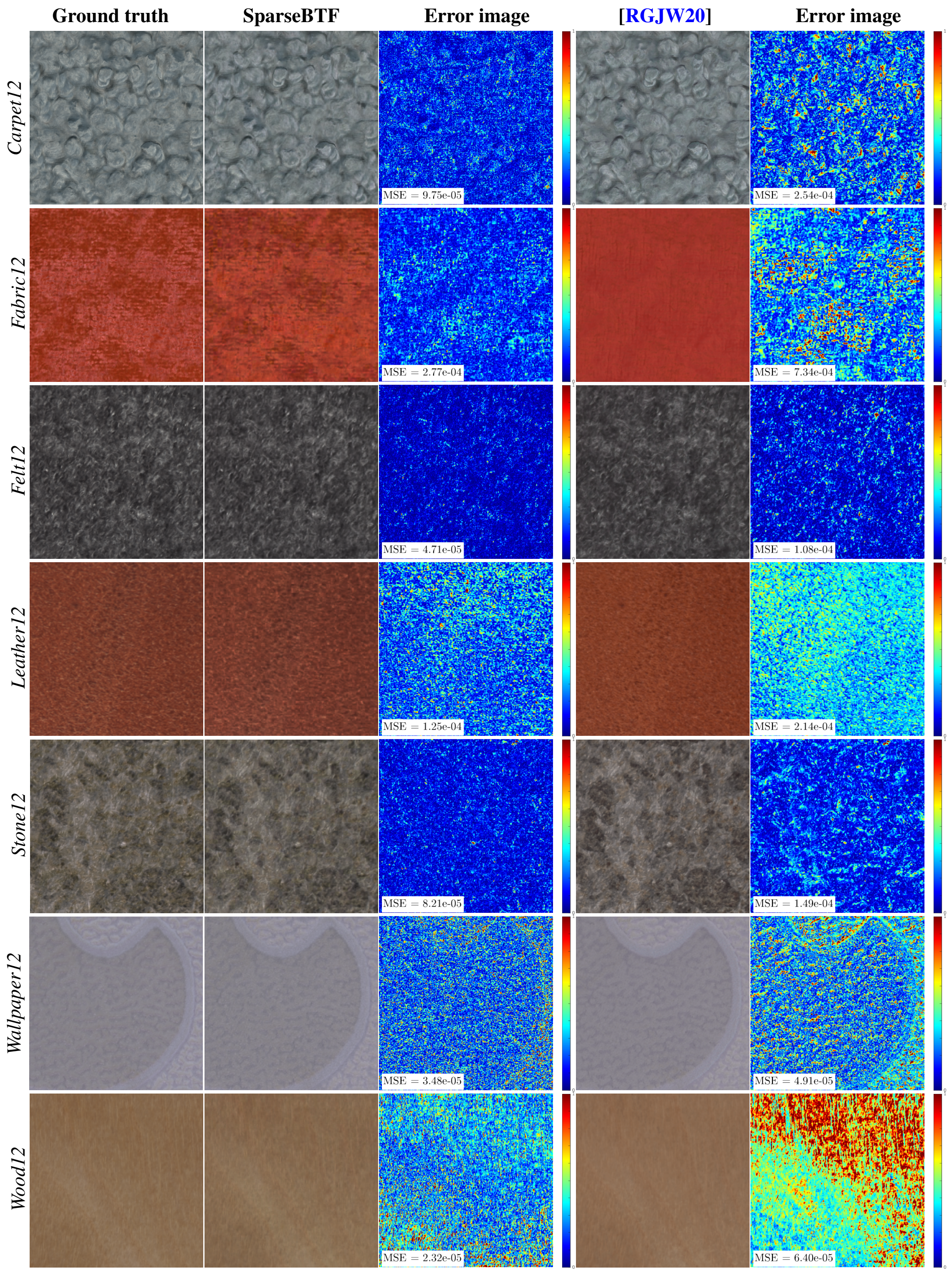

Figure 5: *Comparison of reconstructed BTFs using SparseBTF (32) and [RGJW20] (32). Incident azimuth and elevation angles:* 0°, 90°. *Observation azimuth and elevation angles:* 15°, 30°.

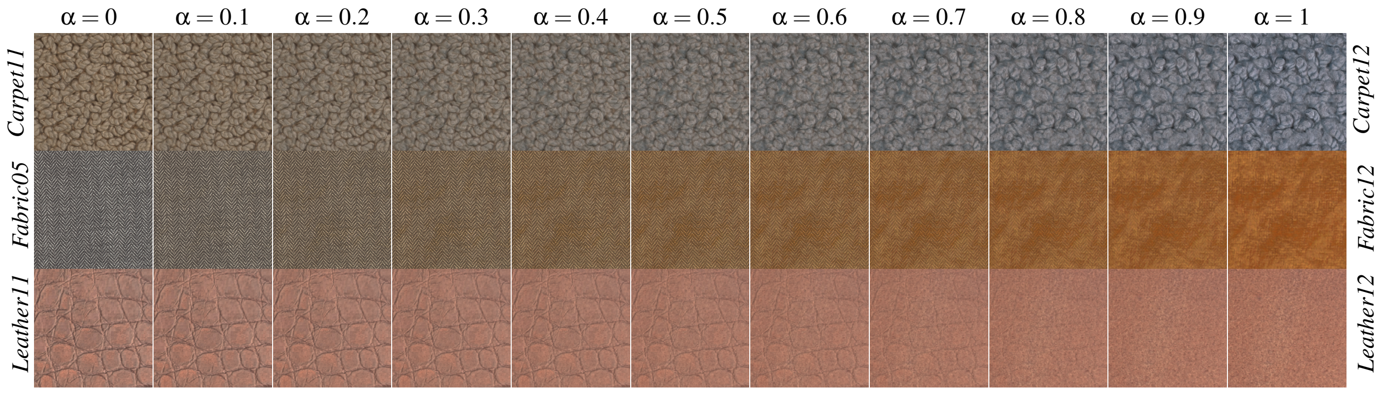

Figure 6: *BTF interpolation results with different interpolation coefficients, α, ranging from 0 to 1 from left to right. Incident azimuth and elevation angles:* 270°, 66.5°. *Observation azimuth and elevation angles:* 15°, 79°.

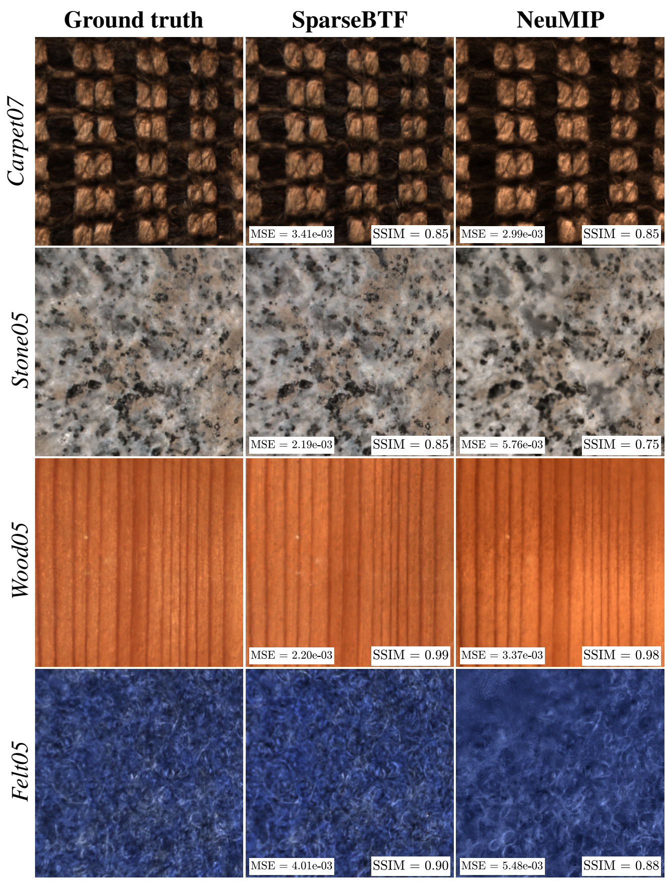

Figure 7: *Comparison of renderings using SparseBTF (64) and NeuMIP [KMX*21]. Model, view and lighting directions were the same for all these renderings.*

400 different combinations of light and camera directions for training, the model may not be able to fully capture the intricate details of the BTFs leading to a blurry representation of the appearance of the material when rendered from certain angles. In contrast, SparseBTF can capture the complex reflectance behavior of materials using a common dictionary. Figure 8 illustrates a comparison between the renderings produced by SparseBTF and the method of Rainer et al. [RGJW20]. All images are rendered with 1024 samples per pixel, using the Pixar Campus environment map with fixed camera positions. SparseBTF successfully recovers complex and high-frequency texture patterns, showcasing the effectiveness of our model.

5. Limitations and Future Work

One of the limitations of our model is the assumption of identical angular resolution within the data points comprising the training and the testing sets. If the data dimensionality changes, one has to re-train the ensemble. Alternatively, we can train multiple 4D dictionaries ensembles, one for each data set with a distinct angular resolution. By storing an index for each data set, we can choose the appropriate ensemble during rendering, with minimal impact on storage costs. In this direction, one can also define variable patch sizes for the spatial dimension. Adaptively setting the patch size based on the complexity of the spatial domain is expected to increase the compression ratio and the representation quality. Complex regions of the texture will require smaller patch sizes, and possibly a large number of coefficients, while for smooth regions we can achieve acceptable quality with large patch sizes and a smaller number of coefficients.

In this work, we have used a fixed sparsity value for training. Our experimental results demonstrate that SparseBTF exhibits a favorable tradeoff between quality and storage cost, resulting in reduced sensitivity to overfitting and underfitting. However, having a fixed training sparsity for all the dictionaries may lead to some level of overfitting or underfitting depending on the BTF. Since SparseBTF utilizes an ensemble of 4D dictionaries, it is interesting to explore the possibility of optimizing the training sparsity during the training by utilizing a validation set. Ideally, the training method should learn an optimal sparsity value for each 4D dictionary in the ensemble to minimize overfitting and underfitting. Moreover, tailoring the representation model to different classes of BTFs in the training set

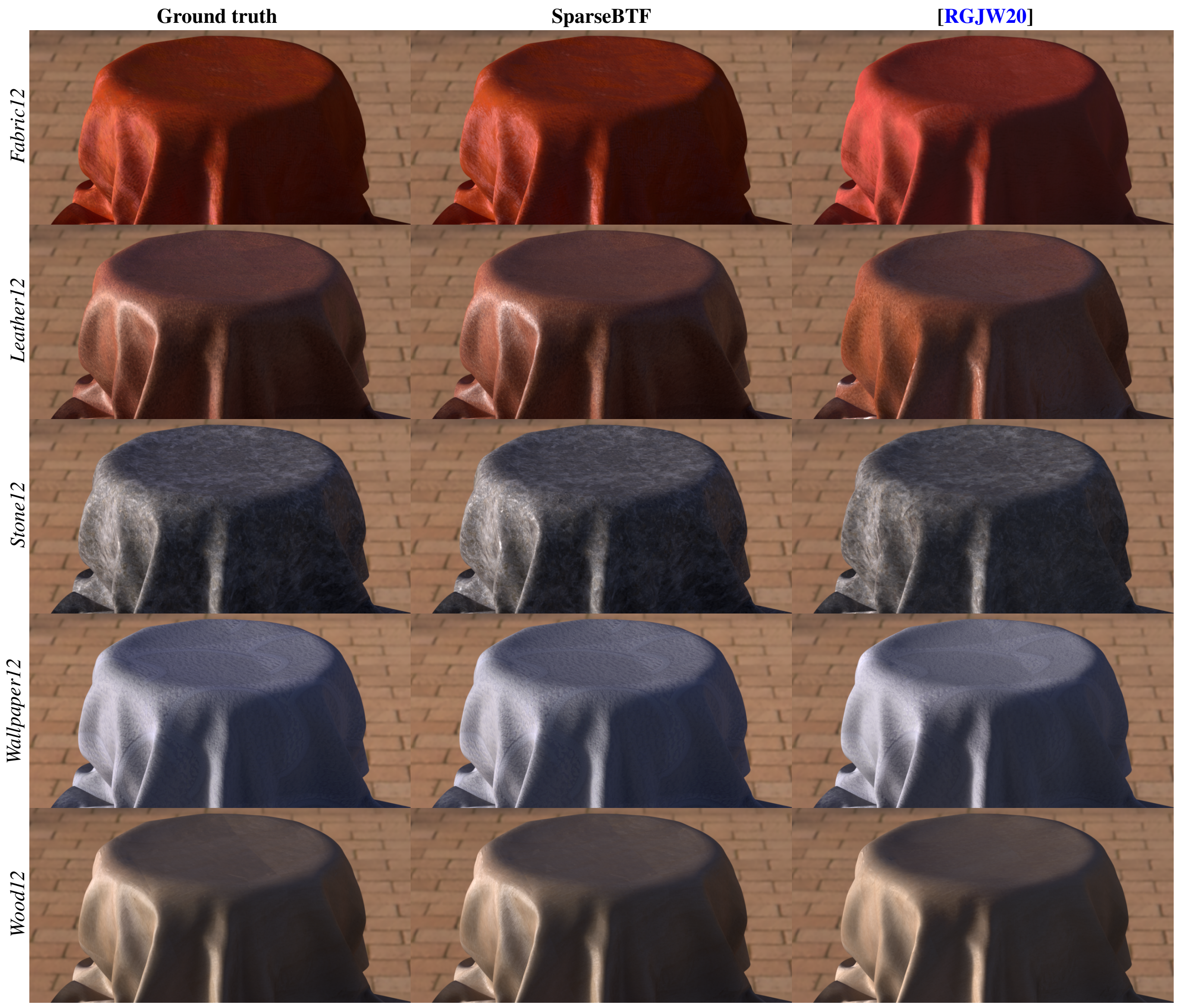

Figure 8: *Comparison of renderings using SparseBTF (32) and [RGJW20] (32). Model, view and lighting environment were the same for all these renderings.*

can reduce storage costs and increase the rendering speed and the representation quality.

Due to their limited spatial resolution, BTFs need to be tiled across surfaces. While this paper does not focus on exploring tiling methods, we acknowledge that simply repeating the BTFs can result in visible block artifacts at certain angles. A promising direction for future work involves investigating texture tiling and blending techniques that can preserve complex correlations within the data [LPF*07; LSD23]. These methods could address the issue of tiling artifacts and improve the overall quality of BTF renderings.

Furthermore, future research can explore the effect of various pre-clustering techniques on the quality of reconstruction and training speed. The empirical results of Miandji et al. [MHU19] demonstrate that a sparsity-based pre-clustering algorithm not only expedites the training process but can also improve the quality of the representation. Indeed, such pre-clustering algorithms require a larger training set than what was utilized in our experiments.

6. Conclusion

In this paper, we presented SparseBTF, a novel learning-based sparse model for efficient representation of BTFs, that enables compact storage and real-time rendering. We achieve this by a 4D dictionary ensemble that is trained once and can be used for a wide range of BTFs in a test set. By projecting a BTF in the test set on the most suitable dictionary in the ensemble, followed by sparsifying the coefficients, we obtain a representation of the BTF with a low memory footprint. We presented the demarcation of our method from previous approaches. Specifically, while SparseBTF fulfills the requirements of low memory footprint, high quality-storage cost tradeoff, real-time rendering, and a one-time training process, previous methods address only a subset of such criteria. Our results demonstrate a significantly higher image quality as compared with previous work when we use the same storage cost. More importantly, we achieve this while enabling real-time performance, which is not supported by the state-of-the-art methods we compare with. Finally, we propose a novel angular interpolation algorithm for BTFs that can be directly performed in the coefficient space of SparseBTF.

Acknowledgments

We would like to thank Gilles Rainer for his tremendous help with the comparisons, as well as Alexandr Kuznetsov for sharing the code for NeuMIP. This work is a part of PRIME, which is funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 956585.

References

[AEB06] Aharon, Michal, Elad, Michael, and Bruckstein, Alfred M. "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation". *IEEE Transactions on Signal Processing* 54.11 (2006), 4311–4322. doi: 10.1109/TSP.2006.881199 3.

[CD01] Cula, Oana G. and Dana, Kristin J. "Compact representation of bidirectional texture functions". *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2001. doi: 10.1109/CVPR.2001.990645 2.

[dBWK15] Den Brok, Dennis, Weinmann, Michael, and Klein, Reinhard. "Linear models for material BTFs and possible applications". *Workshop on Material Appearance Modeling*. Eurographics Association, 2015, 15–19. doi: 10.2312/mam.20151198 3.

[dBWK18] Den Brok, Dennis, Weinmann, Michael, and Klein, Reinhard. "Rapid material capture through sparse and multiplexed measurements". *Computers & Graphics* 73 (2018), 26–36. doi: 10.1016/j.cag.2018.03.003 3.

[DvGNK99] Dana, Kristin J., van Ginneken, Bram, Nayar, Shree K., and Koenderink, Jan J. "Reflectance and texture of real-world surfaces". *ACM Transactions on Graphics* 18.1 (1999), 1–34. doi: 10.1145/300776.300778 2.

[FH09] Filip, Jiří and Haindl, Michal. "Bidirectional texture function modeling: a state of the art survey". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.11 (2009), 1921–1940. doi: 10.1109/TPAMI.2008.246 2.

[FKH*18] Filip, Jiří, Kolafová, Martina, Havlíček, Michal, Vávra, Radomír, Haindl, Michal, and Rushmeier, Holly. "Evaluating physical and rendered material appearance". *The Visual Computer* 34.6–8 (June 2018), 805–816. issn: 0178-2789. doi: 10.1007/s00371-018-1545-3 2.

[FWH*22] Fan, Jiahui, Wang, Beibei, Hasan, Milos, Yang, Jian, and Yan, Ling-Qi. "Neural layered BRDFs". *SIGGRAPH 2022 Conference Papers*. ACM, 2022, 4:1–4:8. doi: 10.1145/3528233.3530732 3.

[GK18] Golla, Tim and Klein, Reinhard. "Interactive Interpolation of Metallic Effect Car Paints". *Vision, Modeling and Visualization*. Ed. by Beck, Fabian, Dachsbacher, Carsten, and Sadlo, Filip. The Eurographics Association, 2018. isbn: 978-3-03868-072-7. doi: 10.2312/vmv.20181248 7.

[HF13] Haindl, Michal and Filip, Jiří. *Visual Texture: Accurate Material Appearance Measurement, Representation and Modeling*. Springer, 2013. doi: 10.1007/978-1-4471-4902-6 2.

[HFM10] Havran, Vlastimil, Filip, Jiří, and Myszkowski, Karol. "Bidirectional texture function compression based on multi-level vector quantization". *Computer Graphics Forum* 29.1 (2010), 175–190. doi: 10.1111/j.1467-8659.2009.01585.x 3.

[HFV12] Haindl, Michal, Filip, Jiří, and Vávra, Radomír. "Digital material appearance: the curse of tera-bytes". *ERCIM News* 90 (2012), 49–50 2.

[HGC*20] Hu, Bingyang, Guo, Jie, Chen, Yanjun, Li, Mengtian, and Guo, Yanwen. "DeepBRDF: a deep representation for manipulating measured BRDF". *Computer Graphics Forum* 39.2 (2020), 157–166. doi: 10.1111/cgf.13920 3.

[Jak10] Jakob, Wenzel. *Mitsuba renderer*. 2010. url: http://www.mitsuba-renderer.org 9.

[Kaj86] Kajiya, James T. "The rendering equation". *Computer Graphics (SIGGRAPH '86)* 20.4 (1986), 143–150. doi: 10.1145/15922.15902 9.

[KMBK03] Koudelka, Melissa L, Magda, Sebastian, Belhumeur, Peter N., and Kriegman, David J. "Acquisition, compression, and synthesis of bidirectional texture functions". *3rd International Workshop on Texture Analysis and Synthesis (Texture 2003)*. 2003, 59–64 2.

[KMX*21] Kuznetsov, Alexandr, Mullia, Krishna, Xu, Zexiang, Hašan, Miloš, and Ramamoorthi, Ravi. "NeuMIP: multi-resolution neural materials". *ACM Transactions on Graphics* 40.4 (July 2021), 175:1–175:13. doi: 10.1145/3450626.3459795 2, 3, 5–7, 11.

[KWK17] Krumpen, Stefan, Weinmann, Michael, and Klein, Reinhard. "OctreeBTFs – A compact, seamless and distortion-free reflectance representation". *Computers & Graphics* 68 (2017), 21–31. doi: 10.1016/j.cag.2017.08.001 4.

[KWM*22] Kuznetsov, Alexandr, Wang, Xuezheng, Mullia, Krishna, Luan, Fujun, Xu, Zexiang, Hasan, Milos, and Ramamoorthi, Ravi. "Rendering neural materials on curved surfaces". *SIGGRAPH 2022 Conference Papers*. ACM, 2022, 9:1–9:9. doi: 10.1145/3528233.3530721 3.

[LHZ*04] Liu, Xinguo, Hu, Yaohua, Zhang, Jingdan, Tong, Xin, Guo, Baining, and Shum, Heung-Yeung. "Synthesis and rendering of bidirectional texture functions on arbitrary surfaces". *IEEE Transactions on Visualization and Computer Graphics* 10.3 (2004), 278–289. doi: 10.1109/TVCG.2004.1272727 2.

[LPF*07] Leung, Man-Kang, Pang, Wai-Man, Fu, Chi-Wing, Wong, Tien-Tsin, and Heng, Pheng-Ann. "Tileable BTF". *IEEE transactions on visualization and computer graphics* 13 (Sept. 2007), 953–65. doi: 10.1109/TVCG.2007.1034 12.

[LSD23] Lutz, Nicolas, Sauvage, Basile, and Dischler, Jean-Michel. "Preserving the Autocovariance of Texture Tilings Using Importance Sampling". *Computer Graphics Forum* (2023). issn: 1467-8659. doi: 10.1111/cgf.14766 12.

[MHU19] Miandji, Ehsan, Hajisharif, Saghi, and Unger, Jonas. "A unified framework for compression and compressed sensing of light fields and light field videos". *ACM Transactions on Graphics* 38.3 (May 2019), 23:1–23:18. doi: 10.1145/3269980 5, 8, 12.

[MMK03] Müller, Gero, Meseth, Jan, and Klein, Reinhard. "Compression and real-time rendering of measured BTFs using local PCA". *Vision, Modeling and Visualisation 2003*. Akademische Verlagsgesellschaft Aka GmbH, 2003, 271–280 2.

[MMS*05] Müller, Gero, Meseth, Jan, Sattler, Mirko, Sarlette, Ralf, and Klein, Reinhard. "Acquisition, synthesis, and rendering of bidirectional texture functions". *Computer Graphics Forum* 24.1 (2005), 83–109. doi: 10.1111/j.1467-8659.2005.00830.x 2.

[NJR15] NIELSEN, JANNIK BOLL, JENSEN, HENRIK WANN, and RAMAMOORTHI, RAVI. "On optimal, minimal BRDF sampling for reflectance acquisition". *ACM Transactions on Graphics* 34.6 (Oct. 2015), 186:1–186:11. DOI: 10.1145/2816795.2818085 5.

[NVZJ19] NIMIER-DAVID, MERLIN, VICINI, DELIO, ZELTNER, TIZIAN, and JAKOB, WENZEL. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". *ACM Transactions on Graphics* 38.6 (Dec. 2019), 203:1–203:17. DOI: 10.1145/3355089.3356498 9.

[PBD*10] PARKER, STEVEN G., BIGLER, JAMES, DIETRICH, ANDREAS, et al. "OptiX: a general purpose ray tracing engine". *ACM Transactions on Graphics* 29.4 (July 2010), 66:1–66:13. DOI: 10.1145/1833349.1778803 1, 9.

[PJH16] PHARR, MATT, JAKOB, WENZEL, and HUMPHREYS, GREG. *Physically Based Rendering: From Theory to Implementation*. third. Morgan Kaufmann/Elsevier, 2016 9.

[QP22] QUARTESAN, LUCA and PEREIRA SANTOS, CARLOS. "Neural Bidirectional Texture Function Compression and Rendering". *SIGGRAPH Asia 2022 Posters*. ACM, 2022, 10:1–10:2. DOI: 10.1145/3550082.3564188 3.

[RGJW20] RAINER, GILLES, GHOSH, ABHIJEET, JAKOB, WENZEL, and WEYRICH, TIM. "Unified neural encoding of BTFs". *Computer Graphics Forum* 39.2 (July 2020), 167–178. DOI: 10.1111/cgf.13921 1–3, 5–12.

[RJGW19] RAINER, GILLES, JAKOB, WENZEL, GHOSH, ABHIJEET, and WEYRICH, TIM. "Neural BTF compression and interpolation". *Computer Graphics Forum* 38.2 (Mar. 2019). DOI: 10.1111/cgf.13633 2, 3, 5–8.

[RK09] RUITERS, ROLAND and KLEIN, REINHARD. "BTF compression via sparse tensor decomposition". EGSR '09. Eurographics Association, 2009, 1181–1188. DOI: 10.1111/j.1467-8659.2009.01495.x 3.

[RKAJ08] REINHARD, ERIK, KHAN, ERUM ARIF, AKYUZ, AHMET OGUZ, and JOHNSON, GARRETT. *Color Imaging: Fundamentals and Applications*. first. A K Peters/CRC Press, 2008 4.

[RSK13] RUITERS, ROLAND, SCHWARTZ, CHRISTOPHER, and KLEIN, REINHARD. "Example-based Interpolation and Synthesis of Bidirectional Texture Functions". *Computer Graphics Forum* 32.2pt3 (2013), 361–370. DOI: 10.1111/cgf.12056 7.

[SESM22] SUHAIL, MOHAMMED, ESTEVES, CARLOS, SIGAL, LEONID, and MAKADIA, AMEESH. "Generalizable patch-based neural rendering". *European Conference on Computer Vision (ECCV)*. Springer, 2022. DOI: 10.1007/978-3-031-19824-3_10 5.

[Sil86] SILVERMAN, B. W. *Density Estimation for Statistical and Data Analysis*. Chapman & Hall, 1986. DOI: 10.1201/9781315140919 7.

[SRRW21] SZTRAJMAN, ALEJANDRO, RAINER, GILLES, RITSCHEL, TOBIAS, and WEYRICH, TIM. "Neural BRDF representation and importance sampling". *Computer Graphics Forum* 40.6 (2021), 332–346. DOI: 10.1111/cgf.14335 3.

[SSK03] SATTLER, MIRKO, SARLETTE, RALF, and KLEIN, REINHARD. "Efficient and realistic visualization of cloth". *Eurographics Workshop on Rendering (EGWR '03)*. Eurographics Association, 2003, 167–177. ISBN: 3905673037. DOI: 10.2312/EGWR/EGWR03/167-177 2, 9.

[SWRK11] SCHWARTZ, CHRISTOPHER, WEINMANN, MICHAEL, RUITERS, ROLAND, and KLEIN, REINHARD. "Integrated high-quality acquisition of geometry and appearance for cultural heritage". *VAST: International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*. Eurographics Association, 2011. DOI: 10.2312/VAST/VAST11/025-032 4.

[TFLS11] TSAI, YU-TING, FANG, KUEI-LI, LIN, WEN-CHIEH, and SHIH, ZEN-CHUNG. "Modeling bidirectional texture functions with multivariate spherical radial basis functions". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.7 (2011), 1356–1369. DOI: 10.1109/TPAMI.2010.211 3.

[TS12] TSAI, YU-TING and SHIH, ZEN-CHUNG. "K-clustered tensor approximation: a sparse multilinear model for real-time rendering". *ACM Transactions on Graphics* 31.3 (June 2012), 19:1–19:17. DOI: 10.1145/2167076.2167077 3.

[Tsa15] TSAI, YU-TING. "Multiway K-clustered tensor approximation: toward high-performance photorealistic data-driven rendering". *ACM Transactions on Graphics* 34.5 (Nov. 2015), 157:1–157:15. DOI: 10.1145/2753756 3.

[TUGM22] TONGBUASIRILAI, TANABOON, UNGER, JONAS, GUILLEMOT, CHRISTINE, and MIANDJI, EHSAN. "A sparse non-parametric BRDF model". *ACM Transactions on Graphics* 41.5 (Oct. 2022), 181:1–181:18. DOI: 10.1145/3533427 3, 5, 7.

[VT04] VASILESCU, M. ALEX O. and TERZOPOULOS, DEMETRI. "TensorTextures: multilinear image-based rendering". *ACM Transactions on Graphics* 23.3 (Aug. 2004), 336–342. DOI: 10.1145/1015706.1015725 3.

[WDR11] WU, HONGZHI, DORSEY, JULIE, and RUSHMEIER, HOLLY. "A sparse parametric mixture model for BTF compression, editing and rendering". *Computer Graphics Forum* 30.2 (2011), 465–473. DOI: 10.1111/j.1467-8659.2011.01890.x 3, 9.

[WGK14] WEINMANN, MICHAEL, GALL, JUERGEN, and KLEIN, REINHARD. "Material classification based on training data synthesized using a BTF database". *Computer Vision – ECCV 2014*. Springer, 2014, 156–171. DOI: 10.1007/978-3-319-10578-9_11 1, 2, 5, 8.

[WWS*05] WANG, HONGCHENG, WU, QING, SHI, LIN, YU, YIZHOU, and AHUJA, NARENDRA. "Out-of-core tensor approximation of multidimensional matrices of visual data". *ACM Transactions on Graphics* 24.3 (July 2005), 527–535. DOI: 10.1145/1073204.1073224 3.

[XSJT07] XU, LEILEI, SUN, HANQIU, JIA, JIAYA, and TAO, CHENJUN. "Dynamic texture synthesis in the YUV color-space". *Entertainment Computing – ICEC 2007*. Springer, 2007, 243–248. DOI: 10.1007/978-3-540-74873-1_29 4.

[ZZW*21] ZHENG, CHUANKUN, ZHENG, RUZHANG, WANG, RUI, ZHAO, SHUANG, and BAO, HUJUN. "A compact representation of measured BRDFs using neural processes". *ACM Transactions on Graphics* 41.2 (Nov. 2021), 14:1–14:15. DOI: 10.1145/3490385 3.