

A Dynamic By-example BTF Synthesis Scheme

Supplemental Material

1 IMPLEMENTATION

Training. We implement our network training on PyTorch [Paszke et al. 2019]. The network structure is shown in Fig. 1. We apply AdamW with the default setting for the optimization process. The learning rate for the feature planes is $1e^{-3}$ and $3e^{-4}$ for the MLP. In Table 1, we summarize the size of each feature plane, which follows Biplane [Fan et al. 2023]’s setting. Although the 20×20 resolution for directional planes may seem small, it is sufficient for UBO2014 [Weinmann et al. 2014] dataset with 151×151 angular resolution. The initialization strategy for the feature plane does not have a great impact on the result and we use the method from He et al. [2015]. We circularly pad the phi axis of the angular feature planes ($f^{(H)}$ and $f^{(D)}$) to avoid discontinuity in the angular domain. We use Leaky ReLU for all layers in the MLP. In each iteration, we use a batch of 2,560,000 ($16 \times 400 \times 400$) samples, i.e., 16 different BTF images, and each BTF image is under the same viewing and lighting condition. We train the model for 50 epochs, and after each epoch, the learning rate will decrease by a factor of 0.9. Although the loss will continue decreasing after the 50th epoch, the improvement gain is minimal. Thus, early stopping at the 50th epoch is sufficient. The whole training takes about 2 hours on a single RTX 4090 GPU.

Rendering Integration. Our rendering is performed on the top of the NVIDIA Falcor [Kallweit et al. 2022] renderer using its Path-Tracer pass with our modified BTF material. The inference of MLPs is implemented in Slang [He et al. 2018] shading language. We reorganized the matrix multiplication of MLP inference into a series of float4x4 \times float4 inside the shader. The MLP’s biases are also stored as float4. Each neural texture is stored in a 2D float4 texture array. The detailed performance is summarized in Table 2. We need to mention that our MLP inference is done inside the shader without the use of batching or hardware acceleration for neural network inference. Integrating batch inference or utilizing GPU hardware inference acceleration structures, such as Tensor Cores, still has the potential to improve runtime performance.

Importance Sampling. Importance sampling is crucial to efficient rendering. However, we are not focusing on developing a new importance sampling strategy, as we consider it orthogonal to our main objective. Instead, we employ a straightforward cosine-weighted hemisphere sampling to our method. Previous work as predicting a histogram distribution [Xu et al. 2023; Zhu et al. 2021] or using normalizing flows [Xu et al. 2023] can be adapted to our method.

Parallax Effects. Incorporating offset [Kuznetsov et al. 2021] into neural materials provides a method for achieving a pseudo-3D effect on 2D surfaces. To obtain a parallax effect, previous work either implicitly estimates a 4D offset from BTFs [Fan et al. 2023; Kuznetsov et al. 2021] or explicitly creates a 7D synthetic SBTF (Silhouette BTF) dataset for training [Kuznetsov et al. 2022]. Considering the measured BTF datasets do not include the offset data, obtaining the additional parallax effect involves an unstable, unsupervised

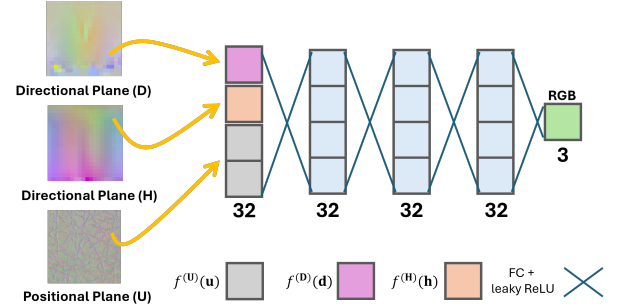


Fig. 1. Our Triple Plane’s network structure. It is a lightweight MLP with 4 fully connected (FC) layers. It takes an input of positional feature and directional features then outputs the RGB reflectance. We apply Leaky ReLU activation function to each layer.

learning process [Kuznetsov et al. 2021]. Consequently, we chose to exclude parallax effects from our method, a decision that does not impact the validity of our conclusions.

Avoiding Cuts in Synthesis. By-example texture synthesis doesn’t require the input texture to be seamlessly tileable. However, for some specific *implementation*, which may require seamlessly tileable 2D texture. Otherwise, it will create some discontinuous cuts because the synthesis process may choose some patches that cross the example’s edges. In our case, we need the input BTF to be seamlessly tileable to avoid these cuts. We employ a trivial method that simply blends a small area around the edges to make BTF’s positional plane seamlessly tileable without claiming the credit. as illustrated in Fig. 2. Moreover, making 6D BTF tileable in this way is only applicable within our proposed BTF synthesis scheme since blending the original 6D BTF is not trivial due to the high dimensionality.

2 LIMITATION

Structure-persevering Synthesis. One of the limitations we have is that we directly apply the existing by-example texture synthesis methods for the BTF synthesis. Therefore, we inherit the same limitations from the chosen texture synthesis approach. As shown in Fig. 3, our method can not handle a highly structured BTF since the used texture synthesis methods can not. Potential improvement may be obtained by applying a better texture synthesis strategy that can preserve the structures.

REFERENCES

- Jiahui Fan, Beibei Wang, Miloš Hašan, Jian Yang, and Ling-Qi Yan. 2023. Neural Biplane Representation for BTF Rendering and Acquisition. In *Proceedings of SIGGRAPH 2023*.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.
- Yong He, Kayvon Fatahalian, and Tim Foley. 2018. Slang: language mechanisms for extensible real-time shading systems. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Table 1. This table summarizes the dimensions of feature planes. The positional feature plane is set to match the dimension of BTF data [Weinmann et al. 2014]. The directional plane’s resolution follows Biplane [Fan et al. 2023]’s setting. Our decomposition method only takes 10 MB storage for each 6D BTF, which originally needed hundreds of GBs without compression.

	Height	Width	Channels	Storage
$f^{(U)}$	400	400	16	10 MB
$f^{(H)}$	20	20	8	12.5 KB
$f^{(D)}$	20	20	8	12.5 KB
Network parameters				12.8 KB

Table 2. This table summarizes the performance of our method via a naive implementation of matrix multiplication inside the shader. The Eval. Time includes both neural texture fetching and MLP inference (once per pixel).

Resolution	1920 × 1080 (1 SPP)	2560 × 1440 (1 SPP)	3840 × 2160 (1 SPP)
Eval. Time	2.0 ms	4.8 ms	8.4 ms

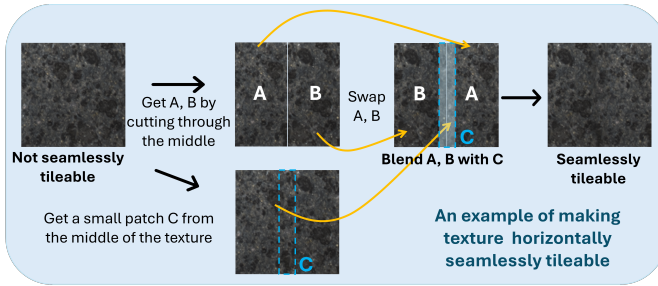


Fig. 2. We first make BTF seamlessly tileable using a simple but effective method. This is an example of making texture horizontally seamlessly tileable, but in order to make BTF fully seamlessly tileable, it needs to be performed twice — first horizontally and then vertically.

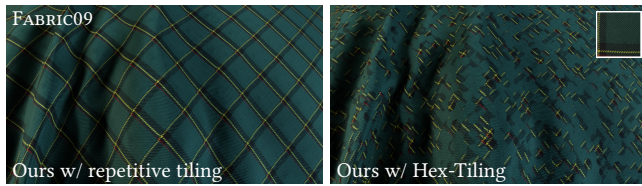


Fig. 3. The example BTF may exhibit highly structured patterns. Applying dynamic texture synthesis methods sometimes ruins the structured patterns. This is an inherent limitation of dynamic texture synthesis. A better texture synthesis strategy may refine it.

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> <https://github.com/NVIDIAGameWorks/Falcor>.
 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>

Alexandr Kuznetsov, Xuezheng Wang, Krishna Mullia, Fujun Luan, Zexiang Xu, Miloš Hasan, and Ravi Ramamoorthi. 2022. Rendering Neural Materials on Curved Surfaces. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) (SIGGRAPH ’22). Association for Computing Machinery, New York, NY, USA, Article 9, 9 pages. <https://doi.org/10.1145/3528233.3530721>
 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
 Michael Weinmann, Juerge Gall, and Reinhard Klein. 2014. Material Classification Based on Training Data Synthesized Using a BTF Database. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 156–171.
 Bing Xu, Liwen Wu, Miloš Hasan, Fujun Luan, Iliyan Georgiev, Zexiang Xu, and Ravi Ramamoorthi. 2023. NeuSample: Importance Sampling for Neural Materials. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) (SIGGRAPH ’23). Association for Computing Machinery, New York, NY, USA, Article 41, 10 pages. <https://doi.org/10.1145/3588432.3591524>
 Junqiu Zhu, Yaoyi Bai, Zilin Xu, Steve Bako, Edgar Velázquez-Armendáriz, Lu Wang, Pradeep Sen, Miloš Hašan, and Ling-Qi Yan. 2021. Neural Complex Luminaires: Representation and Rendering. *ACM Trans. Graph.* 40, 4, Article 57 (jul 2021), 12 pages. <https://doi.org/10.1145/3450626.3459798>