Lightweight Neural Basis Functions for All-Frequency Shading

Supplemental Material

1 DISCUSSION ON BASIS FUNCTIONS

Basis functions are pervasively applied in rendering because of their compact representation and convenient computational properties. However, different type of basis functions has their advantages and limitations. In this section, We evaluate the pros and cons of commonly used basis functions including spherical harmonics (SH), wavelets and spherical Gaussians (SG).

Spherical harmonics (SH) are a set of orthonormal basis functions $Y_{lm}(\boldsymbol{\omega})$ with degree $l = 0, 1, 2, \dots$ and order $m = -l, -l+1, \dots, l$. Therefore, the spherical harmonics up to degree l have $(l+1)^2$ basis functions in total. Besides orthonormality, spherical harmonics have other good mathematical properties such as rotational invariance. However, the biggest problem of SH is their low-frequency nature. Even with hundreds of SH basis functions, we can only represent a relatively low-frequency spherical signal, which makes SH not suitable to preserve sharp details or represent highly glossy materials.

Wavelets are also a group of orthonormal basis functions. They work by performing a 2D wavelet transform from a given function (usually an image) to a set of coefficients. The wavelet transforms itself does not perform any compression, because the number of the resulting coefficients is no less than the number of pixels in the original image. But the coefficients can then be non-linearly thresholded, discarding those close to zero. In this way, with the storage of 0.5%-10% of the resulting coefficients, wavelets are able to faithfully represent a 2D function while keeping its high-frequency details.

However, the non-linearity of wavelets brings about the inconvenience. For the multiple products operation using wavelets, since the positions of the non-zero entries can differ per input function, it is generally not possible to quickly carry out the resulting product in wavelet form. In fact, even for the triple product integral of three wavelet-represented functions, considerable efforts [Ng et al. 2004; Sun and Mukherjee 2006] have to be made for reasonably fast performance, and are still limited to a specific kind of wavelets (Haar wavelets).

The biggest issue of the wavelets is that they do not support efficient rotation. When a function (usually spherical) is rotated, it is difficult to simply perform a rotation on the non-zero coefficients. Instead, the entire function must be first recovered, then rotated, reprojected, and thresholded again. Efforts have been devoted to alleviate the rotation issue [Wang et al. 2006], but the time complexity is still high, diluting the practicality of wavelet basis functions.

Note specifically that the percentage of coefficients kept in the non-linear wavelet transform is significantly *different from the actual compression rate*, since the coefficients must be sparsely stored together with their indices. Therefore, without any further quantization, the storage of wavelet coefficients should at least be doubled (which is used throughout this paper) from the number of coefficients, if not tripled for 2D images with 2D indices. For a fair comparison, in terms of storage, we also do not use any quantization, such as float32 to float16 compression (we only use it for faster inference), and simply count the number of coefficients.

Spherical Gaussians (SG) are a widely used type of Spherical Radial Basis Functions (SRBF). They are simple extensions of the Gaussian Mixture Model (GMM) onto the surface of a unit sphere. It is generally believed that spherical Gaussians are able to represent all frequency contents with very few number of basis functions. However, they are not suitable to represent complex shapes of 2D visibility functions. Moreover, the spherical Gaussians are usually optimized via the Expectation-Maximization (EM) process, and therefore not temporally stable and non-linear. And spherical Gaussians are not orthonormal, indicating not only an $O(n^2)$ DPI performance, where n is the number of selected basis functions, but also a squared number of resulting spherical Gaussians, making TPI and multiple product integrals even more difficult. For this reason, we compare our method against SG, but mainly focus on the wavelets.

Similar to the wavelet case, the number of SGs is not the actual storage. Each SG at least requires 4 floating numbers to specify its 2D center, 1D bandwidth and 1D amplitude, assuming it is isotropic.

2 NEURAL NETWORK DESIGNS

2.1 Representation Network

Our representation network is analogous to the autoencoder used in image compression [Agustsson et al. 2019]. We choose the octahedron parameterization [Clarberg 2008] instead of the commonly used cube maps or sphere maps, since the octahedron parameterization needs only one image, and ensures equal solid angle of all pixels. Given a 2D spherical function as input, the encoder can compress it into a low-dimensional latent vector that can be converted back to the original function via the decoder. However, we do not need to decode the latent vector back to the original function. That said, we just use our representation network to encode 2D spherical functions into latent vectors as a preprocessing step. Only the encoded low-dimensional latent vectors, rather than the expensive decoder, are used during rendering. Since the autoencoder is mainly used in the offline preprocess, we can make it sophisticated enough for capturing high-frequency and local details in given functions.

We show the architecture of our representation network in Fig. 1. Both the encoder and the decoder are fully convolutional neural networks (CNN) without skip connections. The encoded latent vector has a dimension of $3 \times 8 \times 8$, which has a compression rate of 0.39% compared to the original function f. Note that to guarantee the latent/coefficients contain all information to recover the input function, skip connections between the encoder and decoder, usually seen in U-Nets [Ronneberger et al. 2015], are strictly prohibited.

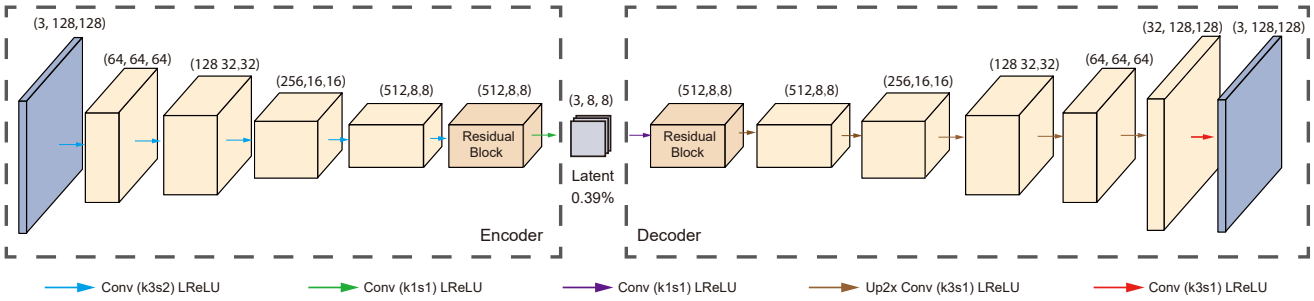

Figure 1: The structure of our representation ($\mathcal{P}$) network. The encoder transform the original 2D function into latent coefficients, which only take 0.39% of the original storage. Given the specific latent coefficients, the decoder would be able to approximately reconstruct the original 2D function.

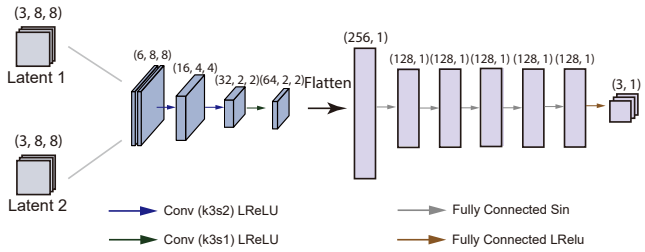

Figure 2: The DPI network has three convolutional layers and 6 fully connected layers with sin activation [Sitzmann et al. 2020] which are helpful to maintain highlights and high-frequency shadows. It take two latent vectors as input and perform double project integral on them.

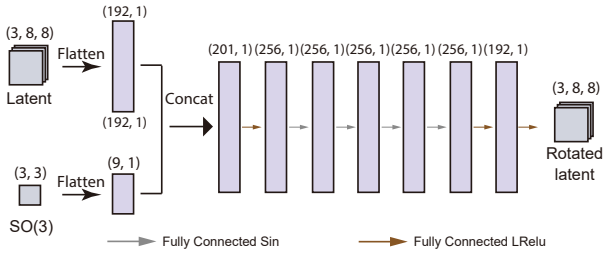

Figure 4: The structure of our $\mathcal{R}$ network. It consists of 7 fully connected layers and perform rotation on a latent vector with a specific $SO(3)$ rotation matrix.

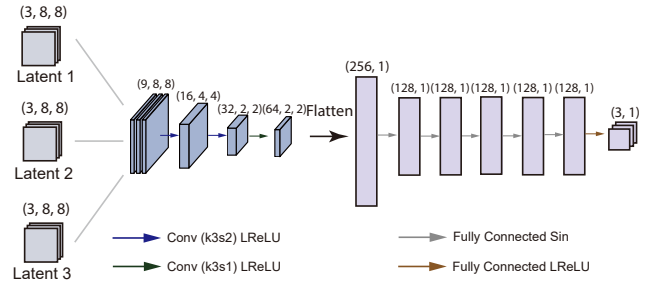

Figure 3: The TPI network's structure is very similar to DPI network except for the input are three latent vectors. It will perform triple product integral on these three latent vectors.

2.2 Computation Networks

We design three individual computation networks to support the operations including double product integral (DPI), triple product integral (TPI), and rotation ($\mathcal{R}$). In order to achieve fast computation during rendering, these networks should be as simple as possible to guarantee real-time performance. Additionally, we enforce the networks to directly operate on the latent space, i.e., taking the latent vectors compressed by the encoder as input.

The network structure of DPI and TPI are shown in Fig. 2 and Fig. 3. They share the same network design, except for the input layer, which is 2 latent vectors for DPI network and 3 latent vectors for TPI network. Note that we choose to use convolutional layers and sine activation functions [Sitzmann et al. 2020] in both DPI network and TPI network since they provide a better image quality. Replacing CNNs with fully connected layers will lead to blurry results.

Our $\mathcal{R}$ network is essentially rotating 2D function in the latent space, and the structure is shown in Fig. 4. It consists of 7 fully connected layers. Instead of taking in three Euler angles or quaternions, we use continuous rotation representation $SO(3)$ as input [Zhou et al. 2019], because Euler angles and quaternions are not continuous representations for spherical rotations and thus are not suitable for neural networks to learn.

3 IMPLEMENTATION

3.1 Training details

We implemented our networks in PyTorch [Paszke et al. 2017] for training. We use a desktop with one NVIDIA RTX 3090 GPU. We use Adam optimizer [Kingma and Ba 2014] with moment parameters of $\beta_1 = 0.5$, $\beta_2 = 0.999$. The initial learning rate is set as

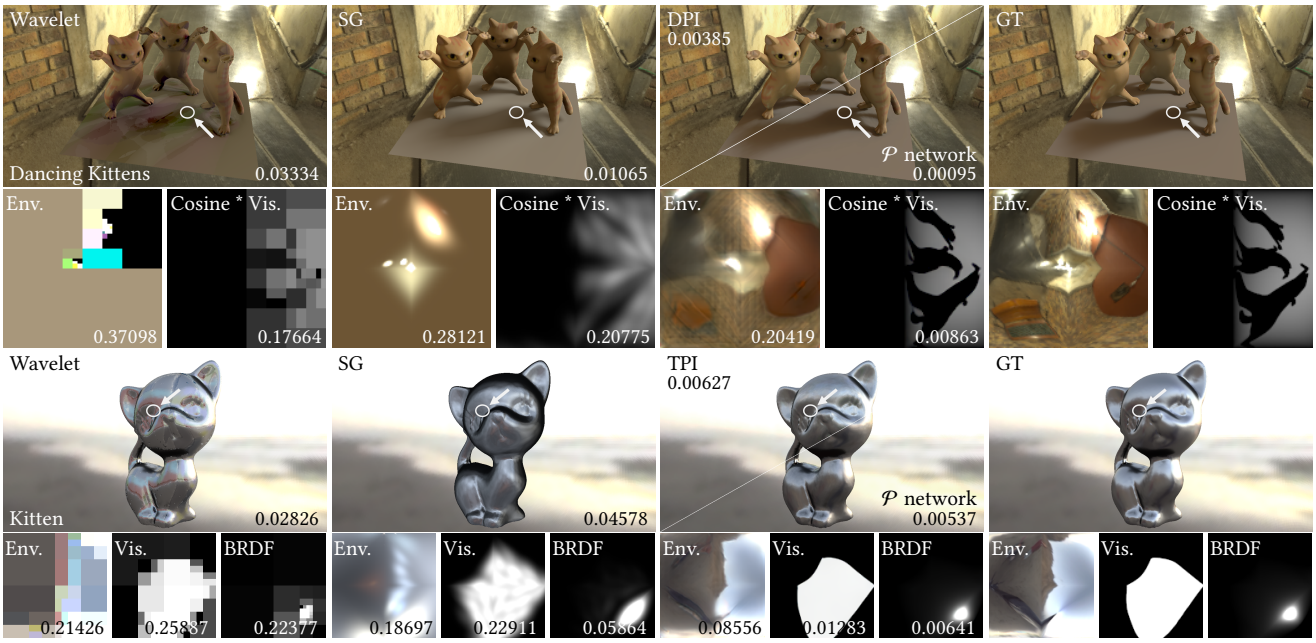

Figure 5: Comparison of rendered images using spherical functions compressed and reconstructed with our representation ($\mathcal{P}$) network, wavelet, and SG at an equal compression rate. Our $\mathcal{P}$ network can capture high-frequency details such as shadows and glossy highlights and produce rendering results close to the ground truth, while wavelet produces flickering colors and blocky shadows and SG fails to capture the high-frequency visibility and BRDF. Below the rendering results, We also visualize reconstructed and reference spherical functions (environment lighting, visibility, and BRDF). We provide the DSSIM error of each result which also indicates that our results achieve the highest quality.

Figure 6: This figure shows the our TPI network's results of using latent vectors directly from encoder (left, TPI w/o $\mathcal{R}$) and using latent vectors from our rotation network (middle, TPI w/ $\mathcal{R}$) with their DSSIM error (right bottom). The error map on the right shows their differences against the reference. Rendering with rotation in the latent space using our rotation network is accurate and matches the reference well.

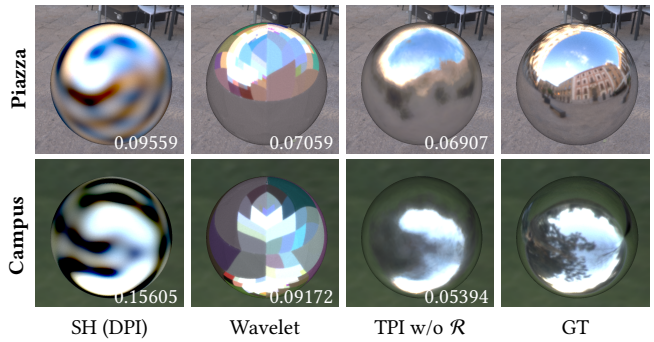

Figure 7: This scene is a sphere with very glossy BRDF. Each point on the sphere will reflect a tiny area of environment light. Our TPI fails to reproduce the desired mirror reflection effect due to information loss from heavy compression. In this case, SH produces ringing artifacts. While wavelets successfully handle the glossy BRDFs, the highly compressed environment lighting are inevitably blocky.

10^{-4}, and decay power of 0.95 for every 20 epochs. We train representation network jointly with DPI network or TPI network. In theory, it is feasible to train both DPI network and TPI network at the same time, but we choose to train either of them. Because they are designed for different applications. Note that when training the whole representation network (encoder and decoder), evaluating $\mathcal{L}_{\text{VGG}}$ in the decoder branch is very time-consuming and we found that the representation network will already have a strong representation ability after just first several epochs. To accelerate training, after jointly training the representation network and DPI/TPI network for 20 epochs, we fix the decoder and only train the encoder and DPI/TPI network. But we resume to train the decoder separately (and fix other components) every 20 epoch to maintain the decoder's representation ability. After finishing the training of DPI/TPI network, we fix everything and start to train the $\mathcal{R}$ network individually with environment maps only.

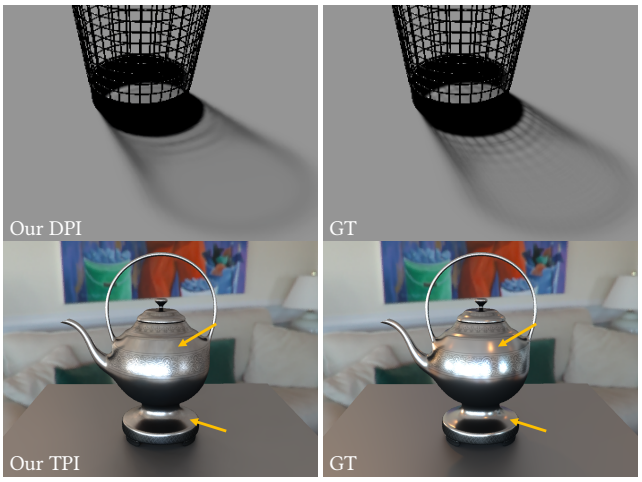

Figure 8: Although our method efficiently supports dynamic all-frequency shading, our lightweight computation networks may still blur/miss some extremely high-frequency shadows and lights due to the information loss of highly compressed latent vectors.

In theory, training our networks on large datasets can provide neural basis functions with good generalizability. However, because of the high image quality requirement of the rendering task, it is extremely challenging to generalize a pre-trained set of neural basis functions to arbitrary scene geometries and environmental lighting conditions. Therefore, we choose to further improve the image quality by fine-tuning our pre-trained model on the specific geometric data. The training time can be considered as part of the precomputation time. It gives us scene-specific neural basis functions with better quality. The pre-train step takes 3 days, and the fine-tuning generally takes about 1 days (e.g., Kitten in Fig. 5), possibly longer for more complex scene, depending on the specific geometric data size. Since we did our training with one single RTX 3090 GPU, it is possible to accelerate training multiple times and reduce the cost of fine-tuning to a few hours by using more GPUs.

3.2 Rendering preparation

For each scene, we run all the 2D spherical functions (environment lighting, visibilities, BRDFs and transport functions) per-pixel through the encoder, then the resulting latent vectors are stored as a multi-channel texture in GPU's global memory. For those pixels not covered by any objects, we use another binary texture to mark them invalid.

4 VALIDATION

4.1 All-frequency capacity

To further study the capacity of our neural basis functions, we show in Fig. 5 a comparison of rendered images using spherical functions (including environment lighting, transport function, visibility, and cosine-weighted BRDF) compressed and reconstructed with our representation network, wavelet, and SG at an equal compression rate. The rendering results are generated by pixel-wised multiplication and integration of the reconstructed spherical functions. And the scene statistics are summarized in Table. 1.

Scene	Figure	#Valid pixels	#Trained lighting	DPI (ms)	TPI (ms)	FPS
Dancing Kittens	Fig. 5	316539	10	7.1	N/A	140.8
Kitten	Fig. 5	174441	12	N/A	4.5	217.4

Table 1: The scene configurations and performance statistics. Our runtime performance of each scene meets the real-time requirements. The running time of the rotation network is negligible since the rotation is only operated on lighting.

Our representation network can capture high-frequency details such as shadows and glossy highlights and produce rendering results close to the ray-traced ground truth, while wavelet produces flickering colors and blocky shadows and SG fails to capture the high-frequency visibility and BRDF.

4.2 Joint Computation

Despite we design different computation neural networks as substitution for some mathematical operations, these neural networks are suppose to act as the general operations of our implicit neural basis functions. It means, in our case, the latent vectors output from rotation network are supposed to be usable for other computation networks. Fig. 6 verifies that our rotation network works well with computation networks.

5 FAILURE CASE

Because we use short latent vectors to compress 2D spherical functions and render images with lightweight computation neural networks, there is inevitable information loss in theory. While these compressed latent vectors may have little effect on the final rendering results, since they can properly represent the geometric functions (see the results of $\mathcal{P}$ network in Fig. 5). The computation networks (DPI, TPI, and $\mathcal{R}$ networks) produce more noticeable artifacts such as the blurred high-frequency shadows and missing specular reflection, as can be seen in Fig. 8 and the supplemental video. When extremely high-frequency signals are present, overblur artifacts can also arise, as shown in Fig. 7. This is because we use lightweight neural networks to perform complex integral computations. We believe that these artifacts can be better suppressed using smarter loss functions, advanced architectures, and more data.

REFERENCES

Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. 2019. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 221–231.

Petrik Clarberg. 2008. Fast equal-area mapping of the (hemi) sphere using simd. *Journal of Graphics Tools* 13, 3 (2008), 53–68.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. 2004. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 23, 3 (2004), 475–485.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661* (2020).

Weifeng Sun and Amar Mukherjee. 2006. Generalized wavelet product integral for rendering dynamic glossy objects. *Acm Transactions on Graphics* 25, 3 (2006), 955–966.

Rui Wang, Ren Ng, David P Luebke, and Greg Humphreys. 2006. Efficient Wavelet Rotation for Environment Map Rendering.. In *Rendering Techniques*. 173–182.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. 2019. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5745–5753.