NeX: Real-time View Synthesis with Neural Basis Expansion

Suttisak Wizadwongsa Pakkapon Phongthawee Jiraphon Yenphraphai
Supasorn Suwajanakorn
Vidyasirimedhi Institute of Science and Technology
Rayong, Thailand
{suttisak.w_s19, jiraphony_pro, pakkapon.p_s19, supasorn.s}@vistec.ac.th

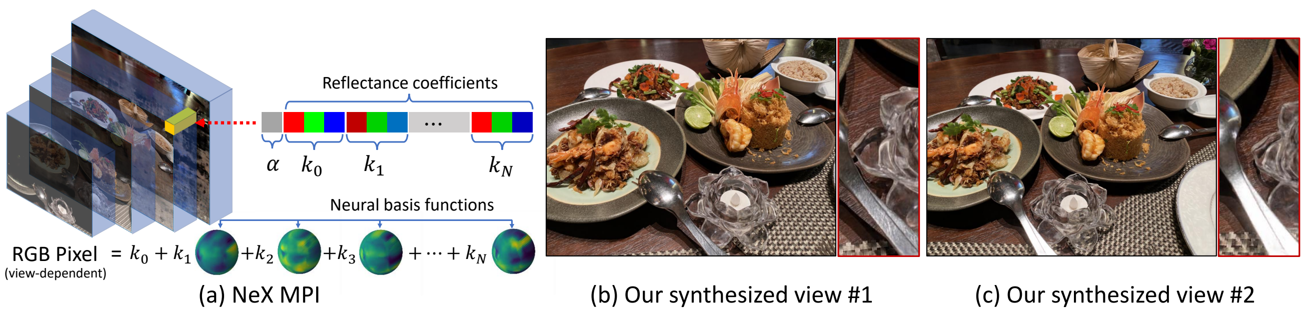


Figure 1: **(a)** Each pixel in NeX multiplane image consists of an alpha transparency value, base color k_0, and view-dependent reflectance coefficients $k_1 ... k_n$. A linear combination of these coefficients and basis functions learned from a neural network produces the final color value. **(b, c)** show our synthesized images that can be rendered in real-time with view-dependent effects such as the reflection on the silver spoon.

Abstract

We present NeX, a new approach to novel view synthesis based on enhancements of multiplane image (MPI) that can reproduce next-level view-dependent effects—in real-time. Unlike traditional MPI that uses a set of simple RGBα planes, our technique models view-dependent effects by instead parameterizing each pixel as a linear combination of basis functions learned from a neural network. Moreover, we propose a hybrid implicit-explicit modeling strategy that improves upon fine detail and produces state-of-the-art results. Our method is evaluated on benchmark forward-facing datasets as well as our newly-introduced dataset designed to test the limit of view-dependent modeling with significantly more challenging effects such as rainbow reflections on a CD. Our method achieves the best overall scores across all major metrics on these datasets with more than 1000× faster rendering time than the state of the art.

1. Introduction

Novel view synthesis is an exciting problem that draws much attention from both the computer graphics and vision communities. The challenge lies in the two intriguing questions of how to construct a visual scene representation from only a sparse set of images and how to render such a representation from unseen perspectives. A wide range of applications are possible from this area of research ranging from virtually visiting tourist attractions to viewing any online product all around in 3D; however, such experiences would only become most compelling and practical when the representation allows photo-realistic and real-time synthesis.

One candidate that can serve this purpose is multiplane image (MPI) [43] which approximates the scene's light field with a set of parallel semi-transparent planes placed along a reference viewing frustum. This representation is shown to be more effective than traditional 3D mesh reconstruction in reproducing complex scenes with challenging occlusions, thin structures, or planar reflections. However, the standard RGBα representation of MPI is limited to diffuse surfaces whose appearance stays constant regardless of the viewing angle. This greatly limits the types of objects and scenes that MPI can capture. Recent research on implicit scene representation has made significant progress in the past months [18, 28, 15, 40, 35] and can be applied to view synthesis problem. Unfortunately, its expensive network inference still prohibits real-time rendering, and reproducing complex surface reflectance with high fidelity still remains a challenge. Our method breaks these limits on both fronts.

We introduce NeX, a new scene representation based on MPI that models view-dependent effects by performing basis expansion on the pixel representation in our MPI. In particular, rather than storing static color values as in traditional MPI, we represent each color as a function of the viewing angle and approximate this function using a linear combination of spherical basis functions learned from a neural network. Furthermore, we propose a hybrid parameter modeling strategy that models high-frequency detail in an explicit structure within an implicit MPI modeling framework. This strategy helps improve fine detail that is difficult to model by a neural network and produces sharper results in fewer training iterations.

We evaluate our algorithm on benchmark forward-facing datasets and compare against state-of-the-art approaches including NeRF [19] and DeepView [5]. These datasets, however, contain mostly diffuse scenes and fairly simple view-dependent effects and cannot be used to judge the new limit of our algorithm. Thus, we collect a new dataset, *Shiny*, with significantly more challenging view-dependent effects such as rainbow reflections on an optical disk, refraction through non-planar glassware and a magnifying glass. Our method achieves the best overall scores across all major metrics on these datasets. We provide quantitative and qualitative results and ablation studies to justify our main technical contributions. Compared to the recent state of the art, NeRF [19], our method captures more accurate view-dependent effects and produces sharper results—all in real-time.

2. Related Work

Learning MPIs. Multiplane image is first introduced by Zhou et al. [43] to solve a stereo problem from a small-baseline stereo pair. This representation consists of parallel semi-transparent planes placed along a reference viewing frustum. In [43], they introduce a learning-based pipeline using a convolutional neural network (CNN) to predict an MPI from a pair of reference images. Their network is specifically designed for two input images and does not generalize to multiple input photos. Subsequent work explores this extension to support multiple input photos [5, 18, 14] or even infers an MPI from a single image [36]. In [18], a CNN is used to predict multiple nearby MPIs which are then blended together to produce the final output. Srinivasan et al. [31] predicts the final MPI using a two-step process that combines 3D CNNs for MPI prediction and a 2D flow field for warping RGB values from an intermediate rendering. In contrast, DeepView by Flynn et al. [5] uses CNN to learn gradient updates to the MPI instead of predicting MPI directly. This learned gradient descent helps avoid over-fitting and requires only a few iterations to generate an MPI. Recently, DeepMPI by Li et al. [14] has been introduced to model time-varying scene appearance and can manipulate colors on their MPI using a CNN. However, these approaches do not model view-dependent effects or only handle them indirectly by blending multiple view-*independent* MPIs. This greatly limits the types of objects and scenes that they can capture.

View synthesis and interpolation. One way to categorize view synthesis algorithms is by how dense the input scene is sampled. When the capture is dense as in lumigraph [8, 2] and light field rendering [13, 39, 12, 27], the challenge becomes how to store and interpolate the light field samples. When there are only 1-2 input images, the challenge becomes how to infer the ill-constrained 3D geometry and disoccluded regions [32, 20, 36, 38, 3]. Our work focuses on the case with a moderate number of captures facing forward. Besides MPI-based approaches, other work that solves this problem includes Soft3D by Penner et al. [22] which combines depth estimation with soft blending of an estimated geometry, and other 3D reconstruction based techniques [44, 9].

Neural approaches to view synthesis and view-dependent modeling include DeepStereo [6] which uses a CNN to predict pixels directly for individual viewing angles and Neural Textures [34] which combines a reconstructed 3D mesh with neural textures that can be rendered with a neural network. Similar ideas of using neural latent code stored in some geometric structure such as a voxel grid or volumes have been proposed [29, 4, 16]. Neural BTF [24] represents the bidirectional texture function with an encoder-decoder network that takes in light and viewing angles and outputs each color pixel. [11] uses a generative adversarial network to model spatially varying BRDFs of specular microstructures.

One recent notable work is Neural Radiance Fields (NeRF) by Mildenhall et al. [19] which represents a 5D radiance field with a multilayer perceptron (MLP) that directly regresses the volume density and RGB colors. This method can handle view-dependent effects as the viewing angle is part of the 5D radiance function. Subsequent work improves upon NeRF by using explicit sparse voxel representation to improve fine detail (NSVF) [15], parameterizing the space to better support unbounded scenes (NeRF++) [40], incorporating learned 2D features that help enforce multiview consistency (GRF) [35], or extending NeRF to handle photometric variations and transient objects in internet photo collections (NeRF-W) [17]. Another related line of work involves implicitly modeling surface reflectance properties in addition to the scene geometry [1] or the light transport function [42].

Our work is inspired by these implicit neural representations NeRF [19], DeepSDF [21], and others as well as deep image prior [37]. However, our goal and design prin-

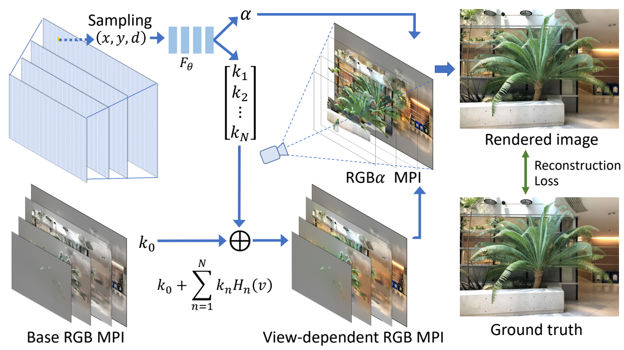


Figure 2: NeX overview: we construct each pixel in our MPI by sampling a pixel coordinate (x, y) at plane depth d and feed it to a multilayer perceptron (MLP) to output alpha transparency and view-dependent basis coefficients $(k_1, k_2, ..., k_n)$. These coefficients, together with explicit k_0, are multiplied with basis functions predicted from another MLP, to produce the RGB value. The output image is the product of the composite operation over all planes (Eq. 1). We train the two MLPs and optimize for the explicit k_0 by comparing the rendered image to the ground truth.

ciple are directed toward a representation amendable to discretization and real-time rendering, and we have achieved this with superior quality compared to the state of the art.

3. Approach

Given a set of multiview images of a scene, our goal is to construct a 3D representation that can render novel views with view-dependent effects in real-time. To solve this, we propose a novel representation based on multiplane image [43] but with significant improvements which include a novel view-dependent pixel representation that can handle non-Lambertian surfaces and a hybrid implicit-explicit parameter modeling to improve fine detail. Our approach focuses on forward-facing captures with around 12 images or more, such as those taken casually with a smartphone in front of a scene. In the following sections, we first briefly review the original MPI representation, then explain our novel representation and a learning method for inferring it.

3.1. Original MPI Representation

Multiplane image [43] is a 3D scene representation that consists of a collection of D planar images, each with dimension $H \times W \times 4$ where the last dimension contains RGB values and alpha transparency values. These planes are scaled and placed equidistantly either in the depth space (for bounded close-up objects) or inverse depth space (for scenes that extend out to infinity) along a reference viewing frustum (see Figure 2).

Rendering an RGBα MPI in any target view can be done by first warping all its planes to the target view via a homography that relates the reference and target view and apply the composite operator [23]. In particular, let $c_i \in \mathbb{R}^{H\times W\times 3}$ and $\alpha_i \in \mathbb{R}^{H\times W\times 1}$ be the RGB and alpha "images" of the i^{th} plane respectively, ordered from back to front. And denote $A = \{\alpha_1, \alpha_2, ..., \alpha_D\}$, $C = \{c_1, c_2, ..., c_D\}$ as the sets of these images. This MPI can then be rendered in a new view, $\hat{I}$, using the composite operator O:

$$\hat{I} = O(W(A), W(C)) \tag{1}$$

where W is a homography warping function that warps each image to the target view, and O has the form:

$$O(A, C) = \sum_{d=1}^{D} c_d T_d(A),\ T_d(A) = \alpha_d \prod_{i=d+1}^{D} (1 - \alpha_i) \tag{2}$$

This rendering equation is completely differentiable, thus allowing MPI to be inferred through image reconstruction loss [43, 5].

3.2. View-Dependent Pixel Representation

One main limitation of MPI is that it can only model diffuse or Lambertian surfaces, whose colors appear constant regardless of the viewing angle.[1] In real-world scenes, many objects are non-Lambertian such as a ceramic plate, a glass table, or a metal wrench. These objects exhibit view-dependent effects such as reflection and refraction. Reconstructing these objects with an MPI can make the objects appear unrealistically dull without reflections or even break down completely (Figure 6) due to the violation of the brightness constancy assumption used for matching invariant and 3D reconstruction. [18] attempts to solve this by combining multiple view-*independent* MPIs, but their results contain warping artifacts when blending between MPIs.

To allow for view-dependent modeling in our MPI, we modify the pixel color representation, originally stored as RGB values, by parameterizing each color value as a function of the viewing direction $\mathbf{v} = (v_x, v_y, v_z)$. This results in a 3-dimensional mapping function $\mathcal{C}(\mathbf{v}) : \mathbb{R}^3 \to \mathbb{R}^3$ for every pixel. However, storing this mapping explicitly is prohibitive and not generalizable to unobserved angles. Regressing the color directly from $\mathbf{v}$ (and the pixel location) with a neural network, as is done in e.g. [19], is possible though inefficient for real-time rendering. Our key idea is to approximate this function with a linear combination of *learnable* basis functions $\{H_n(\mathbf{v}) : \mathbb{R}^3 \to \mathbb{R}\}$ over the spherical domain described by vector $\mathbf{v}$:

$$\mathcal{C}^{\mathbf{P}}(\mathbf{v}) = k_0^{\mathbf{P}} + \sum_{n=1}^{N} k_n^{\mathbf{P}} H_n(\mathbf{v}) \tag{3}$$

[1] A single MPI can simulate planar reflections to some extent by placing the reflected content on one of its planes [5].

where $k_n^{\mathbf{p}} \in \mathbb{R}^3$ for pixel $\mathbf{p}$ are RGB coefficients, or reflectance parameters, of N global basis functions. In general, there are several ways to define a suitable set of basis functions. Spherical harmonics basis is one common choice used heavily in computer graphics to model complex reflectance properties. Fourier's basis or Taylor's basis can also be used. However, one shortcoming of these "fixed" basis functions is that in order to capture high-frequency changes within a narrow viewing angle, such as sharp specular highlights, the number of required basis can be very high. This in turns requires more reflectance parameters which make both learning these parameters and rendering more difficult. With learnable basis functions, our modified MPI outperforms other versions with alternative basis functions that use the same number of coefficients shown in our experiment in Section 4.3.2.

To summarize, our modified MPI contains the following parameters per pixel: $\alpha, k_0, k_1, \ldots, k_N$; and global basis functions $H_1, H_2, \ldots, H_N$ shared across all pixels.

3.3. Modeling MPI with Neural Networks

Given our modified MPI and the differentiable rendering equation 1, one can directly optimize for its parameters that best reproduce the training views. However, as demonstrated in earlier work [5], doing so would lead to a noisy MPI that overfits the training views and fails to generalize. Inspired by Deep Image Prior [37] and NeRF [37], we solve this problem by regressing these parameters with multilayer perceptrons (MLPs) from spatial conditioning, i.e., pixel coordinates. In particular, we use two separate MLPs; one for predicting per-pixel parameters given the pixel location, and the other for predicting all global basis functions given the viewing angle. The motivation for using the second network is to ensure that the prediction of the basis functions, which are global, is not a function of the pixel location.

Our first MLP is modeled as F_θ with parameter θ:

$$F_\theta : (\mathbf{x}) \rightarrow (\alpha, k_1, k_2, ..., k_N) \tag{4}$$

where $\mathbf{x} = (x, y, d)$ contains the location information of pixel (x, y) at plane d. Note that k_0 is not predicted by F_θ but will be stored explicitly in our implicit-explicit modeling strategy, explained in the upcoming section. The second network is modeled as G_ϕ with parameter ϕ:

$$G_\phi : (\mathbf{v}) \rightarrow (H_1, H_2, ..., H_N) \tag{5}$$

where $\mathbf{v}$ is the normalized viewing direction.

It is interesting to note that unlike other MPI-learning networks, our networks do not take as input any information from the training images but will receive supervision from those images through an image reconstruction loss. This kind of parameter estimation and other coordinate regressions (e.g., [21, 19]) can be thought of as applying a deep image prior [37] on the estimated parameters. In other words, instead of allowing the estimated parameters to take arbitrary values which are prone to overfitting, we regularize these parameters by only allowing them to take on certain values that are in the span of a deep neural network's output. When the network is convolutional, the span of the network can capture the natural image manifold surprisingly well [37]. In our case, we found that deep priors based on multilayer perceptrons can regularize our MPI and produce superior results compared to a direct optimization without deep priors or with a standard regularizer such as total variation.

3.4. Implicit-Explicit Modeling Strategy

One observation when using an MLP to model k_n, or "coefficient images" when $k_n^{\mathbf{p}}$ is evaluated on all pixels $\mathbf{p}$, is the absence of fine detail (similar reports in [28, 19, 33]). In our problem, fine detail or high-frequency content tends to come from the surface texture itself and not necessarily from a complex scene geometry. Thus, we use positional encoding proposed in [19] to regress these images, which helps to an extent but still produces blurry results. Interestingly, we found that simply storing the first coefficient k_0, or "base color," explicitly helps ease the network's burden of compressing and reproducing detail and leads to sharper results, also in fewer iterations. With this implicit-explicit modeling strategy, we predict every parameter with MLPs except k_0 which will be optimized explicitly as a learnable parameter with a total variation regularizer.

Coefficient Sharing: In practice, computing and storing all $N + 1$ coefficients for all pixels for all D planes can be expensive for both training and rendering. In our experiment, we use a coefficient sharing scheme where every M planes will share the same coefficients, but not the alphas. That is, there is a single set of $\{K_0, ..., K_N\}$ for planes 1 to M, and another set for planes $M + 1$ to $2M$, and so forth. With proper N and $M > 1$, we do not observe any significant degradation in the visual quality, but a significant gain in speed and model compactness.

Finally, to optimize our model, we evaluate the two MLPs to obtain the implicit parameters, render an output image $\hat{I}_i$, and compare it to the ground-truth image I_i from the same view. We use the following reconstruction loss:

$$L_{\text{rec}}(\hat{I}_i, I_i) = \|\hat{I}_i - I_i\|^2 + \omega \|\nabla \hat{I}_i - \nabla I_i\|_1, \tag{6}$$

where ∇ denotes the gradient operator and ω is a balancing weight [26]. Our approach is summarized in Algorithm 1.

3.5. Real-time Rendering

Every model parameter in our MPI can be converted to an image. This is done by evaluating F_θ on all pixel coordinates and G_ϕ on some pre-defined viewing span. Given these pre-computed images, we can implement Equation 1

Algorithm 1: MPI training with NeX

initialize: θ, ϕ, K_0;
pre-compute $\mathcal{X}$ pixel coordinate for each pixel;
for *Iteration=0 to maxIter* **do**
  sampling image I_i;
  compute $(A, \vec{K}) = F_\theta(\mathcal{X})$ where $\vec{K} = [K_1, K_2, ..., K_N]$;
  compute viewing direction $\mathcal{V}_i$ by $\mathcal{V}_i = \mathcal{X} -$ center of projection of I_i; $\mathcal{V}_i = \mathcal{V}_i / \|\mathcal{V}_i\|$;
  compute view-dependent color $C = K_0 + \vec{K} \cdot \vec{H}_\phi(\mathcal{V}_i)$;
  compute rendered image $\hat{I}_i = O(W_i(A), W_i(C))$;
  compute loss function by $L = L_{\text{rec}}(\hat{I}_i, I_i) + \gamma \text{TV}(K_0)$;
  update θ, ϕ, K_0 with ADAM$(\nabla_{\theta,\phi,K_0} L)$;
end
Result: $A, K_0, K_1, ..., K_N$

in a fragment shader in OpenGL/WebGL and achieve real-time view-dependent rendering of our captured scenes.

4. Experiments

We perform quantitative and qualitative evaluations against state-of-the-art methods for novel view synthesis which include MPI-based methods and others. We also provide an extensive study on the choice of the basis functions and evaluate different variations of implicit-explicit modeling of the MPI parameters, ranging from fully implicit to fully explicit.

4.1. Implementation Details

Our model is optimized independently for each scene. The input photos are first calibrated and undistorted with a structure-from-motion algorithm from COLMAP [25]. In most of our experiments unless stated otherwise, we use an MPI with 192 layers with $M = 12$ consecutive planes sharing one set of texture coefficients.

MLP architectures: For F_θ that predicts per-pixel parameters given the pixel location (x, y, d), we follow NeRF's [19] positional encoding and project input x, y to 20 dimensions each and plane depth d to 16 dimensions with the following projection $p(u) = \left[\sin(2^0 \frac{\pi}{2} u), \cos(2^0 \frac{\pi}{2} u), ..., \sin(2^k \frac{\pi}{2} u), \cos(2^k \frac{\pi}{2} u)\right]$ where input u is first normalized to $[-1, 1]$. The total input dimension is 56. This network uses 6 fully-connected LeakyReLU layers, each with 384 hidden nodes. The output α uses a sigmoid activation, and the others use tanh activations. For G_ϕ that predicts the basis functions, we use positional encoding of the input viewing direction with 12 dimensions including 6 dimensions for v_x and v_y. This network uses 3 fully-connected LeakyReLU layer with 64 hidden nodes to output 8 dimensions of $\vec{H}_\phi(\mathbf{v})$.

Training details: To compute the loss, we randomly sample and render 8000 pixels in the training view and compare them to the corresponding pixels in the ground-truth image. We set $\omega = 0.05$, $\gamma = 0.03$ and train our networks for 4,000 epochs using Adam optimizer [10] with a learning rate of 0.01 for base color and 0.001 for both networks and a decay factor of 0.1 every 1,333 epochs.

Runtime: For a scene with 17 input photos of resolution 1008×756, the training takes around 18 hours using a single NVIDIA V100 with a batch size of 1. Our WebGL viewer can render this scene at 60 frames per second using an NVIDIA RTX 2080Ti. For comparison, NeRF takes about 55 seconds to generate one frame on the same machine.

4.2. Comparison to the State of the Art

We compare our algorithm to state-of-the-art MPI-based methods, DeepView [5] and LLFF [18], as well as non-MPI-based NeRF[19] and neural scene representations (SRN) [30]. We also compare qualitatively to recent work, NSVF [15] in our supplementary; however, their method focuses on object captures and is not designed to handle scenes with background due to the use of a bounded voxel grid. For evaluations, we use Spaces dataset from DeepView and Real Forward-Facing dataset from NeRF. Moreover, we introduce a significantly more challenging dataset, *Shiny*, to test the limit of view-dependent modeling.

4.2.1 Results on Real Forward-Facing Dataset

This dataset contains 8 scenes captured in real-world environments using a smartphone. The number of input images for each scene ranges from 20 to 62 images, each with a resolution of 1008×756. We use the same train/test split as NeRF and evaluate our test results using 3 metrics: PSNR (Peak Signal-to-Noise Ratio, higher is better), SSIM (Structural Similarity Index Measure, higher is better) and LPIPS[41] (Learned Perceptual Image Patch Similarity, lower is better).

As shown in Table 1, our method produces the highest average scores across all 3 metrics. We show scores for individual scenes in our supplementary. Note that we need to undistort the results from NeRF in order to match our calibrated testing views. By doing so, their average scores increase, and we provide both the new scores and their original scores for reference in the supplementary. NeRF has a higher PSNR than ours on one scene, "Orchids," and upon inspection we found that our result looks distorted near the image boundary. Compared to NeRF, our results have much sharper detail and less noise in regions with uniform colors as seen in Figure 3. The detail from LLFF is on a par with ours; however, LLFF produces jumping and warping arti-

Table 1: Average scores across 8 scenes in Real Forward-Facing dataset.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
SRN [30]	21.82	0.744	0.464
LLFF [18]	24.41	0.863	0.211
NeRF [19]	26.76	0.883	0.246
NeX (Ours)	**27.26**	**0.904**	**0.178**

Table 2: Average scores across 8 scenes in Shiny dataset.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [19]	25.60	0.851	0.259
NeX (Ours)	**26.45**	**0.890**	**0.165**

facts when results are rendered as a video. SRN produces blurry results that do not look realistic for this dataset. Note that our algorithm renders state-of-the-art results more than $1000\times$ faster than NeRF, and is the first to achieve real-time 60FPS rendering at this quality.

4.2.2 Results on Shiny Dataset

Our Shiny dataset also contains 8 scenes captured with a smartphone in a similar manner as Real Forward-Facing dataset. However, the scenes contain much more challenging view-dependent effects such as rainbow reflections on an optical disk, refraction through a liquid bottle or a magnifying glass, metallic and ceramic reflections, and sharp specular highlights on silverware, as well as scenes with detailed thin structures.

Table 2 demonstrates that our approach also outperforms NeRF on all 3 metrics on this dataset. In scene CD, our method can reproduce the rainbow reflections and the reflected image of a plastic cup on the CD while NeRF fails to capture the reflected image, as seen in Figure B.8. In scene Tools, our method produces a sharper image of the solder coil stand through the magnifying glass. In scene Food, our method captures the specular microgeometry of the textured ceramic plate with high fidelity. Our failure cases include the lack of sharp sparkles in the crystal candle holder in scene Food and the reflection of tube rack in scene Lab which shown in Figure 7. Currently, no other methods are able to handle extremely sharp highlights that only appear in one distinct location in each input view. Handling these effects is an exciting subject of future work.

4.2.3 Results on DeepView's Spaces Dataset

Spaces dataset contains indoor and outdoor captures using 16 forward-facing cameras on a fixed rig. Each image has a resolution of 800×480. Similarly to DeepView, we evaluate on 10 scenes in Spaces dataset. We train our model on 12 input views, then evaluate on 4 held-out views. Table 3 shows a comparison between Soft3D [22], DeepView [5], and our work. Note that DeepView only estimates an MPI with 80 planes, and these scores are computed from the test images released by those papers. Our method produces higher average scores than DeepView on all metrics. Figure B.8 shows close-up results on one of the scenes from Spaces dataset. Note that DeepView focuses on setups with sparser input views than ours and can produce reasonable results with 4 input views by learning from a large dataset of scenes.

Table 3: Average scores on DeepView's Spaces dataset using 12 input views.

Method	PSNR↑	SSIM ↑	LPIPS ↓
Soft3D [22]	31.57	0.964	0.126
Deepview[5]	31.60	0.978	0.085
NeX (Ours)	**35.84**	**0.985**	**0.083**

4.3. Ablation Studies

We evaluate the effectiveness of our main contributions which are learned basis functions for view-dependent pixel representation and the implicit-explicit modeling strategy. For ablation studies, we train a 72-layer MPI with $M = 6$ sharing scheme and test on two scenes: "Tools," which contains multiple types of view-dependent effects, and "Crest," which contains high-detail patterns and thin structures from Shiny dataset. All images are in 1008×756 resolution.

4.3.1 Varying the Number of Basis Coefficients

We vary the number of basis coefficients from zero, which represents no view-dependent modeling, to 20 and show quantitative results in Figure 5. Adding view-dependent modeling to MPI helps increase PSNR in all test scenes and significantly improve the visual quality for scenes with challenging lighting effects shown in Figure 6.

4.3.2 Types of Basis Functions

We compare our learned basis functions to other types of basis for modeling view-dependent effects by only changing H_n in Equation 3. We test the following basis options: Taylor Series (TS), Spherical Harmonics (SH), Hemispherical Harmonics (HSH) [7], Jacobi Spherical Harmonics (JH), and Fourier Series (FS). Spherical harmonics are commonly used for representing complex illumination and are derived from Legendre polynomials. However, since our captures are mostly forward-facing, the viewing direc-

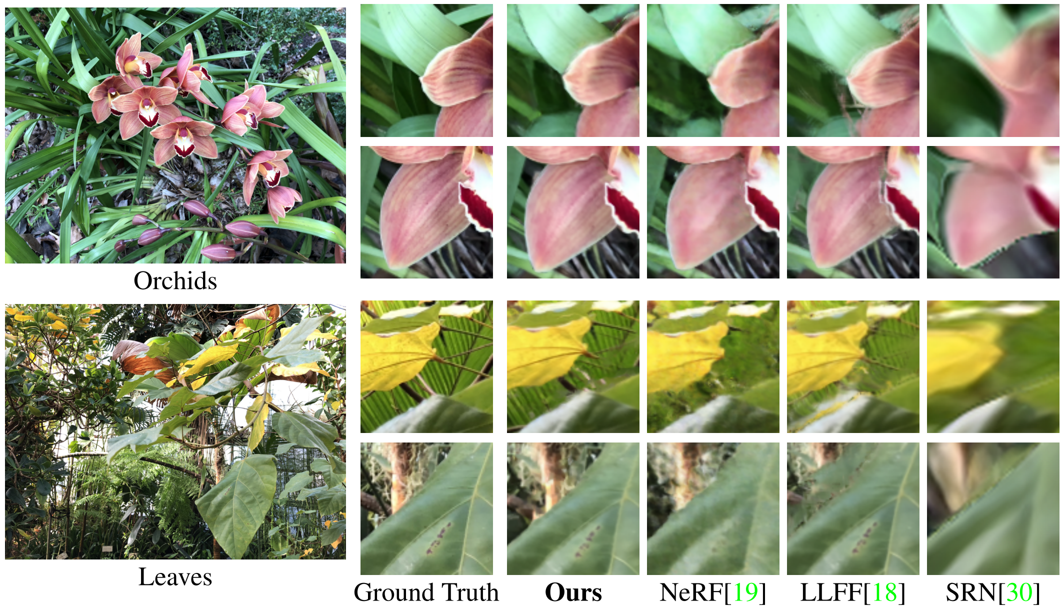
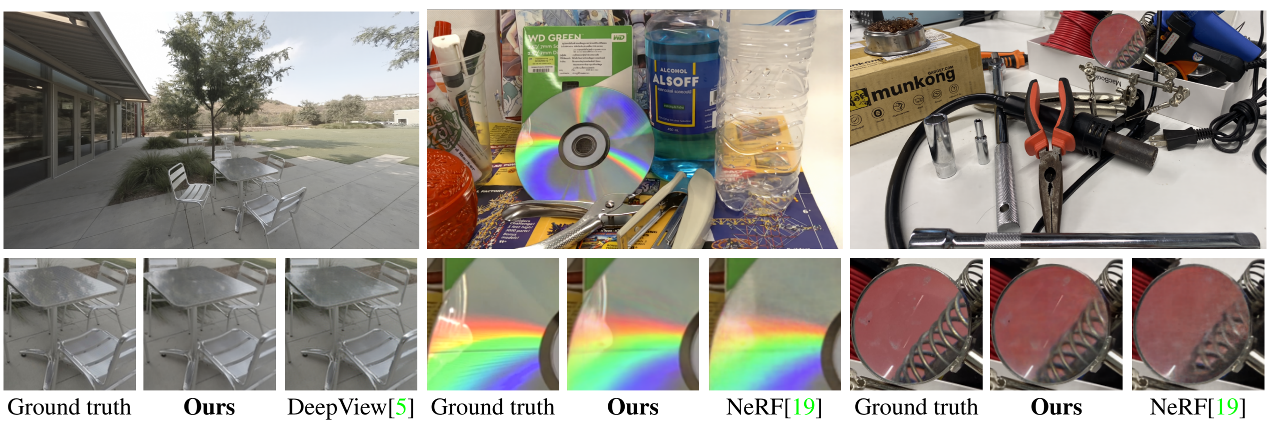


Figure 3: Qualitative results on test views from NeRF's real forward-facing dataset. Our method captures more complete geometry than LLFF and SRN in scene Orchids and recovers the most detail in scene Leaves.



(a) Spaces dataset: Scene 056 (b) Shiny dataset: CD (c) Shiny dataset: Tools

Figure 4: The top row shows our rendered results. (a) Results using 12 input views from Spaces dataset. Our method captures more accurate reflections on the table top. (b) Our method captures the reflected image of a plastic cup as well as the rainbow reflections while NeRF produces a blurry and noisy result. (c) Our method produces a sharper image of the coil through the magnifying glass.

tions from which a point on a surface can be observed will only span a hemisphere. Thus, we also evaluate an alternative basis functions that are more suitable for this viewing span, namely Hemi-spherical harmonics [7] which are derived from shifted Legendre polynomials. Generalizing this further, one can derive modified spherical harmonics that specifically target an even tighter viewing domain than a hemisphere through shifted Jacobi polynomials (JH). The exact forms can be found in our supplementary.

Figure 5 shows that our learned basis outperforms these fixed basis functions, even the ones whose viewing domains have been narrowed down, when the same number of basis coefficients are used. In principle, given a sufficiently expressive network, our learned basis can approximate other kinds of basis functions, if required, or reproduce higher frequencies using the same rank order, which is the number that dictates the highest possible frequency in those fixed basis functions.

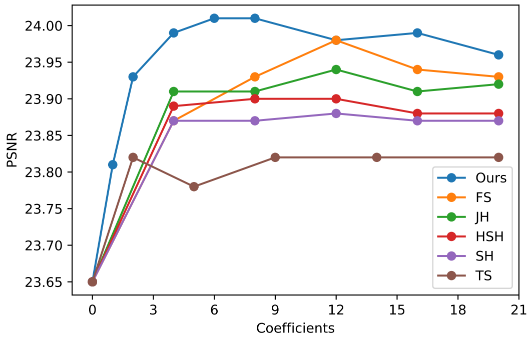
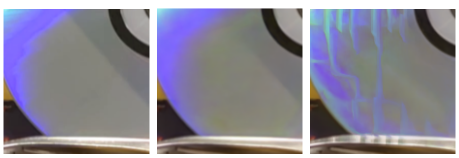


Figure 5: PSNR scores versus the number of basis coefficients for ours (neural basis functions), FS (Fourier's series), JH (Jacobi spherical harmonics), HSH (hemispherical harmonics), SH (spherical harmonics), and TS (Taylor's series).



Ground truth **Ours** Standard MPI

Figure 6: Results on scene CD in Shiny dataset between our model with view-dependent modeling and standard MPI (zero view-dependent coefficients). Our model can replicate the rainbow reflections while the standard MPI breaks down completely.

4.3.3 Implicit Function and Explicit Structure

In this experiment, we validate our design decision that stores base color K_0 explicitly while modeling other parameters with implicit functions. Additionally, in Table 4 we explore all 8 alternatives for representing alpha (A), base color (K_0), and view-dependent coefficients ($K_1, ..., K_N$), ranging from a fully implicit model (Im-Im-Im) to a fully explicit model (Ex-Ex-Ex). Note that the fully explicit model corresponds to optimizing the rendering equation directly without any deep priors on the parameters. The result shows that our Im-Ex-Im outperforms other alternatives and storing base color K_0 explicitly is beneficial also to other configurations regardless of the modeling choices for the alpha and coefficients. Qualitatively, our method produces significantly better detail compared to the fully implicit model and cleaner results that generalize better than the fully explicit model (please see our supplementary).

5. Limitations & Failure Cases

Our model is based on MPI and thus inherits similar limitations. When viewing our MPI from an angle too far away from the center, there will be "stack of cards" artifacts which can reveal individual MPI planes. Our model still cannot fully reproduce the hardest scenes in our Shiny dataset which include effects such as light sparkles, extremely sharp highlights, or refraction through test tubes (Figure 7). Exposure differences in the training images that are not properly compensated may lead to flickering in the rendered output. Training our MPI still takes a long time and may require a higher number of input views to replicate view-dependent effects. Learning how to do this with fewer input images using a dataset of scenes or with learned optimizers [5] could be an interesting direction.

Table 4: Quantitative evaluation on different modeling strategies for the alpha transparency (A), base color (K_0), and view-dependent coefficients ($K_1, ..., K_N$). Modeling with an explicit structure is denoted by (Ex) and with an implicit representation is denoted by (Im).

Method			Metric		
A	K_0	$K_1, ..., K_n$	PSNR ↑	SSIM ↑	LPIPS ↓
Ex	Ex	Ex	24.57	0.857	0.292
Ex	Ex	Im	24.47	0.854	0.300
Ex	Im	Ex	24.55	0.857	0.296
Ex	Im	Im	24.44	0.854	0.302
Im	Ex	Ex	26.30	0.901	0.204
Im	**Ex**	**Im**	**26.32**	**0.904**	**0.202**
Im	Im	Ex	25.82	0.883	0.279
Im	Im	Im	25.63	0.878	0.301

Ground truth **Ours** Ground truth **Ours**

Figure 7: Our failure cases on a crystal candle holder in scene Food (Left) and test tubes in scene Lab (Right).

6. Conclusion

We have investigated a new approach to novel view synthesis using multiplane image with neural basis expansion. Our representation is effective in capturing and reproducing complex view-dependent effects and efficient to compute on standard graphics hardware, thus allowing real-time rendering. Extensive studies and evaluations on public datasets as well as our more challenging dataset demonstrate state-of-the-art quality of our approach. We believe neural basis expansion can be applied to approximate other surface properties such as the bidirectional reflectance distribution function (BRDF) or enable efficient rendering for other scene representations including implicit neural representations.

References

[1] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. Neural reflectance fields for appearance acquisition. *arXiv preprint arXiv:2008.03824*, 2020. 2

[2] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001. 2

[3] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7781–7790, 2019. 2

[4] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 2

[5] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2367–2376, 2019. 2, 3, 4, 5, 6, 7, 8

[6] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. pages 5515–5524, 2016. 2

[7] Pascal Gautron, Jaroslav Krivanek, Sumanta Pattanaik, and Kadi Bouatouch. A Novel Hemispherical Basis for Accurate and Efficient Rendering. 2004. 6, 7, 11

[8] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, 1996. 2

[9] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3d photography. *ACM Transactions on Graphics (TOG)*, 36(6):1–15, 2017. 2

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 5

[11] Alexandr Kuznetsov, Milos Hasan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. Learning generative models for rendering specular microgeometry. *ACM Trans. Graph.*, 38(6):225–1, 2019. 2

[12] Anat Levin and Fredo Durand. Linear view synthesis using a dimensionality gap light field prior. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1831–1838. IEEE, 2010. 2

[13] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996. 2

[14] Zhengqi Li, Wenqi Xian, Abe Davis, and Noah Snavely. Crowdsampling the plenoptic function. *arXiv preprint arXiv:2007.15194*, 2020. 2

[15] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 1, 2, 5, 13

[16] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 38(4):65, 2019. 2

[17] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. *arXiv preprint arXiv:2008.02268*, 2020. 2

[18] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 1, 2, 3, 5, 6, 7

[19] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020. 2, 3, 4, 5, 6, 7, 10, 11, 13

[20] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Transactions on Graphics (TOG)*, 38(6):1–15, 2019. 2

[21] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 2, 4

[22] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017. 2, 6

[23] Thomas Porter and Tom Duff. Compositing digital images. *SIGGRAPH Comput. Graph.*, 18(3):253–259, Jan. 1984. 3

[24] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural btf compression and interpolation. *Computer Graphics Forum (Proceedings of Eurographics)*, 38(2), Mar. 2019. 2

[25] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5, 10

[26] Qi Shan, Jiaya Jia, and Aseem Agarwala. High-quality motion deblurring from a single image. *ACM Transactions on Graphics (SIGGRAPH)*, 2008. 4

[27] Lixin Shi, Haitham Hassanieh, Abe Davis, Dina Katabi, and Fredo Durand. Light field reconstruction using sparsity in the continuous fourier domain. *ACM Trans. Graph.*, 34(1), Dec. 2015. 2

[28] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020. 1, 4

[29] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 2

[30] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 5, 6, 7

[31] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 175–184, 2019. 2

[32] Pratul P Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. Learning to synthesize a 4d rgbd light field from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2243–2251, 2017. 2

[33] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020. 4

[34] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019. 2

[35] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d scene representation and rendering. *arXiv e-prints*, pages arXiv–2010, 2020. 1, 2

[36] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. *arXiv preprint arXiv:2004.11364*, 2020. 2

[37] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018. 2, 4

[38] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. *arXiv preprint arXiv:1912.08804*, 2019. 2

[39] Cha Zhang and Tsuhan Chen. Spectral analysis for sampling image-based rendering data. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(11):1038–1050, 2003. 2

[40] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1, 2

[41] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 5, 10

[42] Xiuming Zhang, Sean Fanello, Yun-Ta Tsai, Tiancheng Sun, Tianfan Xue, Rohit Pandey, Sergio Orts-Escolano, Philip Davidson, Christoph Rhemann, Paul Debevec, Jonathan T. Barron, Ravi Ramamoorthi, and William T. Freeman. Neural light transport for relighting and view synthesis. *arXiv preprint arXiv:2008.03806*, 2020. 2

[43] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018. 1, 2, 3

[44] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM transactions on graphics (TOG)*, 23(3):600–608, 2004. 2

A. Additional Implementation Settings

A.1. Image Preparation Details

We calibrate a set of input images using an a Structure-from-Motion (SfM) algorithm in an open-source software package COLMAP [25]. For COLMAP, we use a "simple radial" camera model with a single radial distortion coefficient and a shared intrinsic for all images. We use "sift feature guided matching" option in the exhaustive matcher step of SfM and also refine principle points of the intrinsic during the bundle adjustment. Note that accurate camera poses and intrinsic parameters are crucial for our method, and errors in these parameters can lead to poor results.

A.2. Ray Sampling for Training

During training, generating a reasonable sized output image via the rendering equation for all pixels at once is not feasible due to the memory limit on our GPU. To solve this, we only sample a subset of pixels from the entire image in each iteration of the optimization. And to facilitate the computation of image gradient needed in our loss function, if a pixel (x, y) is sampled in the process, we also sample $(x+1, y)$ and $(x, y+1)$ so that the image gradients in both x and y directions can be computed through finite difference. In our implementation, we sample 2667 sets of these triplet pixels, resulting in 8001 samples.

For evaluation, we use 3 metrics: PSNR, SSIM, and LPIPS. Functions for computing PSNR and SSIM come from scikit-image software package, and for LPIPS, we use a VGG variant from [41] [2].

B. Additional Experimental Details

B.1. Comparison on Real Forward-Facing Dataset

Real Forward-Facing dataset is provided by NeRF [19] and contains 8 scenes. We show a per-scene breakdown of the results from Table 1 in the main paper in Table B.1. These scores from NeRF are computed from undistorted versions of their results using our estimated radial distortion parameter. We provided their original reported scores for reference in Table B.2. A qualitative comparison can be

[2] https://github.com/richzhang/PerceptualSimilarity

seen in Figure 3 in the main paper as well as in our supplementary video, which shows that our method achieves sharper fine detail.

We measured the training time on a single NVIDIA V100 with a 20-core Intel Xeon Gold 6248. For scene Fern with 17 input photos, the training took around 18 hours. For scene Flower with 30 input photos, the training took around 27 hours.

B.2. Comparison on Shiny Dataset

Our own dataset, Shiny, consists of 8 scenes with more challenging view-dependent effects compared to Real Forward-Facing dataset. Table B.3 shows the image resolution and number of images for each scene. To generate results for NeRF, we use the code implemented by the authors[3] using TensorFlow and train on each scene with their default setting for 150k iterations.

We show a per-scene breakdown of the results from Table 2 in the main paper in Table B.4. Our approach achieves better performance than NeRF on all metrics in all scenes. A full visual comparison is provided in our supplementary webpage.

B.3. Comparison on Spaces Dataset

The authors of DeepView have not made their code publicly available, but they have released their output results. So, we run our algorithm on their Spaces dataset and compare our results to theirs. Table B.5 shows a per-scene breakdown of the results from Table 3 in the main paper. A full visual comparison is provided in our supplementary webpage.

B.4. Details for Types of Basis Ablation Study

In Section 4.3.2, we evaluate our algorithm using different sets of basis functions. The experiment is done by changing the neural basis $\vec{H}_\phi$ in Algorithm 1 to other kinds of basis functions such as $\vec{H}_{FS}, \vec{H}_{TS}$ and $\vec{H}_{SH}$.

Our Fourier's basis is similar to the positional encoding used in NeRF [19] and can be computed by:

$$\vec{H}_{FS}(v) = \big[\cos(2^{-1}\pi v_x), \sin(2^{-1}\pi v_x), ..., \cos(2^N \pi v_y), \sin(2^N \pi v_y)\big]. \tag{7}$$

For forward-facing scenarios, the viewing angle v only covers a hemi-sphere. So, v_z can be fully determined from v_x and v_y through $v_z = \sqrt{1 - v_x^2 - v_y^2}$, and we can parameterize the viewing angle with just v_x and v_y and define the FS basis only on these two parameters.

To calculate other basis functions used in Section 4.3.2, let the following complex-valued functions $K_{a,b}^{(m)}$ and $P_{a,b}^{(m)}$ be defined as:

$$K_{a,b}^{(m)}(v) = \Big(\big(\tfrac{v_x}{1-a}\sqrt{\tfrac{v_z-a}{v_z+1}}\big) + \big(\tfrac{v_y}{1-a}\sqrt{\tfrac{v_z-a}{v_z+1}}\big)i\Big)^m \tag{8}$$

$$P_{a,b}^{(m)}(v) = \tfrac{v_z-1}{1-a} + \tfrac{m+1}{b+2m+2} \tag{9}$$

The general form of the set of basis functions is:

$$\begin{aligned}\vec{H}(v) = \Big[&\mathrm{Re}(K_{a,b}^{(2^0)}(v)),\\ &\mathrm{Im}(K_{a,b}^{(2^0)}(v)),\\ &\mathrm{Re}(P_{a,b}^{(2^0)}(v)\cdot K_{a,b}^{(2^0)}(v)),\\ &\mathrm{Im}(P_{a,b}^{(2^0)}(v)\cdot K_{a,b}^{(2^0)}(v)),\\ &...,\\ &\mathrm{Re}(K_{a,b}^{(2^N)}(v)),\\ &\mathrm{Im}(K_{a,b}^{(2^N)}(v)),\\ &\mathrm{Re}(P_{a,b}^{(2^N)}(v)\cdot K_{a,b}^{(2^N)}(v)),\\ &\mathrm{Im}(P_{a,b}^{(2^N)}(v)\cdot K_{a,b}^{(2^N)}(v))\Big]\end{aligned} \tag{10}$$

where if $a = -1, b = 0$, then it reduces to the spherical harmonics basis (SH). If $a = 0, b = 0$, then it reduces to the hemispherical harmonics basis (HSH) [7]. For Jacobi basis (JH), we set $a = \cos(45°) = 1/\sqrt{2}$ and $b = 2$.

Here are examples of the first five terms for each basis that we use in Section 4.3.2:

$$\begin{aligned}\vec{H}_{SH}(v) =& [v_x/2, v_y/2, v_z v_x/2, v_z v_y/2, (v_x^2 - v_y^2)/4, ...]\\ \vec{H}_{HSH}(v) =& [v_x\sqrt{\tfrac{v_z}{v_z+1}}, v_y\sqrt{\tfrac{v_z}{v_z+1}}, v_x\tfrac{2v_z-1}{2}\sqrt{\tfrac{v_z}{v_z+1}},\\ & v_y\tfrac{2v_z-1}{2}\sqrt{\tfrac{v_z}{v_z+1}}, \tfrac{v_x^2-v_y^2}{(1-a)^2}\tfrac{v_z}{v_z-1}, ...]\\ \vec{H}_{JH}(v) =& [v_x\sqrt{\tfrac{v_z-a}{v_z+1}}, v_y\sqrt{\tfrac{v_z-a}{v_z+1}}, v_x\big(\tfrac{v_z-1}{z-a} + \tfrac{2}{b+4}\big)\sqrt{\tfrac{v_z}{v_z+1}},\\ & v_y\big(\tfrac{v_z-1}{z-a} + \tfrac{2}{b+4}\big)\sqrt{\tfrac{v_z}{v_z+1}}, \tfrac{v_x^2-v_y^2}{(1-a)^2}\tfrac{v_z}{v_z-1}, ...]\end{aligned} \tag{11}$$

Figure 5 in the main paper already shows PSNR scores of these basis functions. SSIM and LPIPS scores from the same experiment are shown in figure B.10 and B.11 respectively.

[3] https://github.com/bmild/nerf

Table B.1: Per-scene breakdown results from NeRF's Real Forward-Facing dataset.

	PSNR↑				SSIM↑				LPIPS↓			
	SRN	LLFF	NeRF	Our	SRN	LLFF	NeRF	Our	SRN	LLFF	NeRF	Our
Fern	20.29	23.09	25.49	**25.63**	0.700	0.828	0.866	**0.887**	0.529	0.243	0.278	**0.205**
Flower	23.94	25.81	27.64	**28.90**	0.819	0.907	0.906	**0.933**	0.390	0.168	0.212	**0.150**
Fortress	25.70	29.56	31.34	**31.67**	0.816	0.934	0.941	**0.952**	0.494	0.171	0.166	**0.131**
Horns	23.15	25.13	28.02	**28.46**	0.801	0.905	0.915	**0.934**	0.479	0.197	0.258	**0.173**
Leaves	17.21	19.85	21.34	**21.96**	0.556	0.769	0.782	**0.832**	0.526	0.226	0.308	**0.173**
Orchids	16.97	18.73	**20.67**	20.42	0.575	0.703	0.755	**0.765**	0.528	0.308	0.312	**0.242**
Room	25.63	28.45	32.25	**32.32**	0.908	0.957	0.972	**0.975**	0.351	0.175	0.196	**0.161**
T-rex	21.71	24.67	27.36	**28.73**	0.784	0.903	0.929	**0.953**	0.412	0.204	0.234	**0.192**

Table B.2: (For reference only) Original reported scores from NeRF's paper where test images are not undistorted.

	PSNR↑			SSIM↑			LPIPS↓		
	SRN	LLFF	NeRF	SRN	LLFF	NeRF	SRN	LLFF	NeRF
Fern	21.37	21.37	25.17	0.822	0.887	0.932	0.459	0.247	0.280
Flower	24.63	25.46	27.40	0.916	0.935	0.941	0.288	0.174	0.219
Fortress	26.63	29.40	31.16	0.838	0.957	0.962	0.453	0.173	0.171
Horns	24.33	24.70	27.45	0.921	0.941	0.951	0.376	0.193	0.268
Leaves	18.24	19.52	20.92	0.822	0.877	0.904	0.440	0.216	0.316
Orchids	17.37	18.52	20.36	0.746	0.775	0.852	0.467	0.313	0.321
Room	28.42	28.42	32.70	0.950	0.978	0.978	0.240	0.155	0.178
T-rex	22.87	24.15	26.80	0.916	0.935	0.960	0.298	0.222	0.249

Table B.3: Image resolution and the number of images for each scene in our Shiny dataset. For most scenes, only 20-50 images are enough to produce good results. However, scenes with complex view-dependent effects like CD require more images.

	image resolution	number of images
CD	1008×567	307
Tools	1008×756	58
Crest	1008×756	50
Seasoning	1008×756	45
Food	1008×756	49
Giants	1008×756	32
Lab	1008×567	303
Pasta	1008×756	35

Table B.4: Per-scene breakdown results on our Shiny dataset.

	PSNR↑		SSIM↑		LPIPS↓	
	NeRF	Ours	NeRF	Ours	NeRF	Ours
CD	30.14	**31.43**	0.937	**0.958**	0.206	**0.129**
Tools	27.54	**28.16**	0.938	**0.953**	0.204	**0.151**
Crest	20.30	**21.23**	0.670	**0.757**	0.315	**0.162**
Seasoning	27.79	**28.60**	0.898	**0.928**	0.276	**0.168**
Food	23.32	**23.68**	0.796	**0.832**	0.308	**0.203**
Giants	24.86	**26.00**	0.844	**0.898**	0.270	**0.147**
Lab	29.60	**30.43**	0.936	**0.949**	0.182	**0.146**
Pasta	21.23	**22.07**	0.789	**0.844**	0.311	**0.211**

Ground truth Nex (Ours) NeRF

Figure B.8: A qualitative comparison on Shiny dataset between ground truth(left), NeX (center), and NeRF[19] (right). A full comparison on all scenes can be found in our supplementary webpage

Ground truth Ours NSVF

Figure B.9: A qualitative comparison on scene CD between ground truth (left), NeX (center), and NSVF [15] (right). We use NSVF code open-sourced by the authors[4]. NSVF does not perform well for this problem setup because it focuses on object captures where a bounding volume can be tightly defined.

Table B.5: Per-scene breakdown results on Spaces dataset.

	PSNR↑			SSIM↑			LPIPS↓		
	Soft3D	DeepView	Ours	Soft3D	DeepView	Ours	Soft3D	DeepView	Ours
scene000	32.66	32.54	**37.61**	0.971	0.983	**0.989**	0.093	0.059	**0.049**
scene009	31.46	31.07	**35.40**	0.962	0.972	**0.981**	0.123	0.091	**0.080**
scene010	32.94	31.22	**37.61**	0.973	0.979	**0.989**	0.137	0.095	**0.095**
scene023	31.52	31.14	**35.69**	0.969	0.978	**0.986**	0.142	0.102	**0.098**
scene024	33.88	33.15	**37.77**	0.978	0.983	**0.989**	0.119	**0.081**	0.090
scene052	30.08	30.22	**34.02**	0.947	0.971	**0.979**	0.119	0.081	**0.076**
scene056	30.64	31.04	**34.77**	0.956	0.975	**0.981**	0.141	0.087	**0.087**
scene062	32.56	32.07	**35.34**	0.969	0.980	**0.984**	0.151	**0.098**	0.121
scene063	29.72	32.72	**35.44**	0.952	0.979	**0.987**	0.122	0.078	**0.073**
scene073	30.28	30.85	**34.81**	0.960	0.977	**0.986**	0.111	0.073	**0.065**

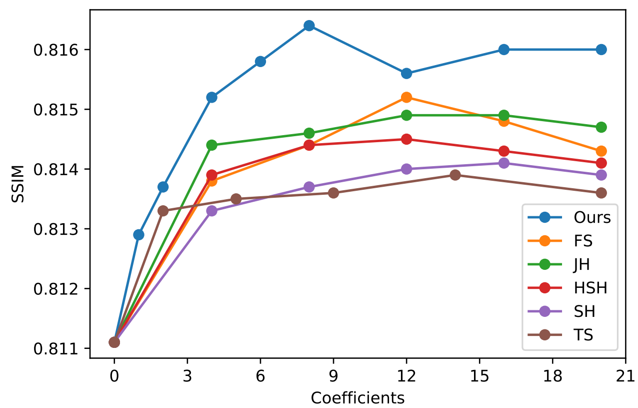


Figure B.10: Number of coefficients versus SSIM score (higher is better)

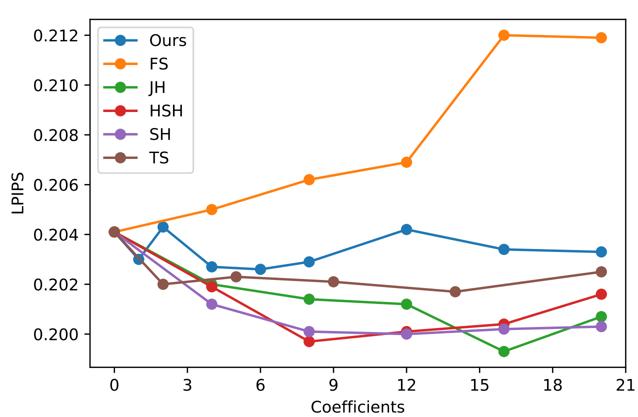


Figure B.11: Number of coefficients versus LPIPS score (lower is better)