This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

NPBG++: Accelerating Neural Point-Based Graphics

Ruslan Rakhimov[1*] Andrei-Timotei Ardelean[1*] Victor Lempitsky[1,2] Evgeny Burnaev[1,3]

[1]Skolkovo Institute of Science and Technology, Russia
[2]Yandex, Russia [3]Artificial Intelligence Research Institute, Russia

Abstract

We present a new system (NPBG++) for the novel view synthesis (NVS) task that achieves high rendering realism with low scene fitting time. Our method efficiently leverages the multiview observations and the point cloud of a static scene to predict a neural descriptor for each point, improving upon the pipeline of Neural Point-Based Graphics [1] in several important ways. By predicting the descriptors with a single pass through the source images, we lift the requirement of per-scene optimization while also making the neural descriptors view-dependent and more suitable for scenes with strong non-Lambertian effects. In our comparisons, the proposed system outperforms previous NVS approaches in terms of fitting and rendering runtimes while producing images of similar quality. Project page: https://rakhimovv.github.io/npbgpp/.

1. Introduction

The ability to render photorealistic views of a scene, as learned from a handful of observations, opens the door to many applications in virtual / augmented reality, cinematography, gaming industry, and virtually any field which involves computer graphics. While there is a high interest in creating such novel view synthesis (NVS) systems, the problem proves to be challenging. Early methods based on view interpolation [4,16,26] do not fare well enough in real-world scenarios, which entail complex geometry, limited proximity of input images, lighting variations, etc. There are mainly three development directions which the different works addressing this task generally seek to improve: rendered image quality, scene fitting time, and rendering speed. Despite the recent development in Computer Vision brought by Deep Learning approaches, there is still a noticeable gap between the current state of the art in NVS and an ideal model for this task.

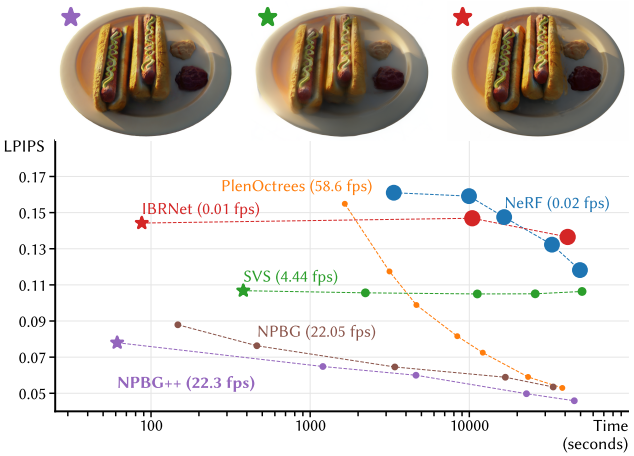

Figure 1. **Time comparison.** Time vs. image quality (LPIPS) comparison of several methods, computed on the 'hotdog' scene from the NeRF-synthetic dataset. The time axis represents the time-to-rendering, i.e., fitting time + rendering time for one image. For methods marked with ⋆, the first scores are reported without per-scene optimizations. Fitting time consists of feature extraction for IBRNet, geometry estimation + 3D modeling stage for NPBG++, and geometry estimation + meshing for SVS (the renderings on top offer qualitative comparisons between these configurations). The remaining scores are computed at different points in the fine-tuning processes. Circle areas are proportional to logarithms of the rendering times (smaller is better) and highlight the methods' rendering speed.

To this end, our work proposes a new system that enables real-time rendering and can rapidly adapt to new scenes. Similarly to *Neural Point-Based Graphics* (NPBG) [1], our approach uses point clouds to model the geometry of scenes. This representation is advantageous as point clouds are generally available by using inexpensive RGBD cameras or by processing RGB streams with classic Structure-from-Motion and Multi-View Stereo pipelines such as COLMAP [35, 37]. These reconstructions are usually not accurate or complete enough to be used directly for rendering, whereas performing surface estimation could yield a loss of geometric details and requires significant additional

*Equal contribution
Correspondence to ruslan.rakhimov@skoltech.ru

computations. Instead, we devise our method to work directly on raw point geometry and address the problem of small noise and low point density using neural rendering.

Using a point-based geometry together with a neural rendering model had been shown to yield good results as NVS systems [1, 19–21]. However, these approaches require per scene optimization of per-point descriptors and optionally a deep rendering network, resulting in a lengthy process to achieve high-quality renderings. To accelerate this process, our method predicts the points' features from the source images, enabling fast scene representations, which can then be rendered in real-time. If needed, one can further finetune these predictions to increase the quality of the result. The challenge of the approach is the proper integration of data from several views while considering view-dependent appearance, occlusions, and missing information.

The system we propose is effective and fast, see Figure 1: for each input image and associated camera parameters, we run the feature extractor network on it, yielding feature maps representing each pixel's local appearance. The point cloud is then projected onto the image, taking into account occlusions to obtain features. An online aggregation method was devised to effectively aggregate the obtained features from one image at a time. After processing all input views, we get the final view-dependent neural descriptors for each point. This representation is then rendered by a U-Net shaped network that integrates multi-scale rasterizations of the point cloud similarly to NPBG [1].

To further improve the quality of the NPBG system, we introduce two important modifications (that can also be applied in analogous systems). Firstly, we show how view-dependency can be added to neural descriptors efficiently, without an excessive increase of scene fitting time, while boosting the quality of NPBG for scenes with non-Lambertian surfaces. Secondly, we introduce two lightweight 2D alignment stages in the pipeline that address the non-equivariance of convolutional parts of the pipeline to in-plane rotations.

To summarize, our main contributions are:

- a new NVS system capable of quickly generating neural scene representations that can then be rendered at interactive rates at high resolution.
- an online, permutation-invariant, aggregation method for incorporating features from any number of source views using constant memory into neural descriptors, which facilitates view-dependent effect modeling.
- an alignment technique that makes the rendering process equivariant to in-plane rotations, appropriate for any pipeline working with neural descriptors estimated from images.

2. Related work

Novel view synthesis. The computer vision and graphics communities have long been interested in the problem of novel view synthesis and, specifically, image-based rendering. Several approaches have been developed that differ in how they leverage and represent the underlying geometry of the scenes. Ranging from methods like light field rendering [22] that do not rely on an explicit geometric proxy and require a dense set of image samples instead [39], to methods that rely on accurate geometry for synthesis. More recently, the interest amplified around the idea of applying deep learning models to replace or enhance parts of the classic pipelines. Among others, neural networks were employed to predict blending weights for composing input images [12], predict camera poses [59], depth maps [15], multi-plane images [9, 60], and voxel grids [17]. Moreover, several recent methods harness neural scene representations to build upon standard 3D scene reconstructions using different proxy geometries such as point clouds [1, 27, 30] or meshes [32, 33, 44]. An important breakthrough was brought by the introduction of Neural Radiance Fields (NeRF) [28] where it is proposed to model the entire scene continuously using a fully connected network that is optimized using differentiable volume rendering. The approach has lately seen many variants which address some of its limitation: the large number of required views [5, 55], inter-observations variance [25], rendering speed [10, 24, 29, 54] and scene fitting time [3, 5, 42, 55].

NVS systems can be generally divided into two categories: methods that require timely per-scene optimization [1, 28, 50] and methods that can easily and rapidly generalize to new scenes by learning to integrate information from the input views efficiently [3, 5, 33, 47, 55]. Most systems in the second category also allow fine-tuning for particular scenes making them more versatile and preferable to the first group. Our proposed system, NPBG++, belongs to the latter category, as it can aggregate information from all available observations in a single pass.

Point-based graphics. The use of points as a rendering primitive was identified early on [7, 23] and sparked interest due to the simplicity and efficiency it conveys. While they are fast and have little redundancy, point clouds have been shown to be an appropriate representation that allows rendering complex objects in real-time [11]. Recently, point cloud representations were combined with deep learning methods to obtain realistic views even in the presence of noise or sparsity in the point clouds [1, 27, 30, 51]. Other approaches that use a generative neural network for refining the results were also developed for methods that use surfels (oriented planar disks) [2] and spheres instead of points [21]. Moreover, the use of deep learning in conjunction with point clouds was facilitated by the develop-

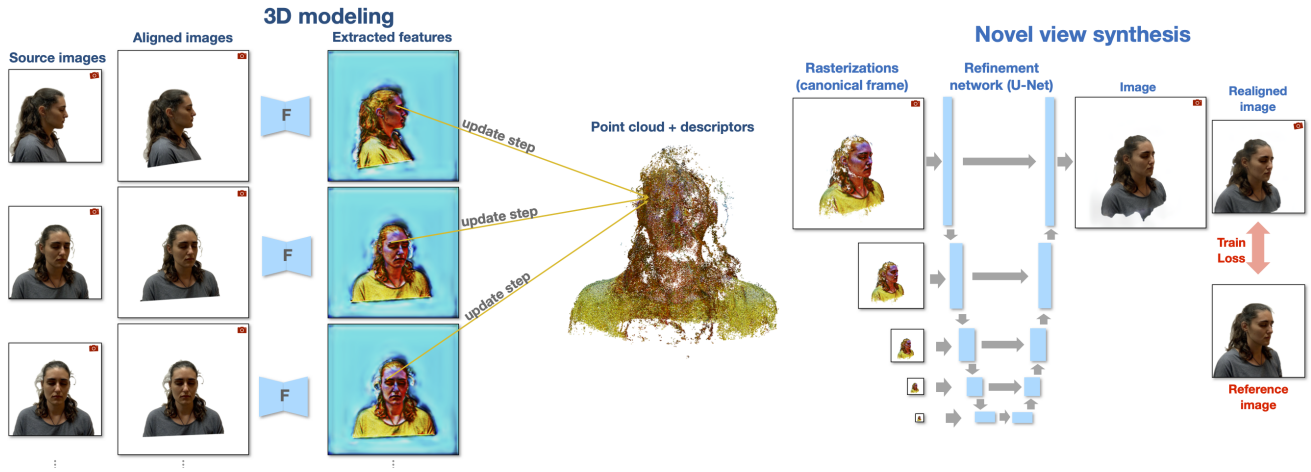

Figure 2. **Overview of NPBG++: the system for fast novel view synthesis.** We represent the scene as a point cloud with a view-dependent neural descriptor embedded in each point. During 3d modeling stage (Sec. 3.1), we sequentially process each input view (input image alignment and feature extraction) and apply online aggregation to update the neural descriptors of each point (no fitting). During the novel view synthesis stage (Sec. 3.2), we rasterize the point cloud, pass the rasterization result through the rendering network, and post-process it (output image alignment) to get the novel view.

ment of differentiable rendering models which allow optimizing neural descriptors [1], the opacity of points [19], their positions [49], and spheres' radii [21]. Our work uses a point-based geometry; however, a generic method for integrating image features was developed instead of optimizing per point descriptors. This extends the single-view feedforward point-based approach to new view synthesis proposed in SynSin [49].

Light field factorization. In order to handle specular effects, a view-dependent rendering system must be employed. A popular approach used to model such effects is to rely on Spherical Harmonics to represent spherical functions [41]. Spherical Harmonics and Spherical Gaussians have been used to speed up inference of Neural Radience Fields by factorizing the view-dependent appearance [54]. A similar idea was used in the work of Wizadwongsa et al. [50], where several basis functions were investigated, such as Hemi-spherical harmonics, Fourier Series, and Jacobi Spherical Harmonics with best results achieved by learnable basis functions. For more detailed information, we refer the reader to a recent overview [43].

3. Method

Given a set of multiview input images, associated camera parameters, and a point cloud of a static scene, our system generates images from novel views. To do this, neural descriptors, latent vectors representing local geometric and photometric properties, are computed from the information contained in the input views. The latent descriptors are later rasterized using geometric point positions and are converted into final rendering using a refiner (renderer) convolutional network. In contrast to NPBG [1], our system is learning-based: we predict the neural descriptors instead of optimizing them for each new scene (Figure 2).

In contrast to image-based approaches that find the nearest views from the input set of images to render a novel view, our system creates a single scene model. To construct the model, we process input views in online mode, one at a time, updating the intermediate states of the points in each iteration. These intermediate states are independent of the number of processed views, allowing us to process the scene in constant memory. After processing all input views, we compute final descriptors using the intermediate states. We discuss the modeling stage below, and the rendering process as well as the learning process after that.

3.1. Modeling 3D Scenes

Feature extraction. The feature extractor, a U-Net-based network [34], takes as input an image and outputs a dense feature map (descriptors for each pixel). The feature map has the same spatial size as the input image. The number of channels is equal to the neural descriptor size (c=8). We project the points onto the camera canvas and bilinearly sample the descriptors. We use these descriptors to update the intermediate states of the points.

Input image alignment. The bilinearly sampled feature descriptor contains information about the surrounding local patch. The descriptor will change if one rotates the input image since the feature extractor network is not rotation-equivariant by default. We want to bring more consistency across the points' descriptors obtained from different views. For this purpose, we could consider the following options:

(i) design the system to be rotation-invariant though this would restrict the representative power of descriptors, (ii) use rotation-equivariant feature extractor [48] though this would inevitably affect the complexity of the feature extractor and/or reduce its capacity, (iii) rotate the input image to a canonical orientation before applying the feature extractor. The last option is the approach we take, i.e., we rotate the input image to align with a *canonical* orientation. We define such orientation so that the world up-axis's projection onto the image plane has a vertical direction (pointing top) in the pixel space (Figure 2-left). We use zero-padding and expand the image size to preserve all image content during the rotation (we change the camera intrinsics and extrinsics to take care of the padding). We call this procedure *input image alignment*.

Estimating point's visibility. To avoid updating descriptors for occluded points, we approximately estimate the visibility of each point. We do this by building a Z-buffer and rasterizing the point cloud onto the image of reduced size $\frac{h}{2^r} \times \frac{w}{2^r}$. We consider visible only the points with the minimum Z-value for a pixel location. The visibility reduction factor is set as $r{=}0$ if the point cloud is dense and increased otherwise. In this process, we apply the "nearest" rasterization scheme (Sec. 3.2). While this procedure estimates the visibility approximately, it is fast in contrast to, e.g., constructing the visual hull first like in [18], and does not require to tune the radius of the points as in more advanced rasterization schemes [21, 49].

Aggregation. First, given a new sampled descriptor from the currently processed input view, we want to either set up or update the intermediate states for a point. After all input views are processed, we get the final neural descriptor for the point by exploiting the intermediate states. These two steps completely define the aggregation procedure.

We want to process the incoming descriptors from new input views in online mode: working with one new sample of information at a time and to be memory-independent from the number of input views. Therefore it is not well-suited to consider: (i) Transformer-based [45] aggregation due to memory limit, (ii) LSTM [13] and GRU [6] recurrent networks as they introduce the undesirable dependence on the order of the input views. Conversely, we design our method to be permutation invariant. One viable option is average or maximum aggregation. However, the drawback of this approach is the loss of information about the view orientations of incoming descriptors, which, as we show, hinders the ability to model view-dependent effects.

Note that view-dependent effects can also be handled by the refiner (renderer) network that can consider view directions during new view synthesis [21, 44]. However, this requires a more complex and slow refiner network. Alternatively, we choose to make the neural descriptor of each point view-dependent.

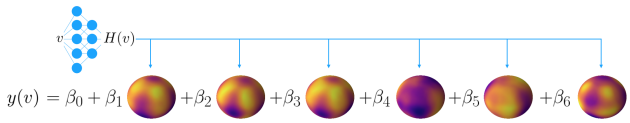

Figure 3. **View-dependent neural descriptor.** We model the view-dependent neural descriptor y: $\mathbb{R}^3{\to}\mathbb{R}^c$ as a linear combination of learnable basis functions over the sphere (H: $\mathbb{R}^3{\to}\mathbb{R}^m$) with coefficients $\beta_i \in \mathbb{R}^c$ (see Equation 1 in the main text). For a new scene, given a set of source images, we find β_i for each point

In more detail, we model the neural descriptor y: $\mathbb{R}^3{\to}\mathbb{R}^c$ ($c{=}8$) for the point as a linear combination of learnable basis functions over the sphere:

$$\underset{1\times c}{y(v)} = \underset{1\times m}{H(v)}\ \underset{m\times c}{\beta}\ + \underset{1\times c}{\beta_0} \tag{1}$$

where v is an unit length view direction, H: $\mathbb{R}^3{\to}\mathbb{R}^m$ represents a set of m (we use $m{=}6$) basis functions. $H(v)$ can represent the spherical harmonics (SH) bases, but we model it as the multilayer perceptron (MLP) with weights shared across all points. We found this setting to be superior to SH, see Sec. 4. Coefficients β and β_0 are to be found and are different for each point. See the graphical illustration in Fig. 3.

The described approach is similar to NEX [50] where they model view-dependent RGB values instead of neural descriptors. For each new scene, they optimize β_0 and an MLP, which takes as input the position of the point and outputs β. In contrast to NEX [50], to find coefficients β and β_0 for all N points, we solve N multivariate linear regression problems.

For each point, we have a set of pairs $\{(v_k, y_k)\}_{k=1}^K$, where K is the number of input views in which we estimate the point to be visible (K might be different for different points), $v_k \in \mathbb{R}^3$ is a unit-length view direction, $y_k \in \mathbb{R}^c$ is a sampled descriptor from the input image. Given this, we find the parameters of the descriptor as follows:

$$\underset{1\times c}{\beta_0} = \frac{1}{K}\sum_{k=1}^{K} \underset{1\times c}{y_k} \tag{2}$$

$$\underset{m\times c}{R} := \frac{1}{K}\sum_{k=1}^{K} \underbrace{H(v_k)^T y_k}_{m\times c} - \frac{1}{K}\sum_{k=1}^{K} \underbrace{H(v_k)^T \beta_0}_{m\times c}$$

$$\underset{m\times c}{\beta} = \left(\frac{1}{K}\sum_{k=1}^{K} \underbrace{H(v_k)^T H(v_k)}_{m\times m} + \frac{\alpha}{K}\ \underset{m\times m}{I_m}\right)^{-1} \underset{m\times c}{R} \tag{3}$$

where I_m is the identity matrix, β_0 captures the mean descriptor, and we set the regularizer $\alpha{=}1$. When a new descriptor sample y_k arrives we update five intermediate states: K, $\sum_{k=1}^K y_k$, $\sum_{k=1}^K H(v_k)^T y_k$, $\sum_{k=1}^K H(v_k)^T$,

$\sum_{k=1}^{K} H(v_k)^T H(v_k)$. Note that the tensor size of intermediate states does not include the number of input views K. For each point, we update its intermediate states until all input views are processed. Then we compute the coefficients β and β_0. We exclude points from the point cloud that were not considered visible in any input view.

3.2. Novel view synthesis

Given the target camera parameters, the final rendering onto the canvas associated with it consists of three steps:

Descriptors calculation step. We calculate descriptors using Equation 1.

Rasterization step. Following NPBG [1], we construct a pyramid of rasterized *raw* images $\{S_t\}_{t=1}^{T}$ (T=5 in all our experiments), where each $S_t \in \mathbb{R}^{\lfloor \frac{h}{2^{t-1}} \rfloor \times \lfloor \frac{w}{2^{t-1}} \rfloor \times (c+1)}$ is formed by assigning to every pixel the neural descriptor of the point which passed the depth test, which projects to it under camera full projection transform, and a binary scalar, which indicates a non-empty pixel. The image with the highest resolution in this pyramid provides fine details, while the image with the coarsest resolution suffers the least from surface bleeding and guides the implicit hole filling process inside the refiner network. We fill the pixels that do not have a point projected to them with zeros instead of using a learnable "void" descriptor, as it was originally proposed in [1]. In our experiments, this setting leads to better generalization to new views. Alternative rasterization schemes, e.g., soft rasterization (SynSin [49]) and sphere tracing (Pulsar [21]) could be used, still we use the NPBG rasterization due to its speed and lack of additional tunable parameters.

Refinement step. Following NPBG [1], we process the rasterizations with the refiner network that has a U-Net [34] architecture with gated convolutions [56]. The refiner network thus takes as an input the rasterized *raw* image S_1 and also appends $S_2, \ldots, S_T$ to the input of the corresponding size in intermediate layers of the encoder network. These intermediate inputs guide the network to contend the bleeding surface. The refiner outputs the final RGB image.

Output image alignment. We make a *modification* to the rasterization and refinement steps. Since the refiner network is convolution-based and not rotation-equivariant, the rendered patch will look differently depending on the target camera's y-axis orientation. This is overlooked by previous approaches working with neural descriptors [1, 21, 49]. Therefore, analogously to the input image alignment (Sec. 3.3), we first rasterize and render our final image to the canonical orientation and then rotate the produced image to align with the original orientation (Figure 2-right).

3.3. Training

We select the batch of target views from different scenes and randomly crop patches of the same size during each training iteration. We use patches instead of entire images to reduce memory load during training. For each patch, during training only, we select three relevant views from which we aggregate descriptors and then render the final image.

View selection. To select the relevant views, we follow the approach from MVSNet [52]: we calculate a score for each input view by taking a sum of scores of points that are visible both in the target patch and the input view. We stress that view selection is used only to improve training and is not used after training, i.e., during scene fitting or new view synthesis. The point score is angle-based: the smaller is the angle between the tracks connecting two camera centers and the point, the higher the score. For the exact formula, we refer the reader to [52]. The input view scores define the probabilities of a discrete multinomial distribution from which we sample the indices of relevant input views. We further modify the original procedure as follows. As we want to avoid the case when all selected images are similar and do not cover some part of the target image, we do not sample all relevant views at once but do that sequentially. After selecting the first view, we remove the points visible in this image and recalculate the scores for the remaining views. We repeat the selection and removal until we get the desired number of relevant views.

Cropping. To avoid running the feature extractor network on the selected images at the original large size, we crop them such that the crops contain as many points from the target patch as possible.

Loss. We employ the following train loss:

$$\mathcal{L}(I, I_{\text{gt}}) = \lambda_1 \mathcal{L}_{\text{vgg}}(I, I_{\text{gt}}) + \lambda_2 \mathcal{L}_1(I \downarrow_4, I_{\text{gt}} \downarrow_4) + \lambda_3 \mathcal{L}_{\text{reg}}(I_{\text{gt}})$$

where I is a rendered image, I_{gt} is a reference ground-truth image. Similar to [1, 38] $\mathcal{L}_{\text{vgg}}$ is a VGG-19 [40] perceptual loss and $\mathcal{L}_1$ loss is used to match four times bilinearly downsampled versions of the images to prevent high-frequency detail smoothing while encouraging color preservation. Additionally, we introduce a self-supervised regularization loss $\mathcal{L}_{\text{reg}}$ that improves the learning signal for the refiner network R by providing high-quality descriptors extracted directly from the ground truth image (unavailable at test time).

$\mathcal{L}_{\text{reg}}(I_{\text{gt}}) = \mathcal{L}_1(R(\text{pyr}(sg(F(I_{\text{gt}})))), I_{\text{gt}})$, where F is a feature extractor, $\text{pyr}(\cdot)$ takes as input the dense output from the feature extractor F and outputs a pyramid of images. The first image in the pyramid is $F(I_{\text{gt}}) \in \mathbb{R}^{h \times w \times c}$. Each new level image is obtained using 2×2 average pooling of the previous one. sg (stop-grad) is the non-differentiable version of the identity function and is used to avoid shortcut solutions e.g. by directly passing the original image through the network. We set λ_1=1, λ_2=2500, λ_3=1000.

Method	Per scene optimization	Nerf-Synthetic			ScanNet			DTU			H3DS		
		PSNR↑	**SSIM↑**	**LPIPS↓**	**PSNR↑**	**SSIM↑**	**LPIPS↓**	**PSNR↑**	**SSIM↑**	**LPIPS↓**	**PSNR↑**	**SSIM↑**	**LPIPS↓**
SVS [33]	✗	22.81	0.919	0.104	23.32	**0.771**	**0.445**	20.98	0.897	0.162	18.96	0.798	0.210
IBRNet [47]	✗	**29.47**	**0.955**	0.157	**23.34**	0.760	0.494	**25.81**	**0.924**	0.231	20.30	0.791	0.279
NPBG++ (Ours)	✗	26.06	0.936	**0.071**	23.11	0.766	0.502	23.23	0.915	**0.154**	**21.80**	**0.818**	**0.177**
NPBG [1]	✓	28.62	0.946	0.058	25.09	0.737	0.459	26.00	0.913	0.125	24.68	0.827	0.146
NeRF [28]	✓	32.49	0.970	**0.041**	**25.74**	**0.780**	0.537	**26.92**	0.913	0.198	23.88	0.833	0.178
SVS_{ft} [33]	✓	23.37	0.919	0.101	22.31	0.610	0.543	20.72	0.864	0.190	20.12	0.770	0.197
$IBRNet_{ft}$ [47]	✓	**32.51**	**0.972**	0.144	24.42	0.774	0.493	23.80	0.917	0.222	24.68	**0.850**	0.195
NPBG++$_{ft\text{-}system}$ (Ours)	✓	26.24	0.940	0.064	23.48	0.768	0.490	24.05	0.919	0.147	23.79	0.836	0.155
NPBG++$_{ft}$ (Ours)	✓	28.67	0.952	0.050	25.27	0.772	**0.448**	26.08	**0.928**	**0.123**	**24.91**	0.845	**0.137**

Table 1. **Quantitative evaluations.** For each dataset, we compute the metrics [58] on holdout frames averaged across holdout scenes. Subscript *ft* indicates finetuned versions of the methods. In the case of NPBG++$_{ft}$ we directly finetune coefficients (β, β_0) and the refiner. In the case of NPBG++$_{ft\text{-}system}$ we finetune the feature extractor, aggregator (MLP: neural basis functions), and refiner.

4. Experimental Results

Datasets. We validate the effectiveness of the proposed method on four different datasets: ScanNet [8], NeRF-Synthetic [28], H3DS [31], and DTU [14]. For H3DS and DTU, we apply masks to all images to make the background white. To obtain the point-cloud geometry in the case of ScanNet, we use the depth data in the same manner as NPBG [1]. For Nerf-synthetic, DTU, and H3DS, we use the multi-view stereo approach PatchmatchNet [46] which offers a good balance between speed and point cloud quality. We have found it to be fast and yield results with more surface coverage compared to other MVS approaches such as COLMAP [36]. For further details about the datasets, we refer the reader to the Supplementary materials.

Training Details. We consider only four different scenes in each epoch to accelerate training time by caching the input point clouds. Each iteration, we sample eight target patches of size 256x256. Each epoch lasts 2000 iterations. To avoid overfitting, we apply color augmentations for the target patch and input view patches. We train our system (the feature extractor, aggregator, and refiner) on the ScanNet dataset. We start from the ScanNet pretraining for all other datasets and fine-tune the system on the training scenes for that dataset. For implementation details we refer the reader to the Supplementary materials.

Compared methods. We compare our results with several state-of-the-art neural rendering algorithms.

- **NPBG [1]:** a neural-point-based graphics approach aimed at per-scene optimization. We reimplement the original network in a faithful manner and pretrain the refiner network on the same set of scenes as our pipeline to provide a fair comparison.

- **NeRF [28]**: a volume-rendering-based approach aimed at per-scene optimization. We use the slightly faster PyTorch implementation [53].

Method	H3DS	DTU	Nerf-Synthetic
NPBG++ w/ H3DS ft	**0.177**	0.176	0.102
NPBG++ w/ DTU ft	0.209	**0.154**	0.093
NPBG++ w/ Nerf ft	0.212	0.164	**0.071**

Table 2. **Cross-dataset generalization.** We report LPIPS↓ on holdout frames averaged across holdout scenes for each dataset.

- **SVS [33]**: Stable View Synthesis uses a geometric scaffold on the surface of which it aggregates image features. The method is capable of rendering new scenes without optimization; further fine-tuning of the system is optional.

- **IBRNet [47]:** an image-based rendering approach that learns a generic view interpolation function applicable to novel scenes.

Evaluation. Table 1 shows the quantitative comparisons with the state-of-the-art on several standard metrics (SSIM, PSNR, and LPIPS [58]). Figure 4 shows the qualitative comparisons. Table 2 demonstrates the generalization ability of our method.

The metrics for results obtained without per-scene optimization show that our method can produce superior renderings to SVS and, generally, on-par with IBRNet - the state-of-the-art NVS method with fast generalization to new scenes. While in some metrics, IBRNet is better on certain scenes, the rendering speed advantage of NPBG++ over IBRNet is drastic ($\approx 1000\times$).

The numerical results in the fine-tuning case affirm that our view-dependence modeling is effective, allowing us to outperform NPBG on all datasets. The proposed approach obtains leading scores on DTU and H3DS scenes and is a close competitor on ScanNet and NeRF-Synthetic datasets. On the latter, our method is limited by the lower quality geometry estimated by the MVS system and could potentially

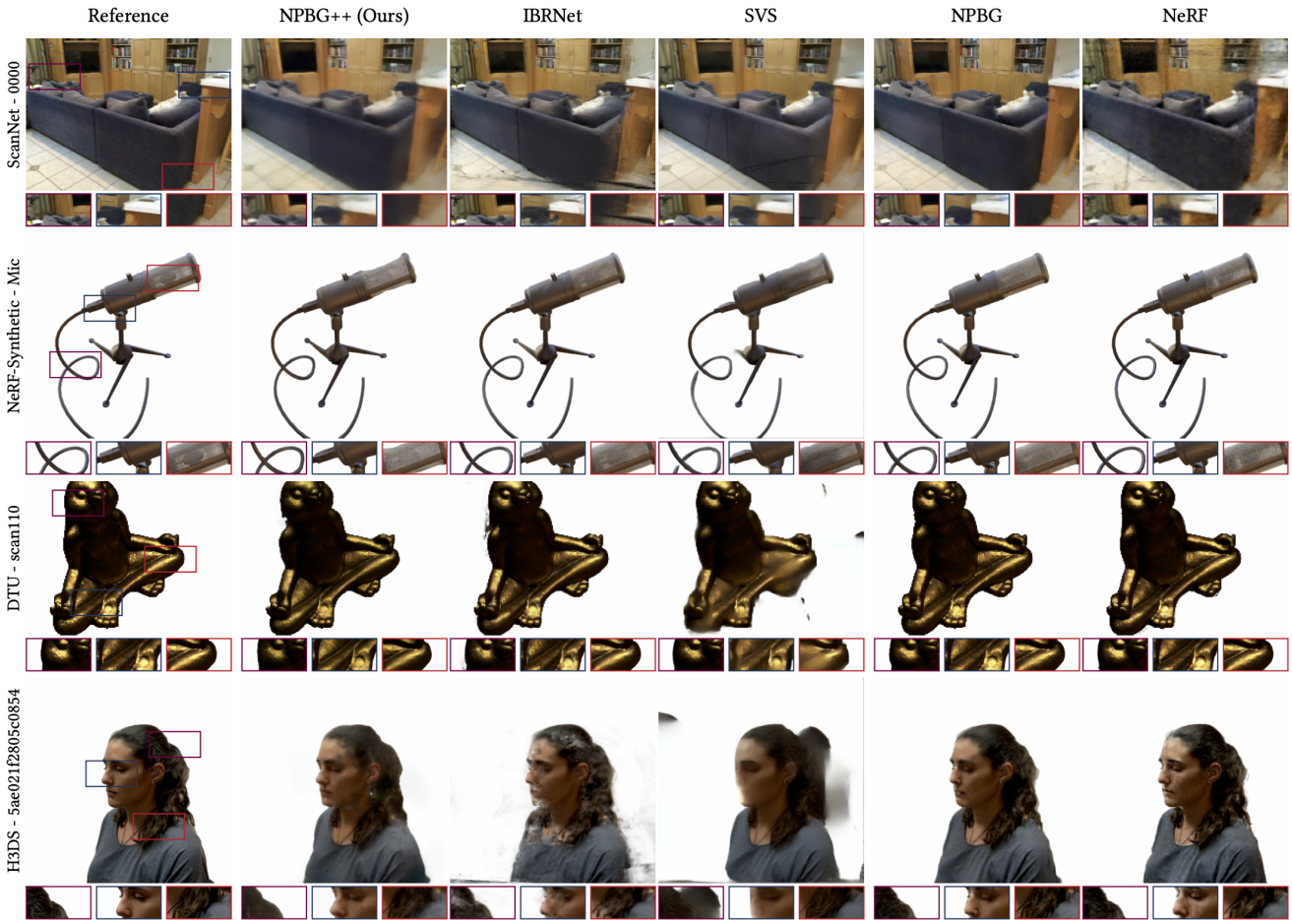

Figure 4. **Qualitative evaluations.** Comparisons with otimization-based approaches (NPBG [1], NeRF [28]) and learning based approaches (IBRNet [47], SVS [33]) on ScanNet [8], NeRF-Synthetic [28], DTU [14], H3DS [31] scenes.

be improved by using a rasterization scheme differentiable w.r.t to points' positions [21, 49].

The qualitative comparison reveals that our method can generate marginally blurry but comprehensive and consistent views of the scenes. Images rendered by IBRNet can introduce artifacts when parts of the image are not included in enough source images (e.g. on `ScanNet-0000`). This may happen if there are only a few training images or if the view selection method underperforms. Note that all methods from the official implementation were tested, and the reported images and scores are obtained with the best performing one. NPBG++ does not suffer from this problem because it aggregates features from all images. SVS displays difficulties in representing thin structures (mic - NeRF synthetic) and often produces spurious artifacts in object-centric scenes. NPBG generally yields high-quality results; however, it cannot handle view-dependent effects by design, leading to bland results for shiny surfaces. NeRF offers generally good results, but the high fitting and rendering time make it impractical for many use-cases.

Runtime analysis. For inference, there are two different stages for which we compare the speed of several state-of-the-art methods. In the first (fitting) stage, the algorithms capture information from the source images. For methods based on per-scene optimization, this generally means training the neural representation of the scene. For IBRNet, it means running the feature extractor on the selected neighboring views. For our method, it refers to the 3D modeling stage (Sec. 3.1). Approaches that use geometric proxies, such as SVS, NPBG, and NPBG++, need to construct the 3D representation as part of this stage. The time required for this process is included in time measurements to provide a fair comparison. This first step is performed a single time per scene, after which each model should be able to render any number of novel views, i.e., the second stage (rendering). As it can be observed in Figure 1, NeRF and IBRNet have very high rendering times. Notably, the time needed to render a novel image using IBRNet is larger than the en-

Figure 5. **Robustness with respect to the point cloud sparsity.** We randomly drop different amounts of points from the point cloud. In each case, we show the percentage of filled pixels after rasterization. The effect is illustrated on scan118 from DTU.

Figure 6. **Robustness with respect to geometry noise.** We move every point along a random unit direction by a specified value for each case. We report LPIPS↓ averaged across holdout frames. The effect is illustrated on scan118 from DTU.

tire fitting process of our method. NeRF, PlenOctrees, and NPBG require per-scene fitting, which leads to larger times until good quality results are obtained. Out of the methods which do not require optimization, our model has the smallest time-to-render (fitting + rendering) time: SVS is slowed down by surface estimation, and IBRNet's timing is dominated by the rendering step.

Ablations. First of all, we demonstrate the impact of input and output image alignment. To demonstrate it, we run different modifications of our system and NPBG [1] (Table 4). Adding output image alignment to NPBG improves metrics. In the case of NPBG++, using either only input or only output image alignment makes metrics worse. This is likely caused by the misalignment between distributions of the camera's y-axis orientations in input and output images. However, turning on both input and output alignments tackles this problem and improves metrics. Second, we test the robustness of our system to geometry imperfections, such as the sparsity of the point cloud (Figure 5), noise (Figure 6) and also few images scenario (Figure 7). The method shows graceful degradation in all cases. Third, we compare different variants for the aggregation procedure (Table 3). We observe that the view-dependent variants outperform the simple average aggregation. In the case of spherical harmonics (SH), we compare only with the 4 harmonics setting due to memory constraints. The best results are obtained using the MLP with m=6 basis functions.

5. Summary and Limitations

In this paper, we introduced NPBG++, a new system for novel view synthesis that can rapidly obtain the neural representation of a scene, passing a single time through all available source views. We obtain the coefficients for learned basis functions by solving online linear regression

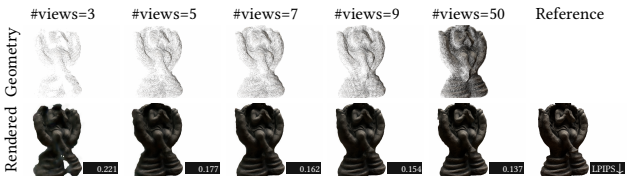

Figure 7. **Robustness with respect to the number of input views.** We report LPIPS↓ averaged across holdout frames. The effect is illustrated on scan118 from DTU.

	PSNR↑	SSIM↑	LPIPS↓
average	21.41	0.749	0.525
spherical harmonics (m=4)	22.19	0.753	0.513
mlp (m=3)	22.97	0.756	**0.500**
mlp (m=6) (default)	**23.11**	**0.766**	0.501

Table 3. **Aggregation.** We test different options for the aggregation procedure as discussed in Sec. 3.1. We report metrics averaged across three holdout ScanNet{0,43,45} scenes.

Method	Input Image Alignment	Output Image Alignment	PSNR↑	SSIM↑	LPIPS↓
NPBG (original) [1]	n/a	✗	26.00	0.913	**0.125**
NPBG	n/a	✓	**26.16**	**0.920**	0.126
NPBG++	✗	✗	22.53	0.912	**0.149**
NPBG++	✓	✗	21.87	0.876	0.201
NPBG++	✗	✓	22.47	0.879	0.203
NPBG++ (default)	✓	✓	**23.23**	**0.915**	0.154

Table 4. **The impact of alignment.** We provide empirical evidence on how the input and output image alignment (Sec. 3.1, Sec. 3.2, Figure 1) improve the final result for neural point-based graphics pipelines. We report metrics averaged across three holdout DTU scenes{110,114,118}. In the case of NPBG, a separate network was trained for each scene and each alignment.

that allows us to model view-dependent neural descriptors. These descriptors are then used to render novel views from arbitrary camera poses easily. We showed that our system can produce good quality results at a fraction of the time required for other methods while also achieving real-time rendering for medium resolution images.

NPBG++ inherits some limitations from NPBG in that it still requires an explicit underlying geometric model (point cloud). In the case of highly erroneous point clouds or inaccurate camera alignments, as well as extreme scale changes, renderings can suffer from blurriness. We additionally introduced an α hyperparameter in Equation 3 but keeping the same value (α=1.0) for all points may over-regularize the solution and dampen view-dependent effects.

Acknowledgment. The Authors acknowledge the use of computational resources of the Skoltech CDISE supercomputer Zhores [57] for obtaining the results presented in this paper. The work was supported by the Analytical center under the RF Government (subsidy agreement 000000D730321P5Q0002, Grant No. 70-2021-00145 02.11.2021).

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pages 696–712. Springer, 2020. 1, 2, 3, 5, 6, 7, 8

[2] Giang Bui, Truc Le, Brittany Morago, and Ye Duan. Point-based rendering enhancement via deep learning. *The Visual Computer*, 34(6):829–841, 2018. 2

[3] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. *arXiv preprint arXiv:2103.15595*, 2021. 2

[4] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1993. 1

[5] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (srf): Learning view synthesis for sparse views of novel scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7911–7920, 2021. 2

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 4

[7] Charles Csuri, Ron Hackathorn, Richard Parent, Wayne Carlson, and Marc Howard. Towards an interactive high visual complexity animation system. *Acm Siggraph Computer Graphics*, 13(2):289–299, 1979. 2

[8] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 5, 7

[9] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5515–5524, 2016. 2

[10] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021. 2

[11] Jeffrey P Grossman and William J Dally. Point sample rendering. In *Eurographics Workshop on Rendering Techniques*, pages 181–192. Springer, 1998. 2

[12] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018. 2

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[14] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014. 6, 7

[15] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Transactions on Graphics (TOG)*, 35(6):1–10, 2016. 2

[16] Takeo Kanade, PJ Narayanan, and Peter W Rander. Virtualized reality: Concepts and early results. In *Proceedings IEEE Workshop on Representation of Visual Scenes (In Conjunction with ICCV'95)*, pages 69–76. IEEE, 1995. 1

[17] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 2

[18] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. *ACM Trans. Graph.*, 26(3):24–es, July 2007. 4

[19] Maria Kolos, Artem Sevastopolsky, and Victor Lempitsky. Transpr: Transparency ray-accumulating neural 3d scene point renderer. In *2020 International Conference on 3D Vision (3DV)*, pages 1167–1175. IEEE, 2020. 2, 3

[20] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 40(4), June 2021. 2

[21] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021. 2, 3, 4, 5, 6

[22] 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

[23] Marc Levoy and Turner Whitted. *The use of points as a display primitive*. Citeseer, 1985. 2

[24] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[25] 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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021. 2

[26] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In *Stereoscopic Displays and Virtual Reality Systems II*, volume 2409, pages 21–30. International Society for Optics and Photonics, 1995. 1

[27] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6878–6887, 2019. 2

[28] 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. In *European conference on computer vision*, pages 405–421. Springer, 2020. 2, 6, 7

[29] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 40(4), 2021. 2

[30] Francesco Pittaluga, Sanjeev J Koppal, Sing Bing Kang, and Sudipta N Sinha. Revealing scenes by inverting structure from motion reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 145–154, 2019. 2

[31] Eduard Ramon, Gil Triginer, Janna Escur, Albert Pumarola, Jaime Garcia, Xavier Giro-i Nieto, and Francesc Moreno-Noguer. H3d-net: Few-shot high-fidelity 3d head reconstruction. *arXiv preprint arXiv:2107.12512*, 2021. 6, 7

[32] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *European Conference on Computer Vision*, pages 623–640. Springer, 2020. 2

[33] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12216–12225, 2021. 2, 6, 7

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3, 5

[35] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[36] Johannes Lutz Schönberger, True Price, Torsten Sattler, Jan-Michael Frahm, and Marc Pollefeys. A vote-and-verify strategy for fast spatial verification in image retrieval. In *Asian Conference on Computer Vision (ACCV)*, 2016. 6

[37] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 1

[38] Artem Sevastopolsky, Savva Ignatiev, Gonzalo Ferrer, Evgeny Burnaev, and Victor Lempitsky. Relightable 3d head portraits from a smartphone video. *arXiv preprint arXiv:2012.09963*, 2020. 5

[39] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing 2000*, volume 4067, pages 2–13. International Society for Optics and Photonics, 2000. 2

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 5

[41] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, 2002. 3

[42] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021. 2

[43] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. Advances in neural rendering. *arXiv preprint arXiv:2111.05849*, 2021. 3

[44] 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, 4

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 4

[46] Fangjinhua Wang, Silvano Galliani, Christoph Vogel, Pablo Speciale, and Marc Pollefeys. Patchmatchnet: Learned multi-view patchmatch stereo, 2021. 6

[47] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021. 2, 6, 7

[48] Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018. 4

[49] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7467–7477, 2020. 3, 4, 5, 6

[50] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8534–8543, 2021. 2, 3, 4

[51] Minye Wu, Yuehao Wang, Qiang Hu, and Jingyi Yu. Multi-view neural human rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1682–1691, 2020. 2

[52] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 767–783, 2018. 5

[53] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020. 6

[54] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 2, 3

[55] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 2

[56] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4471–4480, 2019. 5

[57] Igor Zacharov, Rinat Arslanov, Maksim Gunin, Daniil Stefonishin, Andrey Bykov, Sergey Pavlov, Oleg Panarin, Anton Maliutin, Sergey Rykovanov, and Maxim Fedorov. "zhores"—petaflops supercomputer for data-driven modeling, machine learning and artificial intelligence installed in skolkovo institute of science and technology. *Open Engineering*, 9(1):512–520, 2019. 8

[58] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[59] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017. 2

[60] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv preprint arXiv:1805.09817*, 2018. 2