Eurographics Symposium on Rendering 2021
A. Bousseau and M. McGuire
(Guest Editors)

Volume 40 (2021), Number 4

Point-Based Neural Rendering with Per-View Optimization

Georgios Kopanas[1], Julien Philip[1,2], Thomas Leimkühler[1], and George Drettakis[1]

[1] Inria, Université Côté d'Azur
[2] Adobe

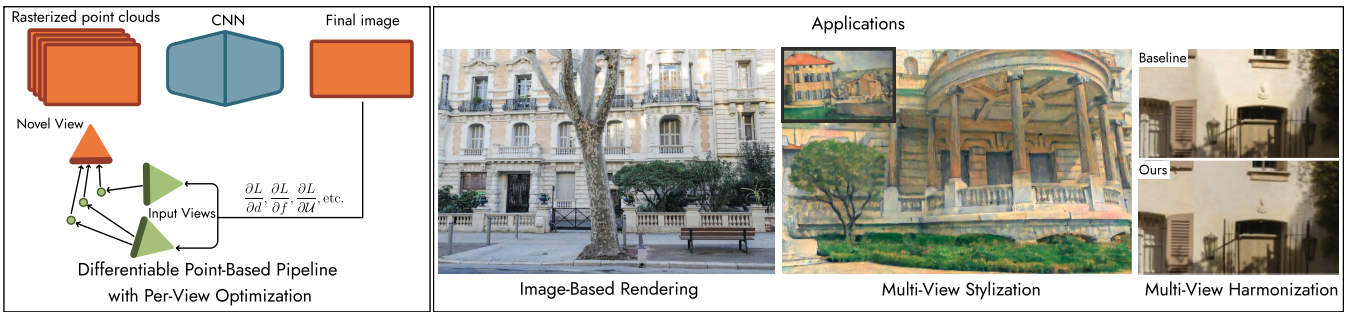


Figure 1: *Left: Our differentiable point-based pipeline allows optimization of attributes such as reprojected features or depth in each input view. Right: We illustrate three applications of our pipeline. From left to right: High-quality image-based rendering; multi-view stylization which can be used to achieve the illusion of navigating inside a painting with temporal and multi-view consistency; multi-view harmonization that can be used to improve image-based rendering (notice the removal of discontinuities due to exposure).*

Abstract

There has recently been great interest in neural rendering methods. Some approaches use 3D geometry reconstructed with Multi-View Stereo (MVS) but cannot recover from the errors of this process, while others directly learn a volumetric neural representation, but suffer from expensive training and inference. We introduce a general approach that is initialized with MVS, but allows further optimization of scene properties in the space of input views, including depth and reprojected features, resulting in improved novel-view synthesis. A key element of our approach is our new differentiable point-based pipeline, based on bi-directional Elliptical Weighted Average splatting, a probabilistic depth test and effective camera selection. We use these elements together in our neural renderer, that outperforms all previous methods both in quality and speed in almost all scenes we tested. Our pipeline can be applied to multi-view harmonization and stylization in addition to novel-view synthesis.

Keywords: Neural Rendering, Image-Based Rendering, Multi-View, Per-View Optimization

1. Introduction

Multi-view capture of real-world scenes has become a very popular way of creating digital content, e.g., as reconstructed 3D textured meshes or as input to Image-Based Rendering (IBR). To improve unreliably reconstructed depth, and thus image quality, traditional IBR algorithms use *per-input-view* information [CDSHD13, HPP*18]. Recently, neural rendering methods have been proposed to navigate in such captured scenes, some directly using traditional 3D reconstruction to guide rendering [RK20, HPP*18] – offering stability and robustness in novel view synthesis – while others use neural networks to learn a neural radiance field representation of the underlying geometry [MST*20, SDZ*21]. The former suffer from the fact that the rendering cannot recover from errors in the reconstructed geometry, while the latter have been shown to be less stable than methods that use explicit geometry [ZRSK20, RK20]. In the *wide-baseline, free-viewpoint* scenarios we target, such neural radiance field methods often present artifacts, partly because they do not optimize *per input view* parameters. We present a new neural rendering pipeline that is initialized with standard 3D reconstruction, but allows subsequent *per-input view* optimization of attributes such as reprojected latent features and depth, offering benefits of both approaches. Our per-view approach optimizes features and depth together, finding a good compromise between correcting depth and blur artifacts in novel-view synthesis.

Our differentiable multi-view pipeline is based on point-based splatting, together with an efficient, high-quality neural renderer, that optimizes *per-input-view* attributes. We first introduce a fast algorithm to select the cameras from which we actually reproject

points, significantly improving the speed/quality tradeoff. Our fully differentiable algorithm then performs soft, alpha-blended splatting of points reprojected from the input images into novel views, using bi-directional Elliptical Weighted Average (EWA) filtering.

When reprojecting images from input views with inaccurate geometry, it is always hard to determine which view contains the correct information. Point splats lack the connectivity of meshes, but also make it easier to formulate a probabilistic approach to this depth testing problem, which is necessary because of the "soft" point splatting approach. We introduce such an approach, that uses the distribution of depths from each reprojected input view to perform soft depth resolution.

Taken together, all the elements of this high-quality differentiable point-splatting framework allow us to introduce a temporally consistent neural renderer that we use to first optimize per-input-view parameters such as reprojected features or depth for a given scene. For each scene we can then use our neural renderer for several multi-view imaging tasks, such as free-viewpoint IBR, multi-view harmonization and multi-view consistent image stylization. In summary, our contributions are:

- A point-based multi-view imaging framework that allows per-view optimization. To do this, we introduce three new components: 1) a camera selection algorithm, for efficient rendering and training; 2) a differentiable point-splatting method with bi-directional EWA filtering and 3) a probabilistic per-view depth testing algorithm.
- A neural renderer based on our framework that allows optimization of per-view parameters and high-quality, temporally coherent rendering.

Our results show how our framework can be used for free-viewpoint IBR, notably with improved IBR quality for difficult cases such as vegetation or thin structures compared to previous work, multi-view color harmonization and multi-view consistent image stylization. Our neural renderer outperforms all previous view synthesis methods we tested both in quality and speed in almost all scenes presented, in terms of quantitative measures and visual quality.

2. Related Work

Our method is related to Image-Based Rendering (IBR), neural and point-based rendering, and in addition to IBR, we show applications of our approach to multi-view image harmonization and stylization. Each of these is a vast field on its own; we discuss only the most closely related representative work for each case.

2.1. Traditional IBR

Early methods in IBR [MB95, LH96] demonstrated the power of blending a set of images to allow various effects such as viewpoint changes or depth-of-field, without the need for manually building and rendering a full 3D scene. Using even approximate 3D was shown to be beneficial [GGSC96], and led to IBR methods that allow free-viewpoint navigation [BBM*01, HKP*99], by calculating *blend weights* to mix input images reprojected into a novel view using approximate geometry. The advent of powerful Structure from Motion (SfM) [SSS06] and Multi-View Stereo (MVS) [GSC*07] which allow the automatic reconstruction of an approximate 3D geometry *proxy*, made IBR a much more attractive solution for capture of multi-view datasets of real-world scenes, and subsequent free-viewpoint navigation with realistic rendering.

More recently, a new class of algorithms has been developed that exploit per-input-view – or simply *per-view* – data to improve rendering quality. These methods show that it is easier to accurately represent depth for a single view, even if that representation is not necessarily multi-view consistent. This can be achieved using a superpixel decomposition [CDSHD13] of each input view to preserve depth edges and synthesize missing depth, or per-view meshes to improve local estimation of depth [HRDB16]. Both methods greatly improve IBR quality in the presence of inaccurate depth. We also use per-view information, but in a *differentiable* pipeline, suitable for deep learning.

Specific IBR solutions have been developed for hard cases such as vegetation or thin structures [TDDD18], sometimes requiring manual intervention. We improve rendering quality for such cases compared to other automatic methods, without user input.

Camera selection is a critical component of many IBR and neural algorithms (see below). For large datasets, it is impossible to consider all input views for projection in a novel view being synthesized due to GPU memory limitations. Simple solutions include the use of Unstructured Lumigraph (ULR)-style weights [BBM*01] to choose a small number of cameras, e.g., for per-pixel ULR [BHE*20], or superpixel-based IBR [CDSHD13]. A voxel grid in which input view meshes are "sliced" [HRDB16] can also be used, providing an approximation valid up to the grid resolution. Riegler and Koltun [RK20] use reprojected depth maps to maximize the overlap with a target view. Our approach builds on the K-Maximum coverage algorithm [HP98] to provide fast and accurate camera selection.

2.2. Neural Rendering

Recent years have seen the use of deep learning to improve IBR, and as a completely new way to perform rendering overall, resulting in the rapidly expanding field of *neural rendering* [TFT*20].

One class of neural rendering solutions builds on MVS proxy geometry. Hedman et al. [HPP*18] refine per-view geometry and train a neural network to learn the *blend weights* for novel view synthesis, while Riegler and Koltun [RK20] use a recurrent architecture to project features using MVS geometry using per-view depth maps; in follow-up work [RK21] they use the MVS mesh to project features in a stable manner, resolving some of the issues of the original method. While we also use MVS geometry for reprojection, a fundamental limitation of these methods is that the fixed, often erroneous reconstructed geometry hinders image quality in novel view synthesis, while our approach optimizes per-view features and depth using backpropagation, improving the overall result (see Sec. 6.1 for comparisons).

Point-based rendering [GP11] has also been used recently for neural rendering [ASK*19, WGSJ20]. This representation is well studied in graphics and the advantages and disadvantages are well

understood [GP11]. Points come naturally from RGBD sensors, as well as SfM and early steps of many MVS algorithms; they also allow easy reprojection of learned features. In the work of Aliev et al. [ASK*19] point positions are assumed to be correct, and the neural renderer learns to correct visual artifacts. This process is successful if the artifacts are present in the training set. In contrast, Wiles et al. [WGSJ20] use points to leverage a differentiable renderer to compute gradients that in turn drive a model to estimate depth maps from a single image subsequently used for view synthesis. In the different context of geometric modelling, Yifan et al. [YSW*19] present forward EWA point splatting in a differentiable pipeline. We extend this approach to be bi-directional, allowing optimization of per-view features in the 2D space of the input images. While we use methodological components from these two methods, our multi-view datasets and per-view reprojection context is very different.

In concurrent work, Lassner and Zollhoeffer [LZ21] develop a fast differentiable point-based pipeline, but this method would need to be extended to allow alpha-blending and the bi-directional component of our pipeline to be used in our context.

Instead of using MVS, several methods start simply with the calibrated cameras from SfM and try and learn geometric representations. Initial methods started with very small numbers of views [CGT*19], plane sweep [FNPS16] or multi-plane images (MPIs) [FBD*19], with similarities in spirit to traditional optimization of per-view depth [PZ17]. Mildenhall et al. [MSOC*19] learn to reduce opacity of individual MPIs in uncertain areas, allowing several of them to be blended, thus supporting somewhat larger camera movement. These methods are limited to small-baseline capture such as light fields, in contrast to our wide-baseline scenario.

Learned volumetric representations have been recently proposed with very promising results either on video [LSS*19] or isolated objects [STH*19]. It is unclear how well these approaches would fare in our target sparse wide-baseline capture of large scenes. Similarly, recent video-based neural rendering solutions [ALG*20, BFO*20] introduce ideas with impressive results, but typically require much involved capture setups.

Learned neural radiance ("NeRF") representations (e.g., [MST*20, MBRS*21]) learn density and color in an outgoing direction. This is a very powerful representation, as testified by the follow-up work for various other applications (e.g., [PSB*20, SDZ*21, LGL*20]). NeRF methods present artifacts in our wide-baseline scenes, especially for vegetation (see Sec. 6). Even though recent, unpublished manuscripts (e.g., [YLT*21, HSM*21, RPLG21]) present interactive NeRF solutions for small rendering resolutions, they all involve some degradation in quality, which would possibly be more pronounced in our scenes.

2.3. Harmonization and Stylization

In addition to IBR, we show two other applications of our differentiable multi-view imaging approach: image stylization and image harmonization.

Image stylization is an active field in Computer Graphics, and many methods have been developed to transform a photograph to look like a painting or drawing [KCWI12]. Deep learning methods have been developed for this application with widespread success (e.g., [GEB16, IZZE17, UVL18] to name a few). Recently, solutions allow video stylization using deep learning [RDB18, TFK*20]; however they usually require reliable optical flow to work well, which is not available for our wide-baseline inputs. Image harmonization is useful in many different contexts e.g., panoramic image stitching, multi-view texturing etc. Unknown vignetting, exposure and camera response functions can be removed in individual images [KP08, Gol10], but adapting these solutions to multi-view can be challenging. Zhang et al. [ZCC16] address the challenge of recovering per-image exposure and radiance at each vertex to radiometrically calibrate the scene. Image harmonization can also be performed using color transfer using image statistics for single [HSGL11] or multiple images [HSGL13]. These methods tend to require more overlap between images than we have in our wide baseline datasets. Finally, Huang et al. [HDGN17] optimize a parametric curve per image of a multi-view dataset, and harmonize image intensities.

Image harmonization can be used to improve the quality of a textured mesh produced from an MVS reconstruction. Previous methods often optimize view selection and apply image editing operators to remove seams [ZK14, WMG14], although they also eliminate all view-dependent effects which is undesirable in IBR.

3. Differentiable Multi-View Rendering with Per-View Optimization

Our framework allows us to optimize *per-view attributes*, such as projected features, depth or others as required by different applications (see Sec. 5), in the context of neural rendering. This is in contrast to previous methods that focus only on optimization for the synthesis of a novel view or the global geometry.

To allow this, we need to select the cameras that will be reprojected; we first present and effective and robust camera selection algorithm. To allow optimization of attributes in each selected *input view*, we introduce a bi-directional point splatting approach from input views to a common novel view space, and back again to the input views so that the properties to be optimized are differentiable, while providing high-quality rendering using efficient filtered splatting. Finally, we introduce a probabilistic depth-test to resolve visibility between projected input views.

3.1. Camera Selection

Selecting a subset of cameras to use when synthesizing a novel view is an important component of many IBR and neural rendering algorithms (see Sec. 2.1, 6.1). Our goal is to develop a solution that is effective in choosing a good set of cameras by considering pixel coverage. Previous work [RK20] ranks and chooses input views based on how many novel-view pixels they cover. However, this approach does not necessarily lead to an optimal coverage of the entire novel view.

Our solution is inspired by the Maximum K-Coverage problem from set theory to approximate the solution [HP98]; in this problem

Figure 2: *With a limited budget of cameras, i.e., $N = 4$, we see the importance of camera selection even in simple scenes like Ponche. Left is the camera selection method of [RK20]. In the middle is ours and on the right we show a mask with black for missing content. In the top right (red), we see that the first method clearly fails to provide information for the whole frame, while ours (bottom, green) with the same number of cameras covers the frame.*

we are given several possibly overlapping sets and a number K. The goal is to select K of these sets so that the union of the selected sets has maximal size. In our case the input views are the sets, and their elements are (reprojected) visible pixels.

We first estimate a downscaled score map S_i in the novel-view space for each input camera. We use the proxy mesh estimated from the MVS algorithm as a basis for our re-projection, including a depth test for occlusions using this mesh. We use the mesh instead of point splatting for speed. Our algorithms chooses a subset of cameras that when combined maximizes the maximum score of the surface projected to the novel view. We do this by solving the following optimization problem:

$$\underset{V_{\text{sel}}}{\operatorname{argmax}} \sum_{p \in P} \max_{v \in V_{\text{sel}}} S_i(p) \text{ with } |V_{\text{sel}}| = k$$

where P are the pixels in the novel view, V_{sel} target set of views and the visibility score $S_i(p)$ is an indicator function returning 1 if p is visible in view i. This algorithm has a lower bound over the ratio of the score compared to the best solution $\frac{\text{greedy}}{\text{best}} < 1 - 1/e$ [HP98].

In the degenerate case where the novel view is the same as an input view, this algorithm is ambiguous. That is because the first input view will cover all the pixels in the set, hence all other views will have an equal score of zero. We can detect this case in its most general form when $\frac{V_{k+1}}{V_k} < \varepsilon$ where V_k is the total score after selecting the k-th candidate view. In this case we switch the candidate selection criterion and pick the view with the absolute maximum $\sum_{p \in P} S_i(p)$.

The visibility function can be replaced trivially to incorporate a per-pixel score that is non-binary. We choose the ratio between the distance of the point visible from the input camera and the novel view. This is a fast approximation of the density of the points in the novel view. Given M input cameras the complexity of the algorithm is $O(N \cdot M)$ which is tractable for interactive rendering.

We compare our camera selection in Fig. 2 to that of Riegler et al. [RK20]; this approach misses important content on the lower right corner (shown in black on the red box – upper right).

3.2. Bi-Directional Differentiable Point Cloud Rasterization

As discussed in Sec. 2, using point-based rendering allows us to directly benefit from the geometry provided by MVS/SfM and from the advantages of a fully differentiable pipeline [ASK*19, WGSJ20, YSW*19]. We want to allow optimization of *per-view* attributes, such as latent features, depth etc. in each *input image*.

Traditional rasterization suffers from discontinuities during splatting and z-buffering. Overcoming this and incorporating rasterization techniques in a differentiable pipeline requires a soft splatting approach of input view pixels into the novel view; we adapt Elliptical Weighted Averaging (EWA) [Hec89, RPZ02, YSW*19] for this task. EWA has been used for the projection of 3D points to 2D images where Ren et al. [RPZ02] (eq. 9-12) provide the required procedure to compute the Jacobian of the transformation from a 3D point to a regular 2D grid. In our approach, we need to account for the combined stretch induced by two transformations: First the lifting of a sample from a regular 2D grid to 3D, and then the projection of this point back to the novel view. This naturally gives rise to a bi-directional procedure, where we first use the inverse Jacobian of the transformation from the input view to 3D, followed by the forward Jacobian to move from 3D to the novel view. Since the Jacobian of an inverse transformation is the inverse of the Jacobian, we can use the method of Ren et al. in both cases and just invert the first Jacobian (Fig. 3). The net result of this process is an anisotropic 2D Gaussian $\mathbf{G}$ associated with each pixel to reproject (see [YSW*19] for details). This bi-directional approach allows us to extend the differentiable point-splatting method for per-view optimization.

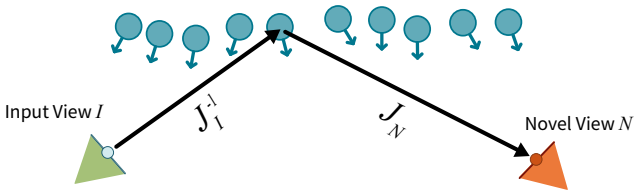


Figure 3: *Points (cyan, arrows denote normals) are lifted from the 2D input image (green), and then reprojected to the novel view (orange). This requires the use of a* bi-directional *EWA filtering method, employing two Jacobians.*

Alpha Blending. We use alpha-blending to avoid discontinuities at point boundaries and when points overlap [WGSJ20]. We find the opacity $\alpha_{n,i}^{(v)}$ of pixel i in input-view n by evaluating $\mathbf{G}$ for novel-view pixel v. Specifically, alpha blending is performed using front-to-back compositing:

$$c_n^{(v)} = \sum_{i \in \mathcal{N}_n^{(v)}} \left(c_{n,i} \alpha_{n,i}^{(v)} \prod_{j=1}^{i-1} (1 - \alpha_{n,j}^{(v)}) \right) \tag{1}$$

where c is a pixel attribute (e.g., color) and $\mathcal{N}_n^{(v)}$ the depth-ordered set of splats from input view n that overlap novel-view pixel v. We will omit the superscript v in the remainder of this section to aid readability. The alpha-blending step requires a specific gradient computation, extending Yifan et al. [YSW*19]:

$$\frac{\partial c_n}{\partial c_{n,i}} = \alpha_{n,i} \prod_{j=1}^{i-1} (1 - \alpha_{n,j})$$

$$\frac{\partial c_n}{\partial \alpha_{n,i}} = c_{n,i} \prod_{j=1}^{i-1}(1-\alpha_{n,j}) - \sum_{l=k+1}^{|\mathcal{N}_n|} c_{n,l}\alpha_{n,l} \prod_{\substack{j=1 \\ j\neq i}}^{l-1}(1-\alpha_{n,j})$$

For computational efficiency, we limit the support of **G** based on two criteria: First, we stop considering more input pixels if the accumulated alpha reaches one grayscale level, since in this case any subsequent points will not alter the color. Second, we consider the spatial decay of **G**, by computing the eigenvalues of its covariance matrix. We remove outliers (top 3% of points with the highest variance), and choose the highest remaining variance $\sigma_{\max}$. We define the cut-off radius r to be 99% of the energy of the corresponding Gaussian: $r = \lceil 3\sigma_{\max} \rceil$, i.e., the standard "three σ" cutoff. In general, we only maintain the front-most $k_d = 150$ splat contributions per novel-view pixel to keep a constant memory footprint.

The EWA filter process correctly accounts for distance and orientation during our bi-directional splatting. However, there is inherent *uncertainty* in the position of each input-view pixel arising from the MVS reconstruction. We model this uncertainty $\mathcal{U}$ as a multiplicative factor on the covariance matrix of **G**, effectively controlling the fuzziness of the individual splats. Thanks to our differentiable framework, we can include these per-input view pixel uncertainties to the set of optimizable attributes. We found this to be a simple yet expressive way to let the pipeline handle and balance geometric uncertainty arising from reprojection errors. We initialize uncertainty with $\mathcal{U} = 0.5$; we initially used a value of one (corresponding to standard EWA), but our tighter kernel improves image sharpness. During optimization (Sec. 4.2) uncertainty is refined and adapts to reconstruction and rendering errors.

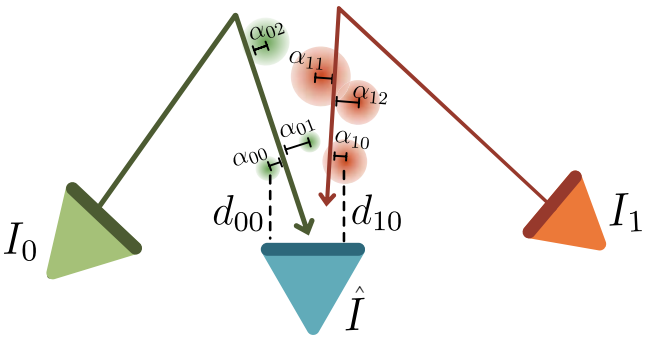


Figure 4: *Soft rasterization incorporates uncertainty, allowing this information to propagate to the depth test. This makes all decisions soft, which is required due to the noisy data from MVS reconstruction; hard visibility could discard valuable information. We see how points that do not directly intersect a ray convey depth information to the novel view.*

3.3. Probabilistic Depth Testing for Point-Based Rendering

A given pixel in the novel view will receive splats of continuous opacity from points coming from different input images (Fig. 4). Determining which view is in front of all others is a complex task, further exacerbated by the uncertainty in the depth values of the different points. Many ad hoc solutions have been proposed to this problem of resolving uncertain depth in IBR [CDSHD13, HRDB16, PLRD21]. Hedman et al. [HPP*18] rely on the learned blending step to make the correct decision, which is not always reliable. Soft Z-buffering methods (e.g., [TTS18]) are not directly applicable since they treat visibility during rasterization; we need to maintain separate information for each view, and our depth test determines which input view to use *after* rasterization.

Initial experiments showed that our neural renderer struggles to learn the depth test without any inductive bias. To introduce the depth-test explicitly, we define a distribution of depths d_n for a given input view n. The density function of this distribution is given by the splats reprojected with soft rasterization; we can thus opt for a probabilistic approach to this depth test that amounts to determining whether the points reprojected from n are more likely to be in front of those projected from all other views.

We define a random variable D_n representing the depths projected to a novel-view pixel. Our goal is thus to determine probability $P(D_n < \min_{m\neq n}(D_m))$ i.e., that D_n is closer than all other D_m.

We derive a solution to this problem based on a mixture model. Intuitively, we aim to compute the probability that the re-projection of a pixel for input view n on a pixel of the novel view is closer than all re-projections coming from the other input views.

We first rewrite this as a product of probabilities that compares the depth distributions coming from two views. Using a mixture model further allows us to compare each component of the two mixtures, leading to a quadratic computation with respect to the number of points re-projected on the pixel. We choose a triangle distribution that allows for softness in the depth test and accounts for depth uncertainty, while having a finite support. We choose this distribution over Gaussians, since it simplifies and speeds up the quadratic computations. The detailed derivation is presented in the supplemental; we state the final expression here:

$$P(D_n < \min_{m\neq n}(D_m)) \approx$$
$$\frac{2\sigma}{S} \sum_{i\in\mathcal{N}_n} \beta_{n,i} \sum_{t=1}^{S} \prod_{m\neq n} \left(\sum_{j\in\mathcal{N}_m} \beta_{m,j} T(s(t), d_{m,j}, \sigma) \right) f_i(s(t), d_{n,i}), \tag{2}$$

with T the integral of the symmetrical triangular distribution f with support 2σ, $d_{m,j}$ the depth of point j in view m and

$$\beta_{n,i} = \alpha_{n,i} \prod_{j=1}^{i-1}(1-\alpha_{n,j}).$$

Further, S is the number of samples and $s(t) = d_{n,i} - \sigma + \frac{t}{S+1}$. We found that setting $S = 1$ provided satisfactory results in all our experiments. This computation is performed in parallel per pixel using CUDA.

4. Temporally Consistent Neural Rendering and Optimization

Our bi-directional point splatting approach allows us to define a powerful neural renderer that we will use both for multi-view imaging and rendering as well as for per-view attribute optimization.

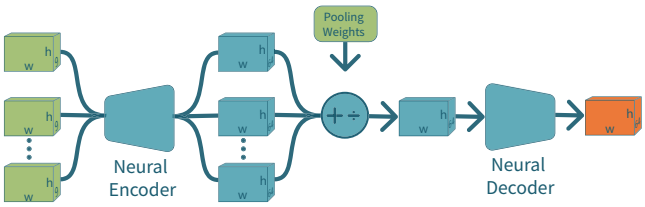


Figure 5: *Our neural network architecture takes as input the rasterized point clouds and feeds each one individually in an encoder network. The encoded rasterizations are then pooled by a set of weights that introduce inductive bias for the visibility but also allow for temporal smoothing in an interactive scenario. The pooled features are fed through a decoder to generate the final rendering.*

4.1. Point-Based Neural Renderer

For each view we generate a set of attribute layers, namely the input-view colors and a set of optimized latent features reprojected to the novel view using our point splatting method. The architecture of our pipeline is visualized in Fig. 5.

The first part of the network encodes each view in a feature space of $\mathcal{F}$ dimensions for each input view with a series of convolutional residual blocks, with shared weights across all input views. Then we apply a weighted average pooling [ZKR*17] and produce a single $\mathcal{F} = 64$ channel image, which is subsequently decoded by consecutive convolutional residual blocks. The weights for each input image consist of three terms, as detailed below.

First, camera selection (Sec. 3.1) can cause temporal instability when cameras appear/disappear for a selected novel view. We deal with this problem with a smooth fading strategy using temporal filtering. Our camera selection algorithm returns a set of selected cameras that get a score of $s = 1.0$ while the rest get $s = 0.0$. We temporally filter the score s_i^t of view i in frame t as follows:

$$w_i^t = \lambda s_i^t + (1.0 - \lambda) w_i^{t-1}$$

where λ controls how fast the temporal filtering adapts to changes; we set $\lambda = 0.05$. Once we update the weights, we select the N highest to use for rendering. Unless stated otherwise, we used $N = 9$ in all our experiments. To avoid popping we further refine the weights across all views, similar to the approach used by [SXZ*20], to obtain final smooth camera selection weights w_{CS}:

$$w_{CS} = \frac{\mathbf{w} - \min_i w_i}{\sum(\mathbf{w} - \min_i(w_i))}$$

ensuring that views which are coming in and out of the set have zero weight, and $\min_i$ is the element-wise min of a vector, where $\mathbf{w}$ is the vector of all w_i and the equation uses element-wise difference. This moves the C_0 discontinuity to C_1, and thus every time the selected set changes, the gradients of the weights become discontinuous instead of the actual values of the weights.

Second, we observe that weighted average pooling can lead to artifacts in the presence of view-dependent effects. To compensate for this, we use weights based on input-view texture stretch: For each point reprojected in the novel view we quantify the amount of stretch as the ratio between the two eigen-values of the covariance of the splatting kernel $\mathbf{G}$ (Sec. 3.2) [KCS14]. This weight w_{TS} penalizes pixels at grazing angles, increasing the influences of more front-facing views.

Third, we employ probabilistic depth weights $w_{\text{PD}} = P(D_n < \min_{m \neq}(D_m))$ (Sec. 3.3, Eq. 2) which account for point visibility.

The final weights w are the product of the three weights described above:

$$w = w_{\text{CS}} \cdot w_{\text{TS}} \cdot w_{\text{PD}}$$

Compared to previous neural renderers, this architecture shows improved temporal stability. Importantly, if we decide to change the number of views we select, we do not need to retrain. This is because the weighted average pooling normalizes the sum of the weights of all views per pixel, hence no matter how many views are added the distribution of the magnitude of the features will stay the same for the decoder.

4.2. Optimization

We optimize the scene representation and our neural renderer jointly. The list of attributes we optimize in each input view reads as: color, depth, normal, uncertainty, and a 6-component latent feature vector. The feature vector extends the input-view color channels and allows our pipeline to encode useful additional information per input-view pixel. The decoder uses the optimized features to improve rendering. We initialize the six features to 0.5, and depth is initialized with the generated per-view meshes from Hedman et al. [HPP*18]. The neural renderer is initialized using the method of Zhang et al. [ZDM19].

We perform optimization with a leave-one-out strategy, i.e., at each iteration we randomly hold out one view to use as ground truth. We select the 9 views (chosen randomly from the best 13), which empirically gave the best compromise in our test scenes; please see Sec. 6.2 for an ablation study on the number of views selected. We lift all the pixels to 3D using the depth maps of their corresponding input views so we can splat them to the view we want to render. We stack all views in batches and feed them through the neural renderer. We use an L_1 loss

$$L = \|I - I_{gt}\|_1,$$

while certain applications require additional loss terms as described in Sec. 5. We use an ADAM optimizer for all our parameters and we renormalize normal vectors in each iteration to avoid numerical instabilities. We also feed the features through a sigmoid to scale them to unit range so they are in the same range as colors, before feeding them to the next module. The learning rates we use are the following: $lr_{\text{CNN}} = lr_{\text{normal}} = lr_{\text{depth}} = 0.0001$, $lr_{\text{features}} = 0.001$, $lr_{\mathcal{U}} = 0.01$. Unless stated otherwise, we trained all scenes for 100,000 iterations in 150x150 patches, at which point the loss no longer improved.

4.3. Forward Rendering

Once optimized for a given scene, our neural network can be used for high-quality free-viewpoint rendering. For each novel view, we choose the input views using our camera selection approach, then

Figure 6: *Comparison of our full soft rasterization approach with the interactive version. Differences are subtle and do not significantly degrade visual image quality.*

project the points from these views using our splatting approach, perform the probabilistic depth test and then run our neural network to synthesize the image.

A major bottleneck of the forward pass is the point-splatting. During training, we need to use the full approach including the gradient computations in the backward pass. However, for fast rendering at runtime, we have implemented an efficient OpenGL-based approximation of the point reprojection in Sec. 3.2. We use $k_d = 10$ global depth layers, where the relevant depth interval is determined by rasterizing the MVS mesh depth at low (1/16) resolution followed by a min-max mipmap construction. We then splat all pixels of all input views in parallel into the depth layers, where the multiplicative accumulation of opacity (Eq. 1) utilizes hardware accelerated additive blending of $\log(1-\alpha)$. We composite the depth layers back-to-front in parallel over all re-projected pixels of all input images. We then apply the probabilistic depth test (Sec. 3.3) that now only needs to consider at most k_d alpha-weighted depth samples. Quality degrades only marginally using this approximation, allowing interactive viewing; please see Fig. 6 and the supplemental for further visual comparisons and statistics. Results are computed at full solution unless otherwise stated. Our unoptimized implementation runs at 4.5 fps, where camera selection takes 87 ms, point splatting 36 ms, the depth test 38 ms, and network evaluation 59 ms. While we achieve interactive frame-rates, this comes at the cost of memory consumption. First, the depth layers take approximately 4 GB of memory for a resolution of 900×600 with 10 layers and 9 views. This could be compressed through standard texture compression methods. Second, the fact that we batch together all the input views and feed them through our neural renderer means that memory consumption is high for this stage as well; pytorch reported 4.5 GB of memory usage for the above configuration.

5. Applications and Results

We illustrate our differentiable multi-view pipeline on three applications. The first is IBR, the second multi-view harmonization and the third multi-view style transfer.

We implemented our system in PyTorch, but wrote custom CUDA kernels for the point splatting algorithm used in training and accurate rendering, since available implementations (e.g., PyTorch3D) are an order of magnitude slower and we have integrated our network into a C++ framework with OpenGL for display; our interactive version implements point splatting in OpenGL. We will release all code and data, please see https://repo-sam.inria.fr/fungraph/differentiable-multi-view, which also includes the supplemental website.

Figure 7: *Novel views synthesized with our method for the scenes (top to bottom): Museum, Ponche, Hugo and Stairs.*

5.1. Image-Based Rendering

Our IBR pipeline uses our neural renderer directly, and allows free-viewpoint navigation. We show example images in Fig. 7; three different scenes, taken from the Deep Blending datasets [†] and the "Stairs" scene from [TDDD18]. We also show the Truck scene from Tanks and Temples [KPZK17] in Fig. 10. Our method achieves sharp results in regions with vegetation and can recover from some of the reconstruction artifacts due to thin structures. We

[†] See http://www-sop.inria.fr/reves/publis/2018/HPPFDB18/datasets.html.

Figure 8: *Multi-view capture can suffer from differences in exposure and other camera parameters between views; our algorithm optimizes for a brightness coefficient to achieve harmonization e.g., in the second and fourth image. Top row: the images adjusted by our optimization. Bottom row: original images.*

encourage the reader to watch the supplemental videos to appreciate the visual quality of our method.

All timings are reported on an RTX6000 GPU for display and RTX8000 for training. Training for 100K iterations takes 12-14h. For small scenes (Hugo and Tree) only 20K iterations are required (approximately 3h).

5.2. Multi-view Harmonization

We demonstrate a multi-view harmonization technique that works well for one of the most common problems in real-world multi-view captured content: Exposure and other camera parameters can fluctuate between images, creating multi-view inconsistencies breaking basic IBR algorithm assumptions. We model this as an additional coefficient μ_i per view that is multiplied with the color of the corresponding input image. We initialize $\mu_i = 1.0$ and we allow μ_i to be optimized like any other per-view parameter. This parameter modifies the input images that are both used for ground truth and for re-projections. It is thus necessary to avoid degenerate solutions during the optimization i.e., $\mu_i = 0$ for all views. We address this problem by adding a regularization term in the loss function:

$$L_\mu = \lambda \frac{\sum_i^N (\mu_i - 1.0)^2}{N}, \tag{3}$$

where N is the number of input views and $\lambda = 0.2$ is a hyper-parameter that controls the weight of the reguralization. We also introduce a photoconsistency loss between the optimized image I_n and the other reprojected images $R(I_m)$, where M is a mask of the pixels containing content in $R(I_m)$.

$$L_{\text{photo}} = \sum_{m \neq n} \left(MI_n - R(I_m)\right)^2 \tag{4}$$

We illustrate the effect of harmonization on 5 images taken from the Ponche scene in Fig. 8, and visualize the optimization of individual μ_i in the supplemental.

5.3. Multi-view Style-Transfer

For multi-view style-transfer we leverage the differentiability of our point-based reprojection to jointly optimize each input image to match a given style, while maintaining photo-consistency. We base our stylization method on the approach of Mechrez et al. [MTZM18]. We first make the input image colors parameters of the optimization process. Then we add the following multi-view style-transfer loss to the optimization:

$$L_{\text{mvst}}(I_n, R(I_{m\neq n})) = L_{\text{photo}} + CX_{\text{style}}(I_n, S) + CX_{\text{cont}}(I_n, I_n^*) \tag{5}$$

The first term ensures photo-consistency. The second and third terms allow style transfer between I_n and a style image S, while maintaining content between I_n and its original version I_n^* and use the contextual loss $CX()$ for style transfer as described in Mechrez et al. [MTZM18]. We show the effect of multi-view style-transfer on 5 images taken from the Museum datasets in Fig. 9. Additional results and comparisons are presented in the supplemental and video.

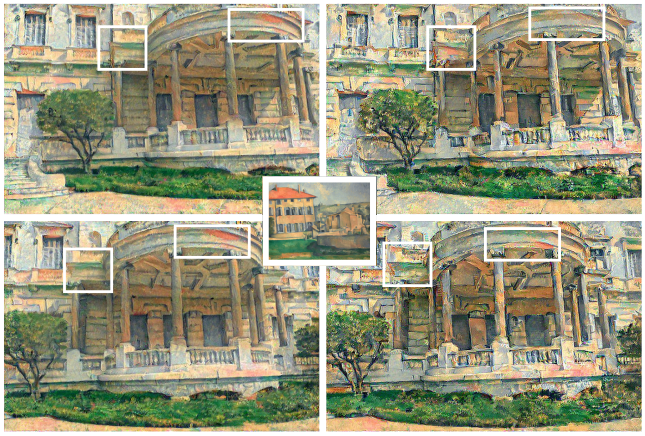
6. Evaluation

We compare our IBR method with previous IBR and recent neural rendering methods; the most meaningful comparisons are qualitative visual inspections of the videos provided in the supplemental web pages. We also provide quantitative comparisons and an ablation study analyzing the effect of each component of our method.

6.1. IBR and Neural Rendering

We compare our algorithm with two baselines: a mesh textured directly from multi-view stereo and a per-pixel ULR algorithm [BHE*20]. The textured mesh was generated using RealityCapture [Rea18] for all scenes from the DeepBlending database and using a direct texturing method by blending using ULR weights for Stairs and Salon [BHE*20]; the figure for this comparison is presented in the supplemental.

Our main comparisons are with state-of-the-art neural renderers: Deep Blending (DB) [HPP*18], Free View Synthesis (FVS) [RK20], Stable View Synthesis (SVS) [RK21] and



(a) Our photoconsistent style transfer (b) Baseline using [MTZM18]

Figure 9: *Two different input views for each stylization method with the painting shown in the middle. Left: our photoconsistent algorithm. Right: results if we apply the method of Mechrez et al. [MTZM18]. Insets in white show multi-view inconsistencies in the baseline.*

NeRF++ [ZRSK20]. Standard NeRF implementations were unable to treat our scenes, for reasons explained by Zhang et al. [ZRSK20]. The comparisons are best appreciated in our supplemental web page where videos can be viewed side-by-side.

Overall, our method has better overall visual quality for all scenes and all methods; only NeRF++ has comparable quality in some cases, and performs better for the thin structures in the Stairs scene. However, Fig. 10 shows the clear benefit of our per-view optimization in scenes with vegetation: our method produces much sharper details in such regions compared to all previous solutions (Hugo, Tree, and Street scenes). Our method also removes some of the artifacts in thin structures (rails in Stairs) and can recover them in much sharper detail when they are not reconstructed perfectly (chair and table in the Ponche scene), due to the combined effect of latent features and depth optimization. NeRF++ performs slightly better visually for Stairs, removing all over-reconstruction artifacts.

We perform a quantitative leave-one-out comparison; the results are shown in Table 1 and Fig. 16. We used the authors' implementation of each method, using COLMAP SfM and the RealityCapture MVS mesh (except for Stairs where Colmap MVS is used); the SVS [RK21] code ran out of GPU memory for this test. All views are used for SfM and MVS, and for training for NeRF++ and our method, but we leave out the target view for rendering for each method except NeRF++ where this is not possible. NeRF++ trained for 48h on average for each scene. NeRF++ thus has an "advantage" over other methods, since the view being rendered is not actually left out. We present results for three error metrics, PSNR, LPIPS and DSSIM. Leave-one-out quantitative comparisons are not particularly informative for IBR: The ranking of methods changes according to the metric, and can be influenced by training (e.g., FVS uses LPIPS in the training loss, and is thus better for this metric, but not for others). The fact that ULR is second best in many cases shows that the metrics do not correspond well with visual perception since ULR shows many visual artifacts compared to the others. Nonetheless, our method has the best scores for all metrics and all scenes.

6.2. Ablations

We investigate the effect of each of our components through ablation studies in per-scene training. In particular we optimize our network in three different configurations: 1) our full pipeline, 2) without the depth test, 3) without the 6-feature latent vector optimization, 4) without normal/depth/uncertainty optimization and 5) only optimizing the neural renderer. We run the ablation test for 50k iterations. The results on paths of a subset of our test scenes can be seen in the supplemental webpage. All test sets comprise three views that were held-out during optimization.

We first show the importance of the depth test in Fig. 11. If we disable the test, the soft rasterization results in incorrect "transparent" regions. The reprojected features play a central role in the quality of our results, especially for vegetation. This can be seen in Fig. 12.

In Fig. 13 we examine the relative importance of features and normal/depth/uncertainty in the quality of our results. In particular, when depth/normals/uncertainty are not used we notice that the some reconstruction artifacts are more visible in the region marked with a red box Fig. 13(b). The optimization cannot recover the background information from the over-reconstructed point splats without resizing and moving them. When this freedom is given, the optimization spreads the points so background information is visible. The neural renderer then filters out the over-reconstruction. Reprojected features also result in sharper rendering; see Fig. 13(c) green box for the railings.

Overall, we see from this study that each component plays a different role, and handles different artifacts. Our method does not "correct" depth *per se*; the strength of our method is making these components (projected features, depth etc.) be optimized together to obtain the best overall compromise between correcting geometry reconstruction errors and obtaining sharp rendering in all regions of the synthesized novel view.

We investigate the effect of network size shown in Fig. 15, by reducing the number of feature and of residual blocks. We see that even with a much smaller model than that used in most of our experiments presented, the results are still of very high quality. We also investigate the effect of varying the number of input cameras. Here, the major limitation is memory. We explore a trade-off between simplifying the neural renderer as described above and the number of cameras in memory during inference. We test in an indoor scene ("Salon") that is challenging for camera selection. The inside-out nature means that we need to increase the number of cameras to achieve good results. The versatility of our neural rendering allows us to explore the number of cameras without the need to retrain the network. In the supplemental we show that decreasing the size of the model often does not affect the results substantially and a higher number of cameras used for rendering in the same time/memory budget compensates for the smaller neural renderer. In Fig. 14 we show how in the Museum scene quality degrades as we lower the number of cameras, although it is possible to lower

Figure 10: *Novel views that do not exist in the input dataset. Left to right: Stable-View Synthesis, NERF++, Deep Blending, Free-view Synthesis and our method.*

the number of cameras with minimal degradation in the quality depending on the scene.

6.3. Limitations

While our method outperforms previous solutions in the vast majority of examples we show, it is not without limitations. In some scenes if the model overfits there is some slightly visible temporal flickering; This is visible for the Hugo scene for 40K iterations (please see supplemental website). However, for a smaller number of iterations (20K iterations), the overall result of rendering is comparable, and the flickering is reduced significantly.

We currently optimize normals and depth separately; it might be beneficial to enforce a consistency loss between them, which could improve geometry optimization overall. Our treatment of view-dependent effects is improved using the texture stretch weight w_{TS} (Sec. 4.1). However, to correctly render view-dependent effects, reflection flow needs to be modeled ([RPHD20]); NeRF achieves this to a certain extent in the depth/appearance optimization, but at

Table 1: ***Leave-one-out view-synthesis quantitative evaluation on real test scenes.***

Metric	SSIM ↑					PSNR ↑					LPIPS ↓				
Method	ULR	DB	FVS	N++	Ours	ULR	DB	FVS	N++	Ours	ULR	DB	FVS	N++	Ours
Street	0.67	0.69	0.64	0.77	0.93	17.84	19.91	17.80	25.23	28.67	0.245	0.27	0.249	0.25	0.06
Hugo	0.77	0.74	0.78	0.75	0.92	21.7	22.9	22.6	25.2	27.25	0.164	0.23	0.162	0.29	0.07
Tree	0.756	0.76	0.754	0.78	0.94	20.7	22.5	22.1	27.3	30.4	0.18	0.22	0.17	0.25	0.05
Ponche	0.78	0.7701	0.72	0.773	0.92	23.5	23.7	21.5	27.5	29.4	0.21	0.27	0.24	0.306	0.09
Museum	0.7501	0.76	0.758	0.78	0.95	20.9	23.02	22.1	25.7	30.6	0.19	0.23	0.16	0.24	0.04
Stairs	0.807	0.821	0.79	0.83	0.94	20.13	22.04	21.33	28.88	30.79	0.16	0.19	0.17	0.22	0.05
Truck	0.76	0.64	0.78	0.703	0.88	21.4	18.4	20.97	23.08	25.3	0.18	0.27	0.179	0.36	0.11

Figure 11: *The probabilistic depth test successfully resolves visibility between different viewpoints as demonstrated in the Museum scene by eliminating transparent regions. Top row: with the depth test. Bottom row: without.*

(a) Full Method (b) No Features

Figure 12: *Ablation for per-view latent features, which play a very important role in preserving the sharpness of vegetation.*

a significant cost in training and rendering. Correctly solving this problem is an exciting future challenge.

Even though we improve over methods that cannot recover from MVS errors, if the initial reconstruction is too erroneous, our optimization cannot always recover and fails similarly to other methods. This includes cases, when a large piece of geometry is missing (Fig. 17), but also more complex cases such as reflections and transparency that require multiple depths. Complete failure of MVS reconstruction can also lead to temporal instability for our method e.g., "Tree" scene in the supplemental website. These are all interesting avenues for future work; we believe our framework provides the initial tools for the development of such solutions.

7. Conclusion

We have presented a new differentiable point-based pipeline that enables per-input-view optimization. To achieve the rendering quality required, we introduce a differentiable splatting pipeline allowing per-view optimization, with probabilistic depth testing between input views and efficient camera selection. These components allow us to define a temporally consistent neural renderer.

This powerful pipeline allows optimization of different properties in the input views of a multi-view dataset. We present three examples: first IBR, where we optimize depth, color and reprojected features, clearly improving neural rendering quality in regions such as vegetation; second multi-view harmonization, where we are able to improve exposure related inconsistencies in multi-view datasets and finally multi-view stylization, where we achieve multi-view consistency in stylization.

Our results show that we provide a solution to our initial goal, i.e., to be able to benefit both from the rich, stable 3D information provided by multi-view stereo, while allowing neural optimization in the input views which improves novel-view synthesis. We believe that our generic pipeline can be applied to further improve neural rendering in the future. In particular, we are interested in investigating ways to recover from very large reconstruction errors, which, based on previous work [CDSHD13, HRDB16, HPP*18] should potentially be easier in the space of input views. One long standing problem of IBR is the treatment of reflections and transparency [SKG*12, RPHD20]. Our framework could be extended to handle the two depths required to treat these problems in a principled manner. Another application area could be multi-view matting.

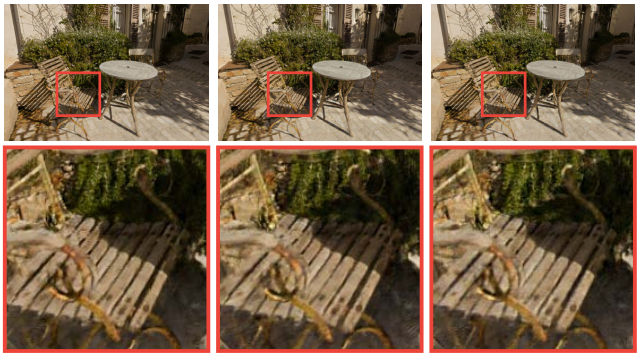
8. Acknowledgments

This research was funded by the ERC Advanced grant FUNGRAPH No 788065 (http://fungraph.inria.fr). The authors are grateful to the OPAL infrastructure from Université Côte d'Azur

(a) Full (b) No depth/normal/uncertainty (c) No features (d) Only neural rendering

Figure 13: *Each component of our method helps correct different artifacts. E.g., without features the stairs behind the bars are blurry, without depth/normal/uncertainty the overreconstruction becomes worse. Taken all the components together results in the overall best rendering.*

Figure 14: *Left to right: N=4, 6 and our chosen value of N=9. For the first row $N = 4$ is sufficient, but for some scenes (second row) we need 9 views.*



(a) 64 features 16 res. blocks (a) 32 features 8 res. blocks (a) 16 features 8 res. blocks

Figure 15: *We study the effect of network model size. As we can see, even with a much smaller model than that used in most of our experiments presented, the results are still of high quality.*

for providing resources and support. The authors thank G. Riegler for help with comparisons. Thanks to A. Bousseau for proofreading earlier drafts, and E. Yu for help with the figures. Finally, the authors thank the anonymous reviewers for their valuable feedback.

References

[ALG*20] ATTAL B., LING S., GOKASLAN A., RICHARDT C., TOMPKIN J.: MatryODShka: Real-time 6DoF video view synthesis using multi-sphere images. In *European Conference on Computer Vision (ECCV)* (Aug. 2020). 3

[ASK*19] ALIEV K.-A., SEVASTOPOLSKY A., KOLOS M., ULYANOV D., LEMPITSKY V.: Neural point-based graphics. *arXiv preprint arXiv:1906.08240* (2019). 2, 3, 4

[BBM*01] BUEHLER C., BOSSE M., MCMILLAN L., GORTLER S., COHEN M.: Unstructured lumigraph rendering. In *Proc. SIGGRAPH* (2001). 2

[BFO*20] BROXTON M., FLYNN J., OVERBECK R., ERICKSON D., HEDMAN P., DUVALL M., DOURGARIAN J., BUSCH J., WHALEN M., DEBEVEC P.: Immersive light field video with a layered mesh representation. 3

[BHE*20] BONOPERA S., HEDMAN P., ESNAULT J., PRAKASH S., RODRIGUEZ S., THONAT T., BENADEL M., CHAURASIA G., PHILIP J., DRETTAKIS G.: sibr: A system for image based rendering, 2020. URL: https://sibr.gitlabpages.inria.fr/. 2, 9

[CDSHD13] CHAURASIA G., DUCHENE S., SORKINE-HORNUNG O., DRETTAKIS G.: Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. on Graphics (TOG) 32*, 3 (2013). 1, 2, 5, 12

[CGT*19] CHOI I., GALLO O., TROCCOLI A., KIM M. H., KAUTZ J.: Extreme view synthesis. In *ICCV* (2019). 3

[FBD*19] FLYNN J., BROXTON M., DEBEVEC P., DUVALL M., FYFFE G., OVERBECK R., SNAVELY N., TUCKER R.: Deepview: View synthesis with learned gradient descent. In *CVPR* (2019). 3

[FNPS16] FLYNN J., NEULANDER I., PHILBIN J., SNAVELY N.: Deepstereo: Learning to predict new views from the world's imagery. In *CVPR* (2016). 3

[GEB16] GATYS L. A., ECKER A. S., BETHGE M.: Image style transfer using convolutional neural networks. In *CVPR* (2016). 3

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *Proc. SIGGRAPH* (1996). 2

[Gol10] GOLDMAN D. B.: Vignette and exposure calibration and compensation. *IEEE transactions on pattern analysis and machine intelligence 32*, 12 (2010). 3

Figure 16: *Renderings of left-out input views. Left to right: Free-View Synthesis, NERF++, Deep Blending, Ours and Ground Truth. Our method renders input views with almost no artifacts.*

Figure 17: *The car roof is not reconstructed; our method cannot completely recover. Previous methods have similar difficulties.*

[GP11] GROSS M., PFISTER H.: *Point-based graphics*. Elsevier, 2011. 2, 3

[GSC*07] GOESELE M., SNAVELY N., CURLESS B., HOPPE H., SEITZ S. M.: Multi-view stereo for community photo collections. In *ICCV* (2007). 2

[HDGN17] HUANG J., DAI A., GUIBAS L. J., NIESSNER M.: 3dlite: towards commodity 3d scanning for content creation. *ACM Trans. Graph. 36*, 6 (2017). 3

[Hec89] HECKBERT P. S.: Fundamentals of texture mapping and image warping. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.3964&rep=rep1&type=pdf. 4

[HKP*99] HEIGL B., KOCH R., POLLEFEYS M., DENZLER J., VAN GOOL L.: Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Mustererkennung 1999*. Springer, 1999. 2

[HP98] HOCHBAUM D. S., PATHRIA A.: Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL) 45*, 6 (1998). 2, 3, 4

[HPP*18] HEDMAN P., PHILIP J., PRICE T., FRAHM J.-M., DRETTAKIS G., BROSTOW G.: Deep blending for free-viewpoint image-based rendering. *ACM Trans. on Graphics (TOG) 37*, 6 (2018). 1, 2, 5, 6, 9, 10, 12, 13

[HRDB16] HEDMAN P., RITSCHEL T., DRETTAKIS G., BROSTOW G.: Scalable inside-out image-based rendering. *ACM Trans. on Graphics 35*, 6 (December 2016). 2, 5, 12

[HSGL11] HACOHEN Y., SHECHTMAN E., GOLDMAN D. B., LISCHINSKI D.: Non-rigid dense correspondence with applications for image enhancement. *ACM transactions on graphics (TOG) 30*, 4 (2011). 3

[HSGL13] HACOHEN Y., SHECHTMAN E., GOLDMAN D. B., LISCHINSKI D.: Optimizing color consistency in photo collections. *ACM Trans. on Graphics (TOG) 32*, 4 (2013). 3

[HSM*21] HEDMAN P., SRINIVASAN P. P., MILDENHALL B., BARRON J. T., DEBEVEC P.: Baking neural radiance fields for real-time view synthesis. *arXiv preprint arXiv:2103.14645* (2021). 3

[IZZE17] ISOLA P., ZHU J.-Y., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial networks. In *CVPR* (2017). 3

[KCS14] KOPF J., COHEN M. F., SZELISKI R.: First-person hyper-lapse videos. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 1–10. 6

[KCWI12] KYPRIANIDIS J. E., COLLOMOSSE J., WANG T., ISENBERG T.: State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE transactions on visualization and computer graphics 19*, 5 (2012). 3

[KP08] KIM S. J., POLLEFEYS M.: Robust radiometric calibration and vignetting correction. *IEEE transactions on pattern analysis and machine intelligence 30*, 4 (2008). 3

[KPZK17] KNAPITSCH A., PARK J., ZHOU Q.-Y., KOLTUN V.: Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG) 36*, 4 (2017), 1–13. 7

[LGL*20] LIU L., GU J., LIN K. Z., CHUA T.-S., THEOBALT C.: Neural sparse voxel fields. *NeurIPS* (2020). 3

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *Proc. SIGGRAPH* (1996). 2

[LSS*19] LOMBARDI S., SIMON T., SARAGIH J., SCHWARTZ G., LEHRMANN A., SHEIKH Y.: Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph. 38*, 4 (July 2019). 3

[LZ21] LASSNER C., ZOLLHÖFER M.: Pulsar: Efficient sphere-based neural rendering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021). 3

[MB95] MCMILLAN L., BISHOP G.: Plenoptic modeling: An image-based rendering system. In *Proc. SIGGRAPH* (1995). 2

[MBRS*21] MARTIN-BRUALLA R., RADWAN N., SAJJADI M. S. M., BARRON J. T., DOSOVITSKIY A., DUCKWORTH D.: NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR* (2021). 3

[MSOC*19] MILDENHALL B., SRINIVASAN P. P., ORTIZ-CAYON R., KALANTARI N. K., RAMAMOORTHI R., NG R., KAR A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. on Graphics (TOG) 38*, 4 (2019). 3

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision* (2020), Springer, pp. 405–421. 1, 3

[MTZM18] MECHREZ R., TALMI I., ZELNIK-MANOR L.: The contextual loss for image transformation with non-aligned data. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 768–783. 8

[PLRD21] PRAKASH S., LEIMKÜHLER T., RODRIGUEZ S., DRETTAKIS G.: Hybrid image-based rendering for free-view synthesis. *Proc. of the ACM on Computer Graphics and Interactive Techniques 4*, 1 (May 2021). 5

[PSB*20] PARK K., SINHA U., BARRON J. T., BOUAZIZ S., GOLDMAN D. B., SEITZ S. M., BRUALLA R.-M.: Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948* (2020). 3

[PZ17] PENNER E., ZHANG L.: Soft 3d reconstruction for view synthesis. *ACM Trans. on Graphics (TOG) 36*, 6 (2017). 3

[RDB18] RUDER M., DOSOVITSKIY A., BROX T.: Artistic style transfer for videos and spherical images. *International Journal of Computer Vision 126*, 11 (2018). 3

[Rea18] REALITY C.: Realitycapture reconstruction software. https://www.capturingreality.com/Product, 2018. 9

[RK20] RIEGLER G., KOLTUN V.: Free view synthesis. In *European Conference on Computer Vision* (2020), Springer, pp. 623–640. 1, 2, 3, 4, 9, 10, 13

[RK21] RIEGLER G., KOLTUN V.: Stable view synthesis. In *CVPR* (2021). 2, 9, 10

[RPHD20] RODRIGUEZ S., PRAKASH S., HEDMAN P., DRETTAKIS G.: Image-based rendering of cars using semantic labels and approximate reflection flow. *Proc. of the ACM on Computer Graphics and Interactive Techniques 3*, 1 (may 2020). 12

[RPLG21] REISER C., PENG S., LIAO Y., GEIGER A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *arXiv preprint arXiv:2103.13744* (2021). 3

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum* (2002), vol. 21, Wiley Online Library. 4

[SDZ*21] SRINIVASAN P. P., DENG B., ZHANG X., TANCIK M., MILDENHALL B., BARRON J. T.: Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *CVPR* (2021). 1, 3

[SKG*12] SINHA S. N., KOPF J., GOESELE M., SCHARSTEIN D., SZELISKI R.: Image-based rendering for scenes with reflections. *ACM Trans. on Graphics (TOG) 31*, 4 (2012). 12

[SSS06] SNAVELY N., SEITZ S. M., SZELISKI R.: Photo tourism: exploring photo collections in 3d. In *Proc. SIGGRAPH*. 2006. 2

[STH*19] SITZMANN V., THIES J., HEIDE F., NIESSNER M., WETZSTEIN G., ZOLLHÖFER M.: Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR* (2019). 3

[SXZ*20] SUN T., XU Z., ZHANG X., FANELLO S., RHEMANN C., DEBEVEC P., TSAI Y.-T., BARRON J. T., RAMAMOORTHI R.: Light stage super-resolution: continuous high-frequency relighting. *ACM Trans. on Graphics (TOG) 39*, 6 (2020). 6

[TDDD18] THONAT T., DJELOUAH A., DURAND F., DRETTAKIS G.: Thin structures in image based rendering. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library. 2, 7

[TFK*20] TEXLER O., FUTSCHIK D., KUČERA M., JAMRIŠKA O., ŠÁRKA SOCHOROVÁ, CHAI M., TULYAKOV S., SÝKORA D.: Interactive video stylization using few-shot patch-based training. *ACM Trans. on Graphics 39*, 4 (2020). 3

[TFT*20] TEWARI A., FRIED O., THIES J., SITZMANN V., LOMBARDI S., SUNKAVALLI K., MARTIN-BRUALLA R., SIMON T., SARAGIH J., NIESSNER M., PANDEY R., FANELLO S., WETZSTEIN G., ZHU J.-Y., THEOBALT C., AGRAWALA M., SHECHTMAN E., GOLDMAN D. B., ZOLLHÖFER M.: State of the Art on Neural Rendering. *Computer Graphics Forum (EG STAR 2020)* (2020). 2

[TTS18] TULSIANI S., TUCKER R., SNAVELY N.: Layer-structured 3d scene inference via view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 302–317. 5

[UVL18] ULYANOV D., VEDALDI A., LEMPITSKY V. S.: Deep image prior. In *CVPR* (2018). 3

[WGSJ20] WILES O., GKIOXARI G., SZELISKI R., JOHNSON J.: Synsin: End-to-end view synthesis from a single image. In *CVPR* (2020). 2, 3, 4

[WMG14] WAECHTER M., MOEHRLE N., GOESELE M.: Let there be color! large-scale texturing of 3d reconstructions. In *European conference on computer vision* (2014), Springer. 3

[YLT*21] YU A., LI R., TANCIK M., LI H., NG R., KANAZAWA A.: Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024* (2021). 3

[YSW*19] YIFAN W., SERENA F., WU S., ÖZTIRELI C., SORKINE-HORNUNG O.: Differentiable surface splatting for point-based geometry processing. *ACM Trans. on Graphics 38*, 6 (2019). 3, 4

[ZCC16] ZHANG E., COHEN M. F., CURLESS B.: Emptying, refurnishing, and relighting indoor spaces. *ACM Trans. on Graphics (TOG) 35*, 6 (2016). 3

[ZDM19] ZHANG H., DAUPHIN Y. N., MA T.: Residual learning without normalization via better initialization. In *ICLR* (2019). 6

[ZK14] ZHOU Q.-Y., KOLTUN V.: Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Trans. on Graphics (TOG) 33*, 4 (2014). 3

[ZKR*17] ZAHEER M., KOTTUR S., RAVANBAKHSH S., POCZOS B., SALAKHUTDINOV R. R., SMOLA A. J.: Deep sets. In *Advances in Neural Information Processing Systems* (2017), vol. 30. 6

[ZRSK20] ZHANG K., RIEGLER G., SNAVELY N., KOLTUN V.: Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492* (2020). 1, 9, 10, 13