PointNeRF++: A multi-scale, point-based Neural Radiance Field

Weiwei Sun[1] Eduard Trulls[2] Yang-Che Tseng[1] Sneha Sambandam[1]
Gopal Sharma[1] Andrea Tagliasacchi[3,4,5] Kwang Moo Yi[1]

[1]University of British Columbia [2]Google Research
[3]Google DeepMind [4]Simon Fraser University [5]University of Toronto
https://pointnerfpp.github.io

Abstract. Point clouds offer an attractive source of information to complement images in neural scene representations, especially when few images are available. Neural rendering methods based on point clouds do exist, but they do not perform well when the point cloud quality is low—*e.g.* sparse or incomplete, which is often the case with real-world data. We overcome these problems with a simple representation that aggregates point clouds at multiple scale levels with sparse voxel grids at different resolutions. To deal with point cloud sparsity, we average across multiple scale levels—but only among those that are valid, *i.e.*, that have enough neighboring points in proximity to the ray of a pixel. To help model areas without points, we add a global voxel at the coarsest scale, thus unifying "classical" and point-based NeRF formulations. We validate our method on the NeRF Synthetic, ScanNet, and KITTI-360 datasets, outperforming the state of the art, with a significant gap compared to other NeRF-based methods, especially on more challenging scenes.

Keywords: Point Cloud · Multi-Scale · Neural Radiance Field

1 Introduction

With the introduction of Neural Radiance Fields (NeRF) [29], the quality of novel-view synthesis from a collection of images has increased dramatically. However, the problem is far from solved when field-of-view overlaps sparsely amongst cameras [7, 21, 54], which makes them difficult to apply to many uncontrolled, real-world scenarios. Researchers have attempted to solve this problem in various ways, including content-based regularization [21], patch-based regularization [31], image features [54], or diffusion priors [11, 51].

One way to address this issue is to leverage point clouds obtained from additional sensors and/or photogrammetry [32, 42, 53]. The use of point clouds (as a representation) for neural rendering was pioneered by PointNeRF [53], which demonstrated that point clouds can indeed help achieve higher-quality renderings. However, as we demonstrate through experiments, the benefits of

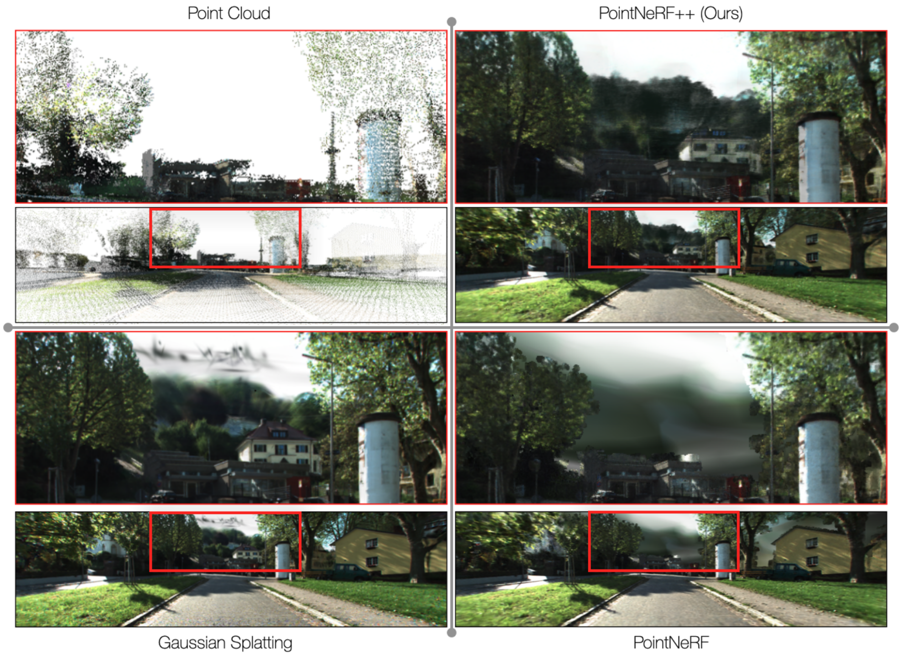


Fig. 1: Teaser – We introduce a novel volume-rendering framework to effectively leverage point clouds for Neural Radiance Fields. Our formulation aggregates points over multiple scales—including a global scale governing the entire scene, equivalent to the standard, point-agnostic NeRF. Our solution leads to much better novel-view synthesis in challenging real-world situations with sparse or incomplete point clouds. Here, we show example renderings from the `KITTI-360` test set.

PointNeRF diminish when point clouds are sparse and/or incomplete. This is often the case in most real-world applications, such as for point clouds obtained by LiDAR scanners in autonomous-driving datasets [3,15–17,20,27,45]. We posit that this shortcoming is mainly due to a missing key element: the lack of *multi-scale* modeling within the architecture of PointNeRF. Multi-scale modeling is helpful in point cloud processing, as small 'holes' (regions without points) can often be naturally filled-in via multi-scale aggregation. We liken this intuition to that followed by two seminal papers in point cloud semantic understanding—PointNet [38] and PointNet++ [39]—where the latter improved upon the former by simply introducing a multi-scale network design, and the notion of hierarchical structure.

In this paper, we introduce a simple multi-scale representation for point cloud-based rendering. Specifically, we aggregate point clouds at various scale levels, defined as voxel grids (Sec. 3.1), up to a scale level that encompasses the *entire* scene. We then use this multi-scale representation to volume-render as in PointNeRF (Sec. 3.2)—but instead of averaging features locally, we do so across multiple scale levels. This allows us to naturally deal with the sparsity of point clouds, without the need for failure-prone heuristics such as 'pruning' and ‘growing’ from PointNeRF [53]. To account for the large support region required at coarser scales, we propose to replace the commonly used Multi-Layer Perceptron (MLP) with a tri-plane representation (Sec. 3.3). We note that using a single voxel at the coarsest scale (*i.e.*, global) is equivalent to a ‘standard’ (*i.e.*, not point-based) NeRF model. Therefore, in a sense, our solution *unifies* classical with point cloud-based NeRF formulations (Sec. 3.1).

As we illustrate in Figure 1, our approach results in novel-view synthesis that is of significantly higher quality than previous methods. Compared to PointNeRF, our approach is able to deal with regions with both high and low point cloud density, even those without points (highlighted with red boxes). The recently popular 3D Gaussian Splatting [23] also suffers at these empty regions as Gaussians are often initialized from point clouds. We evaluate our method across three datasets, NeRF Synthetic [29], ScanNet [9], and KITTI-360 [27], significantly outperforming the state of the art (Sec. 4).

We summarize our main contributions:

– we introduce an effective multi-scale representation for point-based NeRF;
– we propose to incorporate a global voxel/scale, uniting “classical” and point-based NeRF formulations;
– we propose to use a tri-plane representation for coarser scales to effectively cover larger support regions;
– we outperform all baselines, and specifically show large improvements over point-based NeRF, especially when the point clouds are sparse or incomplete.

2 Related work

The introduction of Neural Radiance Fields [29] represented a paradigm shift for scene representation and realistic novel-view synthesis. NeRF employs a 5D implicit function to model a scene through a continuous volumetric approach, which estimates both density and radiance for any given position and direction. Among many applications [14], NeRFs have been used to reconstruct individual objects [29] and unbounded scenes [2], in uncontrolled [8, 28, 52] or dynamic environments [22, 33, 34, 36, 37], in few-shot settings [7, 21, 31, 51, 54] and large urban landscapes [42, 46, 50].

Accelerated training. While NeRF yields remarkable results, this comes at the cost of long training time, owing to the need to evaluate large MLP models hundreds of times for each pixel. The prevailing approach to tackling this issue involves making a trade-off between compute and memory. This is achieved by storing features within various types of grids, including dense grids [13], sparse grids [18], multi-resolution hash grids [30], large sets of small MLPs [40], low-rank tensor approximations of dense grids [6, 12], and hybrid planar and volumetric representations [41].

Neural rendering with point clouds. While the techniques above can train efficiently, it is difficult to adapt them to model large environments. An alternative approach is to use point clouds to model the geometric structure of the scene [4, 5, 23, 32, 53]. Point clouds can have variable density, helping allocate

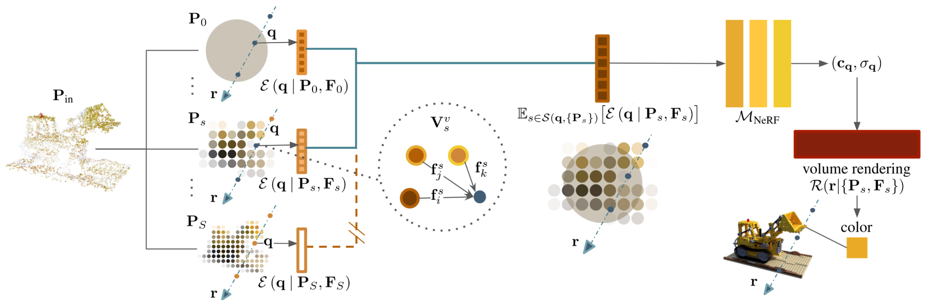

$\mathbf{P}_0$
$\mathbf{q}$
$\mathbf{r}$
$\mathcal{E}(\mathbf{q} \mid \mathbf{P}_0, \mathbf{F}_0)$
$\mathbf{P}_{\text{in}}$
$\mathbf{P}_s$
$\mathbf{q}$
$\mathbf{r}$
$\mathcal{E}(\mathbf{q} \mid \mathbf{P}_s, \mathbf{F}_s)$
$\mathbf{P}_S$
$\mathbf{q}$
$\mathbf{r}$
$\mathcal{E}(\mathbf{q} \mid \mathbf{P}_S, \mathbf{F}_S)$
$\mathbf{V}_s^v$
$\mathbf{f}_j^s$
$\mathbf{f}_k^s$
$\mathbf{f}_i^s$
$\mathbb{E}_{s \in \mathcal{S}(\mathbf{q}, \{\mathbf{P}_s\})}[\mathcal{E}(\mathbf{q} \mid \mathbf{P}_s, \mathbf{F}_s)]$
$\mathbf{r}$
$\mathcal{M}_{\text{NeRF}}$
$(\mathbf{c}_\mathbf{q}, \sigma_\mathbf{q})$
volume rendering
$\mathcal{R}(\mathbf{r} | \{\mathbf{P}_s, \mathbf{F}_s\})$
color
$\mathbf{r}$

Fig. 2: Overview – Given an input point cloud, we aggregate it over multi-scale voxel grids (Sec. 3.1). For clarity, we draw the voxel grids in 2D. We then perform volume rendering based on points, relying on feature vectors stored thereon, which we aggregate across multiple scales (Sec. 3.2). Importantly, when aggregating across scales, we only take into account 'valid' scales, *i.e.*, those with nearby points—indicated with **solid blue lines** and illustrated as the two overlaid scales in the middle—naturally dealing with incomplete/sparse point clouds. The coarsest scale (the top row in the figure) is a single, global voxel, equivalent to standard NeRF—*i.e.*, it is not point-based.

computational resources where needed, and conveniently (not) represent empty space. To perform volume rendering, point cloud features are queried in the *local* neighborhood of a ray to produce density and color. These approaches can be classified on the basis of their neural point representations, *e.g.* per-point features [5, 53], factorized volumetric representations [19], tetrahedral meshes [25], and learnable Gaussians [23].

With PointNeRF, Xu et al. [53] and Chang et al. [5] use point cloud data to learn per-scene representations, by querying per-point features within a local neighborhood. Kulhanek et al. [25] create tetrahedra using the points from COLMAP [44] and use barycentric interpolation to query the features within a tetrahedron. Gaussian splatting [23] represents a 3D scene with 3D anisotropic Gaussians initialized by COLMAP, and optimizes their location to faithfully represent the scene. Despite the high rendering quality, overall, Gaussian splatting is limited by the heuristics that they use to grow and prune points, similar to PointNeRF. For example, as shown in Fig. 1. In contrast to these works, our approach builds a hierarchy of feature representation, efficiently aggregating features in the local neighborhood at different levels; does not require optimizing the location of the points nor heuristics to grow and prune points; and leads to superior performance even with sparse or incomplete point clouds.

Finally, rather than using geometric proximity, one can learn a point-to-query affinity function via transformers. Ost et al. [32] use transformers to combine features of points along a ray to predict its color. A shortcoming of this approach is that it does not take into account occlusions and combines all points in the neighborhood of a ray. Similarly, Chang et al. [4] use a set-transformer to find ray-surface intersections and use local features and blending weights to estimate ray colors. Both of these approaches are different from ours, as we employ geometric, rather than learned, proximity.

3 Method

An overview of our method is shown in Fig. 2. We build a representation starting from an input point cloud, which we then use to volume-render [29] a scene. Specifically, given an input point cloud $\mathbf{P}_{\text{in}}$, we spatially aggregate the point cloud to build a *point cloud hierarchy* with S levels. Denoting this operation as $\mathcal{A}(.)$, we write

$$\{\mathbf{P}_s\}_{s=1}^{S} = \mathcal{A}(\mathbf{P}_{\text{in}}), \tag{1}$$

and equip each point cloud level $\mathbf{P}_s$ with randomly initialized point features $\mathbf{F}_s$. We then optimize the features $\mathbf{F}_s$ by volume-rendering them along a ray (pixel) $\mathbf{r}$ by $\mathcal{R}(.)$, so that the estimated color matches that of the ground-truth pixels $\mathbf{C}_{\text{gt}}$, using a photometric loss:

$$\underset{\{\mathbf{F}_s\}}{\arg\min}\ \mathbb{E}_{\mathbf{r}}\left[\|\mathbf{C}_{\text{gt}}(\mathbf{r}) - \mathcal{R}(\mathbf{r}|\{\mathbf{P}_s, \mathbf{F}_s\})\|_2^2\right]. \tag{2}$$

We next detail our multi-scale aggregation strategy to define a hierarchical representation for point clouds (Sec. 3.1), and how we use it to volume-render a scene (Sec. 3.2). Finally, we propose to use a tri-plane-based feature representation in lieu of MLPs, in order to obtain a good trade-off between representation capacity and speed (Sec. 3.3).

3.1 Multi-scale aggregation – $\mathcal{A}$

We first detail our aggregation operation $\mathcal{A}$ in Eq. (1). To obtain a point cloud that represents a desired scale level s, we cluster based on voxels. At level s, consider a regular grid of resolution $V_s \times V_s \times V_s$, consisting of a set of voxels $\{\mathbf{V}_s^v\}$. We perform voxel-wise clustering to determine one representative point per voxel as

$$\mathbf{p}_s^v = \mathbb{E}_{\mathbf{p}\in\mathbf{V}_s^v}[\mathbf{p}] \quad \text{s.t.} \quad \mathbf{p} \in \mathbf{P}_{\text{in}}. \tag{3}$$

Importantly, note that this is performed only over non-empty voxels, hence the resulting representation is *sparse*. Note also that the aggregation is built at each scale level *independently*, and that while some fine-grained scales may not have valid aggregated points, more space regions will be covered at the coarser scales. This allows for point clouds with *variable density*, or even *incomplete* ones to a certain degree, to be dealt with naturally. Finally, we set the coarsest voxel to cover the entire scene, effectively setting $\mathbf{p}_0^0 = \mathbb{E}_{\mathbf{p}\in\mathbf{P}_{\text{in}}}[\mathbf{p}]$. This coarsest scale can also be understood as a global NeRF model that is independent of the local distribution of the point cloud—providing a unified representation for both standard and point-based NeRF.

3.2 Point-based rendering – $\mathcal{R}$

We use volume rendering to render an image from the multi-scale point cloud. Given a set of quadrature points along ray $\mathbf{q} \in \mathbf{r}$, let us denote the volume rendering integral [29]

$$\hat{\mathbf{C}}_{\mathbf{r}} = \mathcal{R}_{\mathbf{q}\in\mathbf{r}}(\mathbf{c}_{\mathbf{q}}, \sigma_{\mathbf{q}}), \tag{4}$$

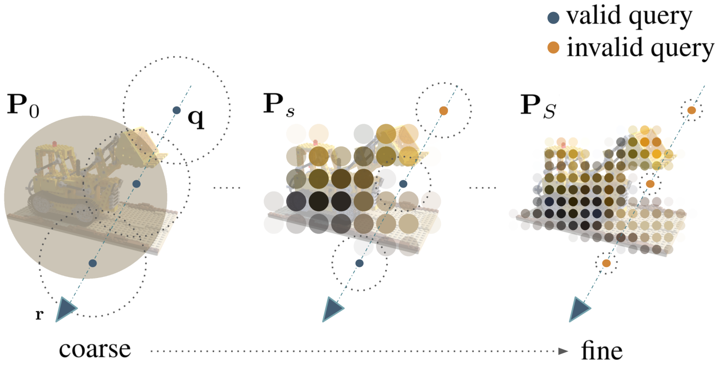


Fig. 3: Increasing coverage with multiple scales – We illustrate our sparse, hierarchical representation at three granularity levels, including a single, global voxel (left). We also show three query points, with their respective neighborhoods (dotted circles) at each scale level—color-coded in **blue** if they have neighbouring features, and in **orange** if they do not. Our multi-scale approach naturally fills in empty regions, removing the need for failure-prone region-growing heuristics [53]. Drawn in 2D, for clarity.

where $\mathbf{c}_\mathbf{q}$ is the radiance and $\sigma_\mathbf{q}$ is the density of a location $\mathbf{q}$ in space. To obtain these values, we operate on our point cloud hierarchy, as opposed to the raw point cloud $\mathbf{P}_{\text{in}}$ as in PointNeRF [53]. More explicitly, we extend [53] to multiple scales by averaging over valid scale levels, *i.e.*, scale levels with any points within the vicinity of $\mathbf{q}$:

$$\mathbf{c}_\mathbf{q}, \sigma_\mathbf{q} = \mathcal{M}\Big(\mathbb{E}_{s \in \mathcal{S}(\mathbf{q}, \{\mathbf{P}_s\})}\left[\mathcal{E}\left(\mathbf{q} \mid \mathbf{P}_s, \mathbf{F}_s\right)\right]\Big), \tag{5}$$

where $\mathcal{S}(\mathbf{q}, \{\mathbf{P}_s\})$ is the set of valid scale levels associated to query $\mathbf{q}$; $\mathcal{E}$ is the feature extraction operation in PointNeRF [53] that converts the point cloud into a feature embedding at the query location $\mathbf{q}$; and $\mathcal{M}$ is an MLP that converts those feature embeddings into radiance and density. We now describe $\mathcal{S}$ and $\mathcal{E}$ in more detail.

Valid scale levels – $\mathcal{S}(\mathbf{q}, \{\mathbf{P}_s\})$**.** Given a scale S, define $\mathcal{N}$ the local neighbors of $\mathbf{q}$ within distance τV_s, where τ is the threshold ratio:

$$\mathcal{N}(\mathbf{q}, \mathbf{P}_s) = \big\{\mathbf{p} \mid \mathbf{p} \in \mathbf{P}_s \;\;\&\;\; \|\mathbf{p} - \mathbf{q}\|_2 \le \tau V_s\big\}. \tag{6}$$

which is then aggregated across levels to define:

$$\mathcal{S}(\mathbf{q}, \{\mathbf{P}_s\}) = \{s \mid \mathcal{N}(\mathbf{q}, \mathbf{P}_s) \neq \varnothing\}, \tag{7}$$

Point cloud to feature embedding – $\mathcal{E}(\mathbf{q} \mid \mathbf{P}_s, \mathbf{F}_s)$**.** We aggregate the features within the support defined by Eq. (6) using normalized inverse-distance weights $w(\mathbf{p}, \mathbf{q}) = (\|\mathbf{p} - \mathbf{q}\|_2 + \varepsilon)^{-1}$, where ε is a small number to avoid numerical problems:

$$\mathcal{E}(\mathbf{q}|\mathbf{P}_s, \mathbf{F}_s) = \frac{\sum_{\mathbf{p} \in \mathcal{N}(\mathbf{q}, \mathbf{P}_s)} w(\mathbf{p}, \mathbf{q})\, \mathcal{F}(\mathbf{f}_\mathbf{p}, \mathbf{p} - \mathbf{q})}{\sum_{\mathbf{p} \in \mathcal{N}(\mathbf{q}, \mathbf{P}_s)} w(\mathbf{p}, \mathbf{q})}, \tag{8}$$

where $\mathcal{F}$ is a learnable function, and $\mathbf{f}_\mathbf{p}$ is the feature in $\mathbf{F}_s$ corresponding to $\mathbf{p} \in \mathbf{P}_s$. Note that this is a simplified version of PointNeRF [53], as we do not use ‘per-point‘ weights [53, Sec. 4.1], which we experimentally found to not contribute to improvements in rendering quality. Rather than relying on large MLPs to implement $\mathcal{F}$ at coarse levels s, we employ a tri-plane representation, described in Sec. 3.3. This effectively increases the representation power of $\mathcal{F}$ at coarse levels so that less populated regions in space can still be modeled effectively, without incurring an excessive computational burden.

3.3 Per-point tri-plane features

As illustrated in Figure 3, points in coarser levels represent larger regions, and thus require preserving more information into each point feature. We could solve this by increasing either the feature dimension or the capacity of the MLPs used to parameterize $\mathcal{F}$. They both come with a hefty price, greatly increasing the computational cost incurred to evaluate $\mathcal{F}$. Instead, we build on recently-proposed factorized representations [12], and represent local features with a local *tri-plane* factorization. In more detail, we store features within three orthogonal feature planes $\mathbf{f}_\mathbf{p} \equiv \{\mathbf{f}_\mathbf{p}^{XY}, \mathbf{f}_\mathbf{p}^{YZ}, \mathbf{f}_\mathbf{p}^{XZ}\}$, which are then accessed at (local) 3D coordinates $\mathbf{u} = (\mathbf{q} - \mathbf{p})/(\tau V_s)$:

$$\mathcal{F}(\mathbf{f}_\mathbf{p}, \mathbf{u}) = \mathbf{f}_\mathbf{p}^{XY}[\mathbf{u}] + \mathbf{f}_\mathbf{p}^{YZ}[\mathbf{u}] + \mathbf{f}_\mathbf{p}^{XZ}[\mathbf{u}], \tag{9}$$

where $\mathbf{f}_\mathbf{p}^{**}[\mathbf{u}]$ denotes querying the plane at position $\mathbf{u}$ with bilinear interpolation. We combine tri-planes at coarser levels with the standard MLPs at finer levels, where we find the latter are sufficient (see Sec. 4.1 for details). At first glance, Eq. (9) may seem like a large deviation from using an MLP, since the features seem to be independent of each other, due to the lack of a *shared* MLP. Note, however, that those features are eventually processed by the shared decoder $\mathcal{M}$ that converts them into radiance field values. Finally, we note that at the coarsest scale level, *i.e.*, the global voxel, our representation is effectively K-Planes [12].

4 Results

4.1 Experimental setup

Datasets and metrics. We primarily use Peak Signal-to-Noise Ratio (PSNR) as a metric, and also structural (SSIM [49]) and perceptual (LPIPS [55]) similarity. We evaluate our method on three well-known datasets:

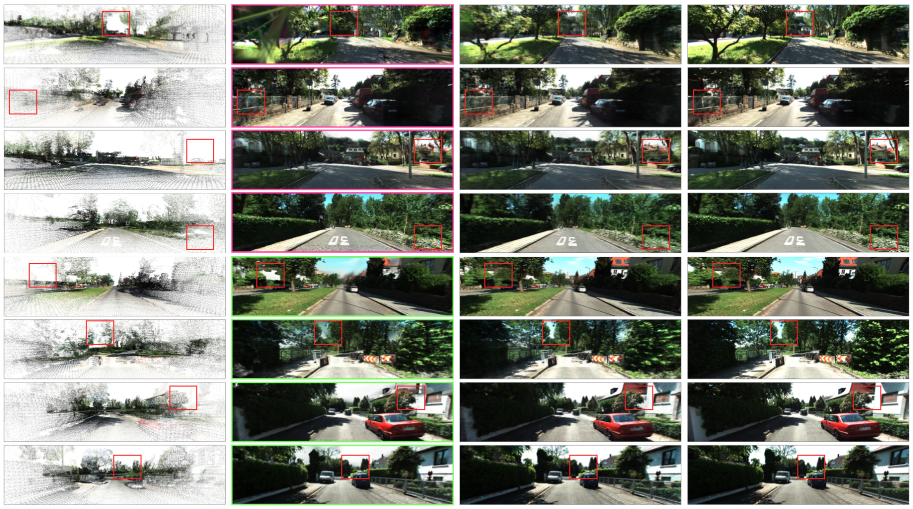
– `KITTI-360` [27] is a recent benchmark of outdoor driving sequences, highly challenging due to the sparsity of views, which have much less visual overlap than other datasets. Each sequence consists of an average of 80 images. We use a random 10% subset for validation purposes, and also for our ablation study, as the ground truth for the test set is not publicly available. To obtain results on the test set, we follow the standard practice of training with the entire training set, to roughly the same number of iterations required for convergence, discovered with the validation split. We use the point clouds provided with the dataset, from LiDAR scans that are accumulated over all views. As this accumulated point cloud is very dense, we resample it over a grid with a cell size of 8cm, and remove points outside the camera frustum of the training views to make it more tractable.

– `ScanNet` [9] is a dataset of indoor scans. We use the point clouds provided with the dataset, which are sampled from mesh reconstructions using RGB-D cameras with BundleFusion [10]. Following PointNeRF [53], we evaluate on two scenes, Scene-101 and Scene-241. The point cloud in Scene-101 has more incomplete regions, which makes it harder than Scene-241. As in PointNeRF [53], we sample 20% of the images, *i.e.* 1463 images for Scene-241, and 1000 images for Scene-101, for training, and use the rest for evaluation. We use the code provided by [53].
– `NeRF Synthetic` [29] is a synthetic dataset with eight objects, each with 100 training images and 200 test images. The images are purely synthetic, rendered with Blender. We use this dataset, as in PointNeRF [53], to validate our method when the scene is favorable to the standard NeRF setting. We take the point clouds provided by PointNeRF [53], which are obtained with COLMAP [44].

Implementation details. We implement our method with PyTorch [35]. We use a total of 5 scale levels, including the global scale. We use a tri-plane resolution of 512×512 for the global scale level. For the largest (*i.e.*, coarsest) two of the remaining scale levels we use tri-planes, where each tri-plane is built as a small two-layer pyramid with 4×4 and 2×2 grid. For all tri-planes, we store 32-dimensional feature vectors followed by a four-layer MLP with 64 neurons. For the remaining two (*i.e.*, finest) scales, we simply use 32-dimensional point features and a four-layer MLP with 64 neurons. To further allow for the global scale to capture details that may be beyond the capacity of its resolution, we augment the features extracted from the global tri-plane with positional encodings with 5 frequencies, as in [53]. We found this to be especially important when modeling large scenes, such as for `KITTI-360`. For $\mathcal{M}$, we use one linear layer for density prediction and a four-layer MLP with 64 neurons for its hidden layers for color prediction.

To speed up neighborhood search, we rely on voxel-grid-based approximate nearest neighbors, as in [53]. We use the same search radius as our neighborhood threshold τ in Eq. (6) after normalizing so that the approximate search is equivalent to a ball query. For speed-ups and to limit GPU memory growth, we set the maximum number of neighboring points to 8 for `ScanNet` and `NeRF Synthetic`, and 6 for `KITTI-360`, as the scenes are larger. We follow PointNeRF [53] to sample 400 points for each ray on `ScanNet` and `NeRF Synthetic`. As `KITTI-360` is larger, we sample 1,000 points for each ray to compensate. We use the contraction function of [2] for regions outside the bounding box of the point cloud. For `KITTI-360`, as in [42], we model the sky using a four-layer MLP that maps ray direction to color. To improve sample efficiency, we also use a proposal network [2].

We train our model with a single NVidia V100 GPU for 200k iterations. For the learning rate we follow [53] and use the same exponential decaying scheduling



(a) Input point cloud [9] (b) Gauss. Splat. [23] & PointNeRF [53] (c) PointNeRF++ (Ours) (d) Ground truth

Fig. 4: Examples on `KITTI-360` – We show novel-view renderings obtained with our method, 3D Gaussian splatting [23] (pink colored, 1-4 rows) and PointNeRF [53] (green colored, 4-8 row) on a challenging outdoors dataset, using the same point clouds as input. Our approach provides significantly sharper renderings with more details, and better coverage in areas without points, where Gaussian Splatting and PointNeRF produce highly salient artifacts highlighted with red boxes.

with an initial learning rate of 5e-4 for $\mathcal{M}$ and a larger initial learning rate of 2e-3 for $\mathbf{F}$. Following [53], we decay every 1000k steps with a rate of 0.1. We will make our code available with an Apache 2.0 license, for reproducibility.

4.2 KITTI-360 results – Fig. 4 and Tab. 1

We first compare our method to the state of the art on `KITTI-360` [27], a challenging outdoors dataset with incomplete point clouds from real LiDAR measurements.

Baselines. We report numbers on the hidden test set, which requires uploading samples to the evaluation server to compare with methods featured on the public leaderboard. We also report numbers on our validation split for methods that do not have an entry in the public leaderboard. We consider methods based on images and, optionally, semantics [1, 26, 29, 43, 48, 56] as well as those that use the LiDAR point clouds [23, 24, 27, 53]. Note that for all PointNeRF experiments in this paper we use the point 'pruning' and 'growing' heuristics introduced in their work [53, Sec. 4.2], which aim at improving geometry modeling and image rendering quality, and can help deal with point cloud sparsity—our algorithm does not rely on it.

Table 1: Results on KITTI-360 [27] – Our method achieves the best performance among methods that supervise only with *color*. It performs on par with those that also rely on *semantics*. We provide results on the (public) validation set and the (hidden) test set as some baselines have results for one, but not the other.

			Validation			Test		
		Uses points	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
w/ Sem.	Nerflets [56]	✗	–	–	–	21.69	–	–
	PNF [26]	✗	–	–	–	22.07	0.820	0.221
	PLANeRF [48]	✗	–	–	–	**22.64**	**0.855**	**0.200**
Color only	FVS [43]	✗	–	–	–	20.00	0.790	0.193
	NeRF [29]	✗	–	–	–	21.18	0.779	0.343
	Mip-NeRF [1]	✗	–	–	–	21.54	0.778	0.365
	PBNR [24]	✓	–	–	–	19.91	0.811	0.191
	PCL [27]	✓	–	–	–	12.81	0.576	0.549
	Gauss. Splat. [23]	✓	18.59	0.642	**0.257**	22.08	**0.844**	**0.139**
	PointNeRF [53]	✓	17.63	0.629	0.337	19.44	0.796	0.266
	Ours	✓	**20.05**	**0.665**	0.305	**22.44**	0.828	0.212

Table 2: Results on two ScanNet scenes [9] as pre-processed by [53] – Our method outperforms all others, especially PointNeRF, the method most similar to ours, by a large margin demonstrating the effectiveness of our multi-scale approach.

		Avg.			Scene-101	Scene-241
	Uses points	PSNR↑	SSIM ↑	LPIPS↓	PSNR↑	PSNR↑
NeRF [29]	✗	24.43	0.670	0.494	27.16	21.69
Gauss. Splat. [23]	✗	29.56	0.812	0.301	29.01	30.11
Gauss. Splat. [23]	✓	29.93	**0.818**	0.275	29.55	30.31
PointNeRF [53]	✓	25.92	0.784	0.263	21.98	29.86
Ours	✓	**30.56**	0.808	**0.238**	**30.27**	**30.85**

Discussions. We show qualitative highlights in Fig. 4 and report results on Tab. 1. Our method achieves a new state of the art in the color-only category among NeRF methods, and performs on par with methods that also use semantic supervision and Gaussian Splatting. Importantly, we significantly improve over other point-based methods. Compared with PointNeRF, our approach yields better renderings on regions where the point cloud is sparse, and the global scale allows us to tackle those with no nearby points, such as structures too far away to be captured by LiDAR. Also of note is how Gaussian splatting, while it provides improved rendering quality in terms of SSIM and LPIPS thanks to its SSIM-based loss, can have failure modes, as highlighted in Fig. 4. We thus believe combining our strategy of utilizing our multi-scale strategy with Gaussian splatting can further lead to performance improvements, but we leave as future work. Please refer to the appendix for video examples.

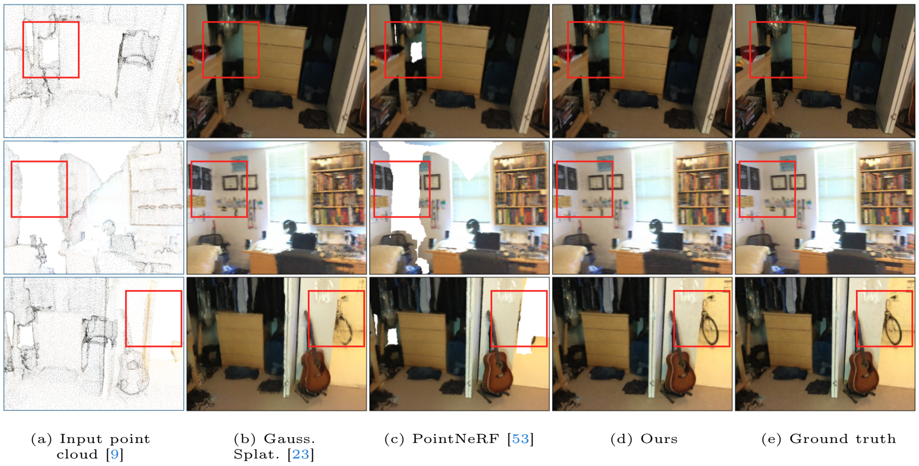


Fig. 5: Examples on ScanNet – PointNeRF fails to reconstruct the scene on regions where the point cloud is empty. Both our method and Gaussian Splatting are able to fill them in, but our approach produces cleaner results, with fewer artifacts. This is especially noticeable for Scene-101 (top and bottom rows), where the mesh has large holes where PointNeRF fails to render meaningful pixels, even with their 'growing' heuristic that is aimed towards filling such gaps.

4.3 ScanNet results – Fig. 5 and Tab. 2

Next, we consider indoor scans, using ScanNet [9]. This dataset is less challenging than KITTI-360, but is a typical use-case for point cloud-based neural rendering, and is where the benefit of using point clouds was strongly demonstrated in PointNeRF [53].

Baselines. We compare our method against NeRF [29], PointNeRF [53], and Gaussian Splatting [23]. For the latter, we consider randomly initialized point clouds as well as those provided by the dataset.

Discussions. As shown in Tab. 2 and Fig. 5, our method performs best. NeRF [29], for this dataset does not perform well as the scene is relatively textureless and smooth. PointNeRF [53] improves over it by leveraging the point clouds. It does, however, have issues on Scene-101, because its point cloud has large incomplete areas, which impair its performance, as shown in Fig. 5. Our method is able to cope with these empty regions, thanks to our multi-scale framework. Interestingly, Gaussian Splatting also works better than typical NeRF while trained purely with images, even when starting from random points—point cloud initialization can further improve its performance. This suggests the importance of including the notion of locality introduced by points to the representation. Our method outperforms all baselines, point-based or not. We use point clouds from mesh inputs instead of depth images, also reported by PointNeRF, as the latter are extremely dense (see [53, Tbl. 8]).

Table 3: PSNR↑ on NeRF Synthetic [29] – Our method performs best overall, even on object-centric data with dense point clouds.

	Uses points	**Avg.**	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
NeRF [29]	✗	31.01	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65
Plenoxels [13]	✗	31.76	33.98	25.35	31.83	36.81	34.1	29.14	33.26	29.62
InstantNGP [30]	✗	33.18	35.00	26.02	33.51	37.40	36.39	29.78	36.22	31.10
MipNeRF [1]	✗	33.09	35.14	25.48	33.29	37.48	35.7	**30.71**	36.51	30.41
Gauss. Splat. [23]	✗	33.32	35.83	26.15	34.87	37.72	35.78	30.00	35.36	30.80
FreqPCR [57]	✓	31.24	33.06	25.95	32.19	35.82	31.56	29.69	33.64	27.97
TetraNeRF [25]	✓	32.53	35.05	25.01	33.31	36.16	34.75	29.30	35.49	31.13
Gauss. Splat. [23]	✓	33.60	36.10	**26.17**	**34.87**	**37.77**	36.14	30.15	36.48	31.12
PointNeRF [53]	✓	31.77	35.09	25.01	33.24	35.49	32.65	26.97	35.54	30.18
Ours	✓	**33.70**	**36.32**	26.11	34.43	37.45	**36.75**	30.32	**36.85**	**31.34**

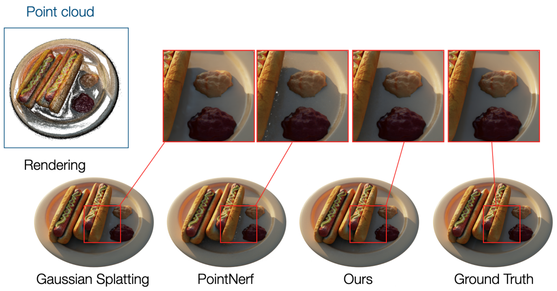


Fig. 6: Examples on NeRF Synthetic – Our multi-scale approach consistently fills in the holes in the input point cloud. PointNeRF relies on 'pruning' and 'growing' heuristics, which can fail where the point cloud is not sufficiently dense, as shown here. While Gaussian Splatting also previous failed to fill in holes with their densification heuristics (Fig. 4 and Fig. 5), for `NeRF Synthetic` these heuristics work well.

4.4 NeRF Synthetic results – Fig. 6 and Tab. 3

Finally, we verify the effectiveness of our method on `NeRF Synthetic`, to demonstrate that it remains helpful even use-cases designed for NeRF.

Baselines. We compare our approach against both methods that only utilize RGB images [29], which this dataset is typically used to evaluate, and those that use point clouds [25, 53, 57], including the recent Gaussian Splatting [23].

Discussions. We report PSNR results in Tab. 3 and provide qualitative examples in Fig. 6. Our method still performs best overall, slightly ahead of Gaussian Splatting. More importantly, it outperforms the point-based baselines by a larger margin.

4.5 Ablation study

We thoroughly ablate our method in this section on `KITTI-360` using the validation split. We consider the number of scale levels, using a tri-plane vs an MLP

Table 4: Number of scales vs rendering quality – As expected, more levels lead to better PSNR↑. We *always* use a global scale—'0' corresponds to using only a global scale, '1' the finest scale plus the global scale, and later adding coarser scale levels, up to our full model ('4').

Num. of scales	PSNR↑	SSIM↑	LPIPS↓
0	17.95	0.520	0.442
1	19.88	**0.674**	**0.283**
2	19.89	0.669	0.289
3	19.90	0.666	0.299
4	**20.05**	0.665	0.305

Table 5: Number of points – We show that our method remains applicable to sparser point clouds, with noticeable improvement over points-agnostic model even at drastic downsampling rates (1%).

Ratio of pts.	PSNR↑	SSIM↑	LPIPS↓
0	17.95	0.520	0.442
1	18.71	0.558	0.434
10	19.35	0.621	0.370
full	**20.05**	**0.665**	**0.305**

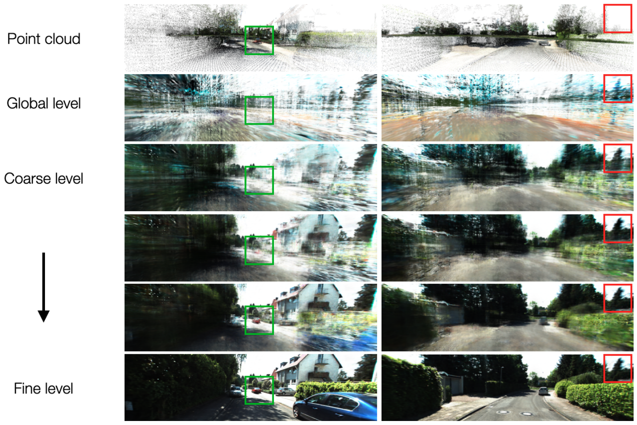


Fig. 7: Rendering across scales – We study the behavior of our hierarchical approach by rendering an image adding one scale level at a time, from the global scale to the finest one. As expected, the coarse scales are responsible for filling in empty regions (highlighted with red boxes) in the point cloud, different from the well-covered regions (highlighted with green boxes) that can be modeled via fine scales.

for $\mathcal{F}$ at the coarsest scale levels, and study the effect of adding a global scale. We also evaluate performance at different levels of point cloud sparsity.

Number of scale levels – Tab. 4 and Fig. 7. We ablate how the number of scale levels affects performance, by training and evaluating models using a different number of scales. We also illustrate what each scale level is *adding*, by rendering views with a multi-scale model adding one scale level at a time, in Fig. 7. As clearly shown in the figure, the global scale is instrumental in rendering accurate pixels in those areas, and each successive scale adds finer details, improving the overall quality of the rendering.

Tri-plane vs MLP – Tab. 6. We also provide an ablation study to evaluate the advantages of using a tri-plane instead of a regular MLP for the parameterized function $\mathcal{F}$. As outlined in Sec. 3.3, we always use MLPs at the two finest scale levels. Using a tri-plane performs slightly better at a similar computational cost.

Table 6: Impact of the tri-plane – We evaluate tri-planes vs. MLPs. Using tri-planes for the coarsest scales improves performance, at a comparable computational cost.

	PSNR↑	SSIM↑	LPIPS↓
MLP	19.59	0.643	0.353
Triplane	**20.05**	**0.665**	**0.305**

Table 7: Using a global feature – We measure PSNR↑ on two datasets for different variants of our approach. Adding a global voxel at the coarsest scale to the hierarchical structure (right), improves performance (left), but is not sufficient by itself (middle).

	PSNR↑	SSIM↑	LPIPS↓
w/o global	19.78	**0.675**	**0.290**
global only	17.95	0.520	0.442
full	**20.05**	0.665	0.305

Using a global voxel – Tab. 7. We evaluate the importance of adding a global voxel at the coarsest scale. We compare three variants of our method: one using four local scales ("w/o global"); one using only the global scale ("global-only"), *i.e.*, a traditional point-agnostic NeRF; and one using both ("full"). The point-based formulation outperforms point-agnostic NeRF, but combining them with our multi-scale-plus-global approach does best.

Number of points – Tab. 5. We measure performance while randomly down-sampling the point cloud with increasing ratios. Our approach performs while even at downsampling rates below 1% and using as few as 10k points.

5 Conclusions

Neural Radiance Fields are a paradigm shift in novel-view synthesis. Despite their promise, and a large number of follow-up papers, challenges persist, particularly when few views of the scene are available. Point clouds provide a very attractive data stream, complementary to images, and often readily available in both indoor and outdoor settings—but have a different set of challenges, due to incompleteness and sparsity. We mitigate this with a simple yet novel multi-scale representation that combines global and local information, yielding significant performance improvements across the board. Our work unifies point cloud-based and standard NeRF pipelines and adapts effectively to variable point densities and empty regions, pushing novel view synthesis on uncontrolled, real-world data closer to practice.

Limitations and future work. While our method provides significant improvements over PointNeRF by combining point-based ones with classic NeRF, the computational cost of our method is naturally bound by classic NeRF—on `NeRF Synthetic`, our method induces a 20% in compute overhead to the classic NeRF backbone that we use. Thus, an interesting research direction would be to combine the strengths of our multi-scale strategy in handling incomplete and sparse point clouds with the high computational-efficiency of 3D Gaussian Splatting [23], especially given the pitfalls of 3D Gaussian Splatting demonstrated in our work.

References

1. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A Multiscale Representation for Anti-aliasing Neural Radiance Fields. In: ICCV (2021) 9, 10, 12
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: unbounded anti-aliased neural radiance fields. CVPR (2022) 3, 8
3. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: NuScenes: A Multimodal Dataset for Autonomous Driving. In: CVPR (2020) 2
4. Chang, J.H.R., Chen, W.Y., Ranjan, A., Yi, K.M., Tuzel, O.: Pointersect: Neural Rendering with Cloud-Ray Intersection. In: CVPR (2023) 3, 4
5. Chang, M., Sharma, A., Kaess, M., Lucey, S.: Neural Radiance Field with LiDAR maps. In: ICCV (2023) 3, 4
6. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: TensoRF: Tensorial Radiance Fields. In: ECCV (2022) 3
7. Chen, A., Xu, Z., Zhao, F., Zhang, X., Xiang, F., Yu, J., Su, H.: MVSNeRF: Fast Generalizable Radiance Field Reconstruction from Multi-view Stereo. In: ICCV (2021) 1, 3
8. Chen, X., Zhang, Q., Li, X., Chen, Y., Feng, Y., Wang, X., Wang, J.: Hallucinated Neural Radiance Fields in the Wild. In: CVPR (2022) 3
9. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: ScanNet: Richly-annotated 3d Reconstructions of Indoor Scenes. In: CVPR. pp. 5828–5839 (2017) 3, 8, 9, 10, 11, 1
10. Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., Theobalt, C.: Bundlefusion: Realtime Globally Consistent 3d Reconstruction Using on-the-fly Surface Reintegration. TOG (2017) 8
11. Deng, C., Jiang, C., Qi, C.R., Yan, X., Zhou, Y., Guibas, L., Anguelov, D.: NeRDi: single-view nerf synthesis with language-guided diffusion as general image priors. In: CVPR (2023) 1
12. Fridovich-Keil, S., Meanti, G., Warburg, F.R., Recht, B., Kanazawa, A.: K-Planes: Explicit Radiance Fields in Space, Time, and Appearance. In: CVPR (2023) 3, 7
13. Fridovich-Keil and Yu, Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance Fields without Neural Networks. In: CVPR (2022) 3, 12
14. Gao, K., Gao, Y., He, H., Lu, D., Xu, L., Li, J.: NeRF: Neural Radiance Field in 3d Vision, a Comprehensive Review. ARXIV (2022) 3
15. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets Robotics: The KITTI Dataset. IJRR (2013) 2
16. Geyer, J., Kassahun, Y., Mahmudi, M., Ricou, X., Durgesh, R., Chung, A.S., Hauswald, L., Pham, V.H., Mühlegg, M., Dorn, S., et al.: A2d2: Audi Autonomous Driving Dataset. ARXIV (2020) 2
17. Gruber, T., Julca-Aguilar, F., Bijelic, M., Heide, F.: Gated2Depth: Real-Time Dense Lidar From Gated Images. In: ICCV (2019) 2
18. Hedman, P., Srinivasan, P.P., Mildenhall, B., Barron, J.T., Debevec, P.: Baking Neural Radiance Fields for Real-Time View Synthesis. In: ICCV (2021) 3
19. Hu, T., Xu, X., Chu, R., Jia, J.: TriVol: Point Cloud Rendering via Triple Volumes. In: CVPR (2023) 4
20. Huang, X., Wang, P., Cheng, X., Zhou, D., Geng, Q., Yang, R.: The ApolloScape Open Dataset for Autonomous Driving and Its Application. TPAMI (2019) 2

21. Jain, A., Tancik, M., Abbeel, P.: Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis. In: ICCV (2021) 1, 3
22. Jiang, W., Yi, K.M., Samei, G., Tuzel, O., Ranjan, A.: Neuman: Neural Human Radiance Field from a Single Video. In: ECCV (2022) 3
23. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D Gaussian Splatting for Real-Time Radiance Field Rendering. TOG (2023) 3, 4, 9, 10, 11, 12, 14
24. Kopanas, G., Philip, J., Leimkühler, T., Drettakis, G.: Point-Based Neural Rendering with Per-View Optimization. In: CGF (2021) 9, 10
25. Kulhanek, J., Sattler, T.: Tetra-NeRF: Representing Neural Radiance Fields Using Tetrahedra. ARXIV (2023) 4, 12
26. Kundu, A., Genova, K., Yin, X., Fathi, A., Pantofaru, C., Guibas, L.J., Tagliasacchi, A., Dellaert, F., Funkhouser, T.: Panoptic Neural Fields: A Semantic Object-aware Neural Scene Representation. In: CVPR (2022) 9, 10
27. Liao, Y., Xie, J., Geiger, A.: KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2d and 3d. PAMI (2022) 2, 3, 7, 9, 10, 1
28. Martin-Brualla, R., Radwan, N., Sajjadi, M.S., Barron, J.T., Dosovitskiy, A., Duckworth, D.: NeRF in the wild: Neural Radiance Fields for Unconstrained Photo Collections. In: CVPR (2021) 3
29. 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: ECCV (2020) 1, 3, 5, 8, 9, 10, 11, 12
30. Müller, T., Evans, A., Schied, C., Keller, A.: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. TOG (2020) 3, 12
31. Niemeyer, M., Barron, J.T., Mildenhall, B., Sajjadi, M.S.M., Geiger, A., Radwan, N.: RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. In: CVPR (2022) 1, 3
32. Ost, J., Laradji, I., Newell, A., Bahat, Y., Heide, F.: Neural Point Light Fields. In: CVPR (2022) 1, 3, 4
33. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable Neural Radiance Fields. In: CVPR (2021) 3
34. Park, K., Sinha, U., Hedman, P., Barron, J.T., Bouaziz, S., Goldman, D.B., Martin-Brualla, R., Seitz, S.M.: HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields. TOG (2021) 3
35. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic Differentiation in PyTorch. In: NIPS-W (2017) 8
36. Peng, S., Dong, J., Wang, Q., Zhang, S., Shuai, Q., Zhou, X., Bao, H.: Animatable Neural Radiance Fields for Modeling Dynamic Human Bodies. In: ICCV (2021) 3
37. Pumarola, A., Corona, E., Pons-Moll, G., Moreno-Noguer, F.: D-NeRF: Neural Radiance Fields for Dynamic Scenes. In: CVPR (2020) 3
38. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CVPR (2016) 2
39. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. NIPS (2017) 2
40. Reiser, C., Peng, S., Liao, Y., Geiger, A.: KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In: ICCV (2021) 3
41. Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P.P., Mildenhall, B., Geiger, A., Barron, J.T., Hedman, P.: MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. SIGGRAPH 2023 (2023) 3
42. Rematas, K., Liu, A., Srinivasan, P.P., Barron, J.T., Tagliasacchi, A., Funkhouser, T., Ferrari, V.: Urban Radiance Fields. In: CVPR (2022) 1, 3, 8

43. Riegler, G., Koltun, V.: Free view synthesis. In: ECCV (2020) 9, 10
44. Schonberger, J.L., Frahm, J.M.: Structure-from-motion Revisited. In: CVPR (2016) 4, 8
45. Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z., Anguelov, D.: Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In: CVPR (2020) 2
46. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-NeRF: Scalable large scene neural view synthesis. In: CVPR (2022) 3
47. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and Deformable Convolution for Point Clouds. In: ICCV (2019) 1
48. Wang, F., Louys, A., Piasco, N., Bennehar, M., Roldão, L., Tsishkou, D.: PlaNeRF: SVD Unsupervised 3D Plane Regularization for NeRF Large-Scale Scene Reconstruction. 3DV (2023) 9, 10
49. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image Quality Assessment: from Error Visibility to Structural Similarity. TIP (2004) 7
50. Wu, X., Xu, J., Zhang, X., Bao, H., Huang, Q., Shen, Y., Tompkin, J., Xu, W.: ScaNeRF: Scalable Bundle-Adjusting Neural Radiance Fields for Large-Scale Scene Rendering. In: TOG (2023) 3
51. Wynn, J., Turmukhambetov, D.: Diffusionerf: Regularizing Neural Radiance Fields with Denoising Diffusion Models. In: CVPR (2023) 1, 3
52. Xu, D., Jiang, Y., Wang, P., Fan, Z., Wang, Y., Wang, Z.: NeuralLift-360: Lifting an In-the-Wild 2D Photo to a 3D Object With 360deg Views. In: CVPR (2023) 3
53. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Point-NeRF: Point-based Neural Radiance Fields. In: CVPR (2022) 1, 3, 4, 6, 7, 8, 9, 10, 11, 12
54. Yu, A., Ye, V., Tancik, M., Kanazawa, A.: pixelNeRF: Neural Radiance Fields from One or Few Images. In: CVPR (2021) 1, 3
55. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In: CVPR (2018) 7
56. Zhang, X., Kundu, A., Funkhouser, T., Guibas, L., Su, H., Genova, K.: Nerflets: Local Radiance Fields for Efficient Structure-aware 3d Scene Representation from 2d supervision. In: CVPR (2023) 9, 10
57. Zhang, Y., Huang, X., Ni, B., Li, T., Zhang, W.: Frequency-Modulated Point Cloud Rendering with Easy Editing. In: CVPR (2023) 12

Supplementary Material

We detail mutl-scale point generation and coordinate system of the global triplane. Furthermore, in the provided https://pointnerfpp.github.io, we provide more rendering results.

A Multi-scale Point Generation via Grid-subsampling

We build multi-scale points from an input point cloud using grid subsampling, which is more robust to varying density as shown in KPConv [47]. Specifically, a new support point at each scale is the barycenter of the original input points contained in a grid cell. Thereby, we control the scale and density at each level via the grid size of cells.

We set the grid size at the level s as $\omega * \gamma^{s-1}$ where ω is the initial grid size at the first level and γ is the stride size. We use the larger grid size for severely incomplete point clouds and a small grid size for the complete point cloud. Specifically, for KITTI-360 [27], we set ω as 8cm and γ as 2.92. As a result, the grid size at the coarsest point level (i.e., s=4) is 2 meters. For ScanNet [9], we set ω as 0.008 and γ as 2.0. For Nerf Synthetic [29] where point clouds are relatively complete, we set ω as 0.004 and γ as 1.6.

B The Coordinate System of Global Triplane

We align world coordinate system and the normalized coordinate system of global triplane. We utilize the principal component analysis (PCA) to calculate the reference coordinate frame of input point cloud. The resultant reference frame consists of rotation, translation and scale, thereby defining the alignment matrix transforming world coordinates to coordinate system of global triplane. In ScanNet [9] and Nerf Synthetic [29] where points distribute uniformly along three axes and center at the origin, we simply normalize world coordinates using the scale part of reference frame. For KITTI 360 [27], we use full reference frame instead – i.e., we rotate, translate and scale the world coordinates – because, in this dataset, the car moves along one major direction, leading to the points heavily unbalanced along three axes. With this PCA-based canonicalization, we compactly compress all possible query points into triplane's normalized frame, allowing for fully utilizing the capacity of global triplane.

C More rendering results

We furthermore provide more rendering results – rendering more frames and forming videos. For more details, please refer to https://pointnerfpp.github.io.