Neural Radiance Fields with LiDAR Maps

Ming-Fang Chang[1] Akash Sharma[1] Michael Kaess[1] Simon Lucey[2]
[1]Carnegie Mellon University [2]The University of Adelaide

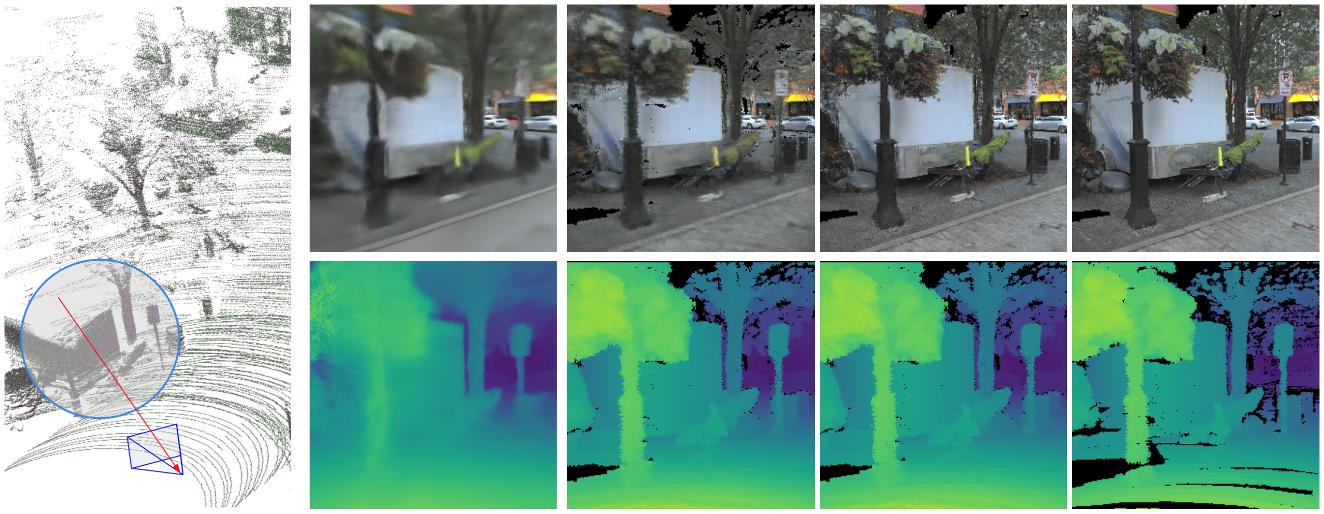


Figure 1: We design a novel view synthesis system from outdoor camera-LiDAR datasets with a point-based NeRF framework and 2D conditional GANs. (a) A LiDAR map (gray points) and queried novel view (blue). (b)-(e) Our method outperforms previous BlockNeRF (with LiDAR depth supervision) [30, 38] and point-based NeRF [44] on Argoverse 2 dataset [42].

Abstract

We address outdoor Neural Radiance Fields (NeRF) [23] with real-world camera views and LiDAR maps. Existing methods usually require densely-sampled source views and do not perform well with the open source camera-LiDAR datasets. In this paper, our design leverages 1) LiDAR sensors for strong 3D geometry priors that significantly improve the ray sampling locality, and 2) Conditional Adversarial Networks (cGANs) [15] to recover image details since aggregating embeddings from imperfect LiDAR maps causes artifacts. Our experiments show that while NeRF baselines produce either noisy or blurry results on Argoverse 2 [42], our system not only outperforms baselines in image quality metrics under both clean and noisy conditions, but also obtains closer Detectron2 [43] results to the ground truth images. Furthermore, this system can be used in data augmentation for training a pose regression network [3] and multi-season view synthesis. We hope this work to serve as a new LiDAR-based NeRF baseline that pushes this research direction forward (released here).

1. Introduction

Despite the fact that recent works have made massive improvements in novel view synthesis (NVS) for small scenes [2, 10, 17, 21, 23, 25, 28], large-scale outdoor scenes – such as street views and parks – are still challenging. Improving the NeRF results on large-scale outdoor scenes would greatly benefit multiple applications, such as realistic simulators for robot navigation [1], localization [20], active mapping and planning [46], and novel view augmentation [24].

The transition from indoor to outdoor presents a non-trivial challenge. Typically, the training of NeRF models demands densely sampled views to achieve good accuracy [7]. However, collecting densely sampled training views in outdoor scenarios requires much labor and storage space, and the camera trajectories in outdoor settings are typically biased (straight and along the lane). Many parts of the scene are often only observed by a limited number of views and range of view angles. This lack of data coverage issue of common outdoor datasets [9, 11, 42] is also addressed by previous works [30, 38], where specially collected datasets instead of common public outdoor datasets were used. Furthermore, previous methods using simple MLPs to represent large blocks tend to generate blurry results (Fig. 1 (b)). On the other hand, it has been demonstrated that the demand of dense training views can be effectively reduced by geometry priors [7, 41], and modern outdoor robots – such as autonomous vehicles – are often equipped with LiDAR sensors in addition to cameras. In contrast to previous LiDAR-assisted works [5,30] that used LiDAR scans only as supervision or as a guidance for ray sampling, our approach treats the LiDAR map as sparse samples of the environment and directly distributes localized embeddings on it.

Using localized embeddings instead of global representation can ease the burden of memorizing the whole scene with a single MLP in NeRF, leading to better embedding locality and convergence speed [13, 19, 44]. PointNeRF [44] embedded per-point features to 3D point clouds (from CNN prediction or COLMAP [34, 35]) and aggregated these point cloud embeddings along ray samples for volume rendering. However, the method in [44] is not directly applicable to the common camera-LiDAR datasets we target. The real-world LiDAR maps are prone to noise due to imperfect conditions such as bad weather. Simply using the point-based NeRF on outdoor LiDAR maps leads to noisy and unsatisfactory image quality, as in Fig. 1 (c). The 3D point cloud refinement techniques proposed in [44] requires the photometric constraints from the texture of dense training views, and thus are not suitable for the outdoor datasets where the views are sparser and the scenes are more complex. Instead of 3D point cloud refinement as [44], we propose to leverage strong 2D image refinement in this work.

The proposed pipeline takes a neural 3D point cloud (i.e. the LiDAR map with embeddings) as input and aggregates the LiDAR embeddings to perform volume rendering. In addition, we propose a tight sampling strategy in contrast to the naive radius-based counterpart in [44] to make samples better align with the LiDAR geometry prior. Finally, we refine the quality of synthesized views in 2D with a conditional GAN (cGAN) [15]. The proposed cGAN module can be trained end-to-end without additional data, and largely improves the final image quality. Besides common image metrics, we also show that the Detectron2 [43] detection results from our rendered images are closer to the results from ground truth images than the baselines. Last but not least, the proposed system can serve several interesting applications, including data augmentation for training a pose regression network [3] and seasonal appearance rendering.

In summary, our pipeline amalgamates the strengths of neural radiance field (for deep implicit 3D representation), LiDAR maps (for geometric priors), and cGAN (for deep realistic appearance rendering). The demonstrated applications also show foreseeable potential of our system to benefit tasks that require a richer neural map representation.

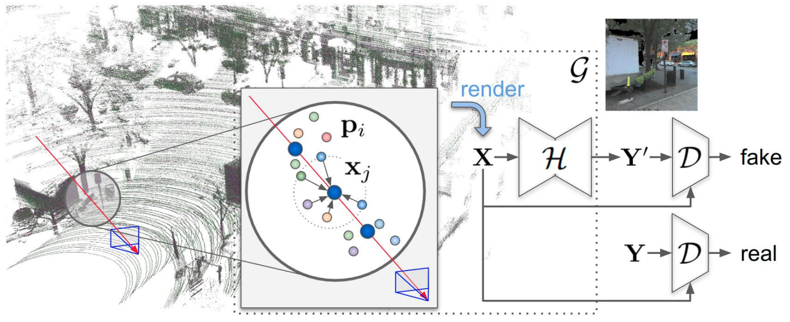
2. Related Works

Large-scale NeRFs Neural Radiance Field (NeRF) [23] is an implicit neural representation trained by overfitting an MLP network to a set of posed 2D images, and can be used to render novel views from complex 3D scenes. The MLP takes a camera view direction and a 3D position as input and predicts the corresponding color and density. When given a novel camera pose and intrinsics, a NeRF system draws rays from the query camera center through its virtual image pixel positions into 3D space, sampling 3D points along the rays, and accumulates the predicted color and density of the 3D sample points for each ray to obtain color values for each pixel. Given proper training data, a NeRF system can render high-quality synthetic images with realistic visual appearance and reasonable depth [2, 10, 17, 21, 23, 25, 28]. Here, we focus on large-scale outdoor environments. Since it is inefficient to use a global MLP [23] to encode a large space, existing work leverages the divide-and-conquer approach – dividing the space into small parts such as street blocks [38] or voxels [13,19,29,45], and assigning localized embeddings to represent the small parts.

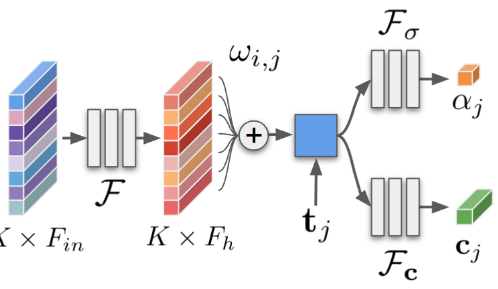
Existing methods have also shown that using depth priors can significantly reduce the required number of source view images for NeRF [7, 17, 31]. Comparing to predicted depth [17] and SfM point clouds [7], LiDAR measurements are more robust and can better cover the geometry of textureless regions where depth values are not well-constrained by photometric information. Rematas et al. proposed using LiDAR depth as supervision [30], and Carlson et al. proposed using trainable occupancy grid to assist ray sampling locality [5], but both [30] and [5] still use global MLPs. On the other hand, PointNeRF [44] proposed a point cloud based neural radiance field with localized embeddings, and greatly improved NeRF sampling locality and convergence speed. NPLF [27] aggregated point features into ray features with self-attention mechanism. In this work, we look into ways to improve [44] for challenging outdoor datasets and explore more practical applications.

Generative Adversarial Networks Generative Adversarial Networks (GANs) have been applied to support NeRF in different ways [6, 16, 22, 26, 36]. For example, generative models were used to represent individual objects that could be combined into a full image by controlling object positions [26]. However, in general outdoor scenes, many objects are not labeled and thus cannot be easily segmented and represented with individual generative models. Previous works also attempted to perform novel view synthesis with 3D-aware GANs [8, 16, 22, 36], while existing 3D-aware GANs are limited to simple geometry such as small objects or faces and cannot be directly applied to general scenes.

On the other hand, 2D conditional GANs can learn the image translation between two distributions and produce visually realistic appearance [15, 47]. In contrast to vanilla



(a) System pipeline



(b) Network structure

Figure 2: (a) We perform spatial interpolation to aggregate the LiDAR map embeddings (smaller dots $\mathbf{p}_i$) onto volume rendering ray samples (larger blue dots $\mathbf{x}_j$). A cGAN is used to refine the volume rendering output $\mathbf{X}$, where the generator $\mathcal{G}$ contains volume rendering parameters and a CNN $\mathcal{H}$ that translates the volume rendering output image $\mathbf{X}$ to a refined image $\mathbf{Y}'$. The discriminator $\mathcal{D}$ aims to predict real or fake based on feeding the ground truth image $\mathbf{Y}$ or the generated image $\mathbf{Y}'$. (b) We process the per-LiDAR point information with an MLP $\mathcal{F}$ and aggregate the processed embeddings by spatial interpolation with weights $\omega_{i,j}$. View-based appearance embeddings $\mathbf{t}_j$ can be incorporated (blue block). Finally we predict sample color $\mathbf{c}_j$ and density α_j with the other two MLPs $\mathcal{F}_\sigma$ and $\mathcal{F}_\mathbf{c}$.

GANs that demands many training data, cGANs benefit from the conditional input and can be trained on much fewer data, such as images captured by sonar and tactile sensors [18, 37]. GANcraft [13] applied a cGAN to translate semantic segmentation images into realistic images. In our case, the dense semantic labels are not available, but we also leverage a 2D cGAN to refine the volume rendering output.

3. Method

Given a camera-LiDAR data sequence with known poses, we first build a LiDAR map $\mathcal{P}$ by accumulating the LiDAR scans, and then assign trainable per-point embeddings to the LiDAR map as our localized neural radiance field representation (Fig. 2). These neural LiDAR points represent sparse samples of the underlying neural radiance field.

For a queried novel view pose, we perform volume rendering that interpolates and aggregates LiDAR point embeddings to generate pixel colors (Sec. 3.1). This volume rendering step generates an initial image $\mathbf{X}$, which is refined with a cGAN to generate a final image $\mathbf{Y}'$ (Sec. 3.3). We train the whole pipeline in an end-to-end fashion with supervision from training source views $\mathbf{Y}$, conditional adversarial losses (Sec. 3.3), and LiDAR map geometry (Sec. 3.4).

3.1. Point-based Volume Rendering

We follow the conventional volume rendering method and compute the per-pixel radiance via ray marching [23, 44]. First, a ray is drawn from the camera center to the pixel center in 3D space, and M ray sample positions $\{\mathbf{x}_j \in \mathbb{R}^3 | j = 1, \dots, M\}$ are selected along the ray. Afterwards, the sample color $\mathbf{c}_j$, and density σ_j are computed at each of the M points. Finally, samples along the ray are accumulated to compute the pixel color $\mathbf{C}$:

$$\begin{aligned} \mathbf{C} &= \sum_{j=1}^{M} \tau_j (1 - \exp(-\sigma_j \delta_j)) \mathbf{c}_j, \\ \tau_j &= \exp(-\sum_{t=1}^{j-1} \sigma_t \delta_t), \end{aligned} \tag{1}$$

where δ_j and δ_t are the intervals between adjacent ray samples, and τ_j is the transmittance accumulated from the density of the ray samples between $\mathbf{x}_j$ and the camera center.

Point sampling with priors. A main challenge in large-scale volume rendering is to determine the positions of ray samples $\mathbf{x}_j$. Ideally, $\mathbf{x}_j$ should cover the potentially occupied regions, but in practical systems, the number of samples and their coverage is limited. The geometry prior from LiDAR maps provides important guidance for sampling at the occupied locations and skipping the large empty space.

Specifically, we first uniformly sample M_0 positions along a camera ray and compute the distances ρ from each sample to its nearest LiDAR point. Among the samples with $\rho < \zeta$, where ζ is an assigned radius threshold, we select the first M samples that are closest to the camera center. To collect local information from our point-based neural radiance field, we query the k-nearest LiDAR point neighbor set ϕ_j^{kNN} for the M selected ray samples $\mathbf{x}_j$:

$$\phi_j^{\text{kNN}} = \{\mathbf{p}_i \in \mathbb{R}^3 | \, ||\mathbf{p}_i - \mathbf{x}_j|| < \zeta \text{ and } \mathbf{p}_i \in \mathcal{P}\}. \tag{2}$$

In practice, k varies among samples but is upper-bounded by an assigned parameter K. The radius threshold ζ depends on the noise and density of the LiDAR map.

The above radius-based selection simply dilates the 3D extent of LiDAR point distribution. However, for datasets with good LiDAR quality (such as autonomous driving datasets), the LiDAR points are usually tightly distributed near object boundary, and we only need to fill the holes without dilation. The dilated regions are most likely to be empty, and we would place unnecessary ray samples in those empty regions (Fig. 3 (b2)) if using naive radius based selection like [44].

To solve this issue, we propose to trust the LiDAR geometry more by tightening the extent of $\mathbf{x}_j$. This can be achieved by removing the $\mathbf{x}_j$s that are not surrounded by the LiDAR points in ϕ_j^{kNN} as in Fig. 3 (a1)(a2). Let $\mathbf{p}_0$ be the nearest LiDAR point in ϕ_j^{kNN}, we discard the sample $\mathbf{x}_j$ if

$$(\mathbf{x}_j - \mathbf{p}_0) \cdot (\mathbf{x}_j - \mathbf{p}_i) > 0,\ \forall i \in \{1, \ldots, k\}. \tag{3}$$

A visualization of the tightened ray samples is in Fig. 3 (b3). A quantitative comparison is in the supplementary material.

Feature aggregation. After collecting $\mathbf{p}_i \in \phi_j^{\text{kNN}}$, we aggregate their map embeddings to predict the color and density for sample $\mathbf{x}_j$. Specifically, we first concatenate the LiDAR point embeddings $\mathbf{f}_i \in \mathbb{R}^{F_m}$ with spatial offsets $\mathbf{o}_{i,j} = \mathbf{p}_i - \mathbf{x}_j$, and then pass the concatenated embeddings through a light-weight MLP $\mathcal{F}$ to obtain processed embeddings $\mathbf{f}'_{i,j}$. Afterwards, we aggregate $\mathbf{f}'_{i,j}$ onto $\mathbf{x}_j$ via spatial interpolation to obtain the sample embedding $\mathbf{h}_j \in \mathbb{R}^{F_h}$:

$$\begin{aligned}\mathbf{f}'_{i,j} &= \mathcal{F}(\text{concat}(\mathbf{f}_i, \gamma(\mathbf{o}_{i,j}))),\\ \mathbf{h}_j &= \frac{\sum_{i=1}^{k} \omega_{i,j}\mathbf{f}'_{i,j}}{\sum_{i=1}^{k} \omega_{i,j}},\end{aligned} \tag{4}$$

where $\gamma(.)$ represents the positional encoding function. Using the spatial offset $\mathbf{o}_{i,j} \in \mathbb{R}^3$ instead of simple distance makes $\mathbf{f}'_{i,j}$ anisotropic and richer. The weights $\omega_{i,j}$ are designed to favor embeddings from closer LiDAR points:

$$\omega_{i,j} = \exp(-\beta||\mathbf{p}_i - \mathbf{x}_j||). \tag{5}$$

The choice of weighting function should depend on the LiDAR sensor and map characteristics. In our case, LiDAR points are locally dense around the object surface, and we found Eq. 5 works well in our experiments. Finally, the per-sample color and density, $\mathbf{c}_j$ and α_j, are predicted from $\mathbf{h}_j$ with two other MLPs, $\mathcal{F}_\alpha$ and $\mathcal{F}_\mathbf{c}$, as shown in Fig. 2 (b).

We also make Lambertian assumption and discard view direction dependency since modeling reflective surfaces is not our focus (see Sec. 6). Although one can also include view direction as in [38,44] or model the lighting for specific objects with object shape priors as in [39], we found this setup suffice the applications we explored (Sec. 5).

3.2. Additional Appearance Embeddings

Additional latent variables can be incorporated to manipulate synthesized image appearance. Here we use view-based embeddings $\mathbf{t}_j$ (Fig. 2 (b)). A season change demonstration with timestamps as $\mathbf{t}_j$ is presented in Sec. 5.

3.3. Image Refinement with cGAN

Conditional Adversarial Networks (cGANs) have been known for the ability to perform image translation that generates visually pleasant details [13, 15, 47], and is trainable from small datasets [18, 37]. Overall, we follow the pix2pix framework [15]. For a training pair $(\mathbf{Y}, \theta)$ consisting of a training source view image and its pose, we perform volume rendering from the neural LiDAR map to generate an initial image $\mathbf{X}$ from θ, and use a CNN $\mathcal{H}$ (Fig. 2 (a)) to translate $\mathbf{X}$ into a realistic domain image $\mathbf{Y}' = \mathcal{H}(\mathbf{X}, z)$, where z is the introduced noise vector described in [15]. We train the whole pipeline in an end-to-end fashion, so our generator $\mathcal{G}$ not only contains parameters from $\mathcal{H}$, but also the volume rendering MLPs, $\mathcal{F}, \mathcal{F}_\sigma, \mathcal{F}_\mathbf{c}$, and the neural map point embeddings $\mathbf{f}_i$ (Sec. 3.1). A discriminator $\mathcal{D}$ takes the concatenation of $\mathbf{X}$ and $\mathbf{Y}'$, $\mathbf{X}$ and $\mathbf{Y}$ as input, and predicts high scores for $\mathbf{Y}$ and low scores for $\mathbf{Y}'$. The cGAN loss contains an adversarial term and a $\mathcal{L}_1$ norm term:

$$\begin{aligned}\mathcal{L}_{\text{cGAN}} &= \mathbb{E}_{x,y}[\log \mathcal{D}(\mathbf{X}, \mathbf{Y})] + \mathbb{E}_{x,z}[\log(1 - \mathcal{D}(\mathbf{X}, \mathbf{Y}')],\\ \mathcal{L}_1 &= ||\mathbf{Y} - \mathbf{Y}'||_1.\end{aligned} \tag{6}$$

Finally, we solve the following optimization problem:

$$\arg\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{L}_{\text{cGAN}}(\mathcal{G}, \mathcal{D}) + \lambda \mathcal{L}_1(\mathcal{G}). \tag{7}$$

In practice, we discard the noise vector z to obtain consistent image outputs for natural video rendering. The same strategy is also used in CycleGAN [47]. The application of CNN $\mathcal{H}$ also enables us to slightly dilate the RGB coverage on the cGAN output (see the difference in black area in Fig. 4 (d) and (e)). We use a 6-layer autoencoder with three ResNet blocks in the middle as $\mathcal{H}$. Interestingly, we can further control the output image style of cGAN with the strength of $\mathcal{L}_{\text{cGAN}}$. Stronger $\mathcal{L}_{\text{cGAN}}$ adds more fine details to $\mathbf{Y}'$, leading to lower PSNR but higher LPIPS. We present ablation study for $\mathcal{L}_{\text{cGAN}}$ strength in the supplementary material. This system can be trained with only the log sequence ($\sim$155 images (Sec. 4)) without additional data.

3.4. LiDAR Depth Loss

We introduce this LiDAR depth loss to constrain the estimated depth to be close to the LiDAR measurements.

To achieve this, we render pseudo ground truth depth from the LiDAR geometry as supervision. We assume the LiDAR measurements form a thin layer of opaque material on object surfaces to render the pseudo ground truth depth image, denoted as $\mathbf{D}^l$. Specifically, for each ray sample $\mathbf{x}_j$, if its distance to the closest LiDAR point is less than an assigned threshold μ, we consider it opaque (i.e. $\exp(-\sigma_j\delta_j) = 0$ in Eq. 1), otherwise transparent ($\sigma_j = 0$). Next, we accumulate the samples to obtain the pseudo ground truth depth image $\mathbf{D}^l$ with Eq. 1. Let $\mathbf{D}$ be the depth map output from Sec. 3.1,

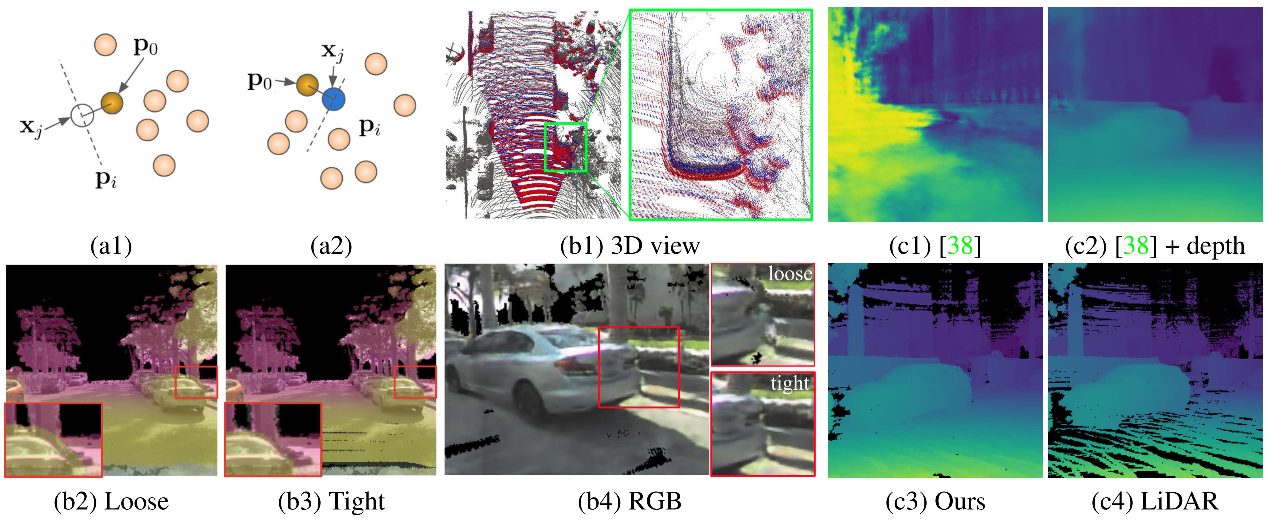


Figure 3: (a) Examples of the proposed tight sampling strategy. The LiDAR points $\mathbf{p}_i$ in ϕ_j^{kNN} are in light orange, and the nearest LiDAR point $\mathbf{p}_0$ is in dark orange. The white sample $\mathbf{x}_j$ in (a1) is discarded because all the $\mathbf{p}_i$s are in the same side as $\mathbf{p}_0$, meaning $\mathbf{x}_j$ is not surrounded by LiDAR points. The blue sample in (a2) remains because its surrounded by LiDAR points. (b1) The LiDAR map points are in gray, and the loose and tight ray samples are in red and blue. (b2)(b3) Overlaid initial depth projection and the rgb of loose and tight sampling strategies. (b4) Artifact around object boundary due to mismatch between depth and rgb (upper right). The object boundary from the tight sampling strategy (lower right) is better aligned with rgb. (c) The proposed depth loss pulls rendered depth in (c2)(c3) closer to the LiDAR depth (c4).

our LiDAR depth loss is:

$$\mathcal{L}_l = \frac{1}{|\phi_{\text{valid}}|} \sum_{n \in \phi_{\text{valid}}} |D_n - D_n^l|, \\ \phi_{\text{valid}} = \{n | D_n^l > 0\}, \tag{8}$$

where ϕ_{valid} is the set of pixels with valid LiDAR depth.

A potential alternative way is to apply an additional loss term that enforces σ_j of the ray samples close to LiDAR points to be large. However, the LiDAR sensor only returns measurements on object surfaces. Ray samples inside objects might be far away from LiDAR points but still have high density, and thus their density cannot be constrained by the nearest distance to LiDAR points. In our case, the ray samples inside objects are not constrained because they are occluded by the opaque samples on the object surfaces (close to LiDAR points) along the same ray.

3.5. Moving Object Removal

Our focus is to render static scenes, and the neural LiDAR maps are also assumed static. The dynamic objects captured by training views and LiDAR scans would introduce unwanted inconsistent appearance information and cause blurry shadows in the rendered images. To overcome this issue, we use the 3D semantic labels [42] to filter out dynamic objects. We first extract the provided 3D bounding boxes of dynamic objects by thresholding the trajectory length of each labeled object. Afterwards, we extract the LiDAR points within the bounding boxes of dynamic objects from each LiDAR scan. For each training image, we project the extracted dynamic LiDAR points from its temporally nearest LiDAR scans onto the image space to form a binary mask, which indicates the pixels occupied by the dynamic objects. We also removed the dynamic LiDAR points from our LiDAR maps. Finally, we apply the mask to Eq. 6.

4. Experiments

4.1. Datasets

We extracted 8 sequences from the Argoverse 2 dataset [42]. Each sequence contains 3 cameras (front, front-left, front-right). The sequences were selected to avoid crowded scenes and scenes with large number of moving objects. The image contents are mostly city scenes with different ratio of artificial buildings and natural textures. We accumulated and resampled the LiDAR scans of the training images with provided ground truth poses to form the LiDAR maps. The LiDAR scans were collected at 10 Hz by two VLP-32C LiDARs with 64 beams in total. Since the original Argoverse 2 dataset only provides 10 Hz LiDAR poses and 20 Hz imagery without assigned poses, we first extracted the 10 Hz posed imagery by using the temporally closest LiDAR poses as image poses to reconstruct a visual map. And then we registered the other half of unposed images to the visual map with COLMAP to obtain totally 20 Hz image poses that match the LiDAR pose scale. We subsampled one in every four images from the original 10Hz posed imagery for validation, and added the images with COLMAP poses to the training set. In total we have 155 training images and 22 validation images for each sequence. The number of points in our LiDAR maps range from 3.0×10^5 to 5.8×10^5 and the camera trajectory lengths span from $6.49m$ to $31.6m$ (details in the supplementary material).

Note that we collected images from 3 front-facing cameras in contrast to the 12 ring camera setting in [38], and our LiDAR point cloud is much sparser than [30]'s. These makes our dataset more challenging than previous works.

4.2. Baselines, Metrics, and Implementation

We compared with the point-based NeRF [44] and the outdoor state-of-the-art BlockNeRF [38].

Point-based NeRF. We compared our final results with our volume rendering pipeline output, which is based on [44], but with view direction dependency and point cloud refinement module removed. We kept the rest of the pipeline (e.g. point sampling, weighting functions) the same as our system except for the refinement module for fair comparison. We applied the same LiDAR depth and $\mathcal{L}_1$ losses to the point-based NeRF output. Each model was trained with Adam optimizer using one whole image as one batch and learning rate 1×10^{-4} for 1000 epochs until convergence.

BlockNeRF [38] + LiDAR [30]. For the image-only baseline, we compared with [38], the state-of-the-art outdoor large-scale NeRF. The longest camera trajectory of our collected Argoverse 2 sequences ($31.6m$) is within a reasonable range of block size in [38], so we used one block for each sequence. Furthermore, we also compared with the BlockNeRF with LiDAR depth supervision [30] for fair comparison. Because the original code of both [38] and [30] are not available, we ran experiments with an unofficial BlockNeRF implementation, and incorporated the depth loss as described in URF [30]. Each model was trained with Adam optimizer with learning rate 5×10^{-4} and batch size 1024 for 500 epochs, where the validation error converged.

Metrics. We compared PSNR, SSIM, LPIPS as in [23]. We masked out the regions without LiDAR depth (the black regions in the volume rendering outout) since our pipeline does not focus on generating those pixels (Sec. 6). The same masks were applied to all the compared results when computing the metrics for fair comparison. Otherwise the evaluation result would be diluted by the large black regions. Note that our numbers are not directly comparable to the numbers reported by previous works due to these masks.

Implementation details. We first built a k-d tree from our LiDAR maps, and queried the nearby LiDAR points to ray samples with the k-d tree. After obtaining the set of ray samples and kNN LiDAR points, we implemented our volume rendering with DGL [40] and PyTorch. A heterogeneous local graph was built with pixels, ray samples and LiDAR points as nodes. Different types of edges connect pixels to the corresponding ray samples, and ray samples to the nearby LiDAR points. The MLPs and the aggregation function were implemented as graph functions. We used $K = 10$, $M_0 = 1000$, $M = 16$, $\zeta = 0.15m$, $\beta = 10$, $\mu = 0.05m$, $\lambda = 100$, $F_m = 8$, and $F_h = 32$ as design parameters. The numbers of MLP layers in $\mathcal{F}$, $\mathcal{F}_\sigma$, and $\mathcal{F}_\mathbf{c}$ are all 3. The parameters and network structures here are tuned with clean Argoverse 2 sequences and applied to all the datasets. Our current system takes about $5s$ to render an image. For each sequence, we trained independent models with the whole pipeline in an end-to-end fashion for 1000 epochs with Adam optimizer, using one whole image as a batch and learning rate 1×10^{-4}. It takes about 0.5 days to train a model on a single GeForce RTX 3090 GPU at image resolution of 256×256.

4.3. Comparison with Baselines

We observed that the proposed 2D refinement strategy outperformed point-based NeRF by a large margin in every sequence tested, as shown in Fig. 5. From the rendered images in Fig. 4 (d)(e), we observed significant image quality improvement with the proposed image refinement stage. The refinement module not only reduced the noise but also rendered details better. Besides, we obtained very blurry results from BlockNeRF even with LiDAR depth supervision, as shown by the metrics in Fig. 5 and the visualizations in Fig. 4 (a)(b). Incorporating the LiDAR depth supervision significantly improved the depth map quality from BlockNeRF (Fig. 3 (c2)) but the rgb result is still blurry. Although the rendered image area of BlockNeRF is not as limited as ours (e.g. the sky), it would require specially collected datasets with better view coverage to improve this result. This drawback of image-only NeRF was also pointed out by [7, 30, 31].

4.4. Resistance to Noise

Although the LiDAR measurements are relatively more accurate than SfM, they can degrade greatly in bad weather conditions. We simulated such LiDAR noise in rainy days [12] before accumulating single LiDAR scans into LiDAR maps, to evaluate the resistance to noise (visuals in supplementary material). Quantitative evaluation for these harder cases are shown in Fig. 5. We observed that our method still outperformed the purely point-based baseline in the evaluated metrics and also recovered reasonable visual details as shown in Fig. 6 (d).

We also provide additional ablation study in the supplementary material, including the quantitative comparison of the sampling strategy, cGAN loss strength, and the positional encoding module.

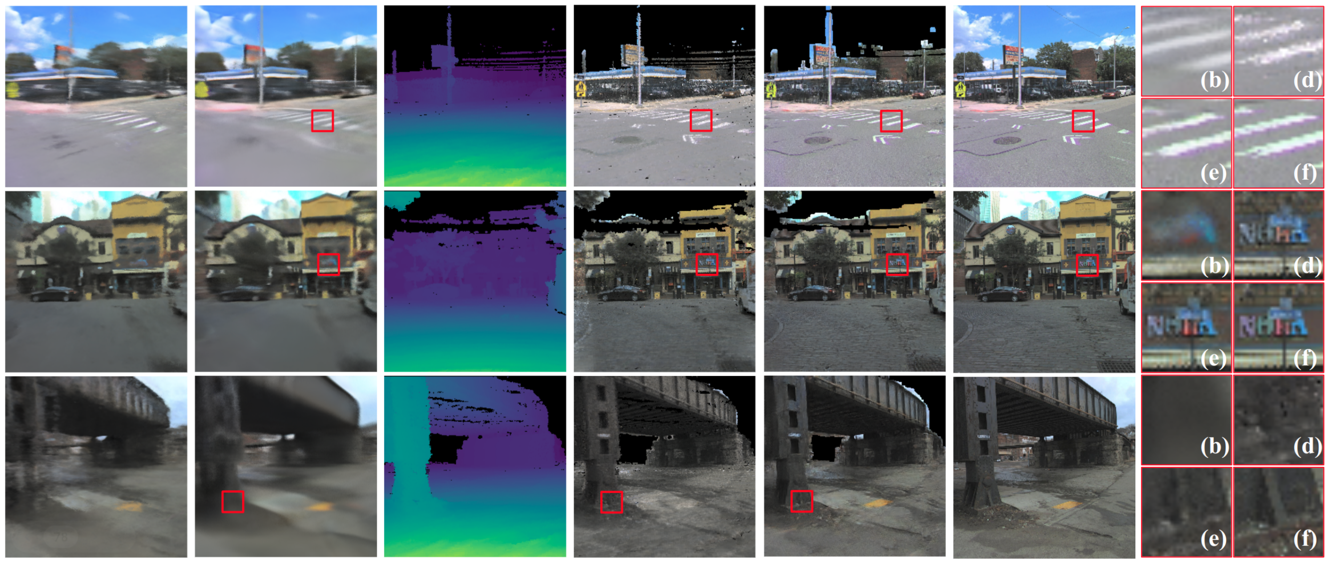


Figure 4: From the enlarged image patches (g) we can observe that our results (e) have better image quality and are visually closer to the ground truth patches (f). More visualizations are in the supplementary material.

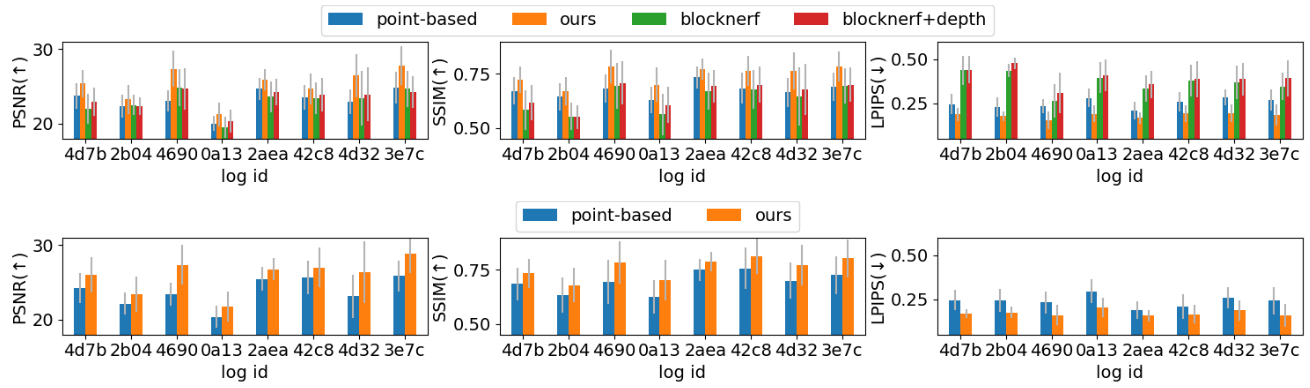


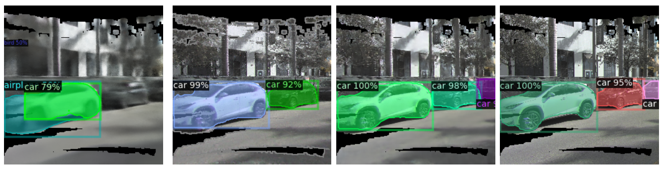
Figure 5: Quantitative comparisons. (Top) our method significantly outperformed the baselines. The BlockNeRF results are very blurry, as reflected by the high LPIPS scores. (Bottom) results with noisy LiDAR maps.

5. Applications

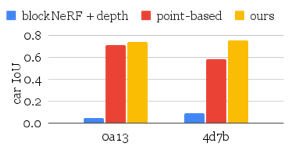
Although many NeRF works have been published since 2020, most of the advancements were for improving image quality. Recently, more and more researchers start to explore wider uses such as robot navigation and SLAM [1, 32]. This trend implies the great yet not well-explored potential of using NeRF in more applications. From this application-oriented standpoint, we should evaluate the NeRF outputs not only based on image quality, but also the performance in targeted applications, as it is what matters in the end.

Object detection simulator. We compared the detection results on our synthetic images and the ground truth images using Detectron2 [43]. Considering that the car class usually dominates autonomous driving applications, we performed this comparison on two logs with more visible vehicles (log 0a13 and 4d7b). Our results show that the detected object masks are visually similar on synthetic and ground truth images, and our mean IoU for the car class among the validation images outperformed the baselines (Fig. 6 (a)). This result indicates that the synthetic images rendered by our system can potentially be used to further simulate the detection-related robot behaviors, such as object tracking when robot is moving, or path-planning considering the existence of other objects.

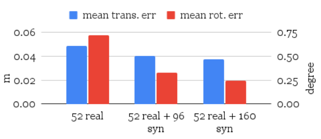
Data augmentation. Data collection is essential for many deep neural network applications. However, collecting real-world data can be an expensive and time-consuming process, because it typically requires driving a mobile robot across the desired environment for multiple passes. One possible way to lighten the burden is to first train a NeRF with some collected images, and augment the dataset with syn-



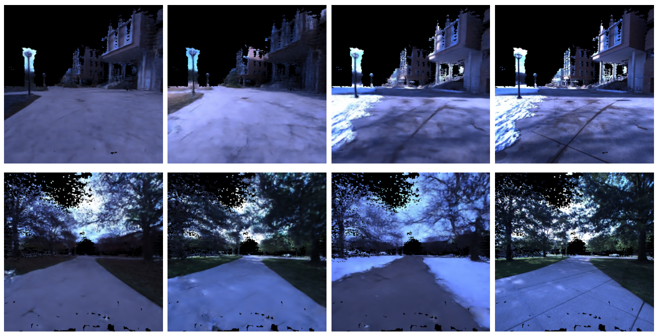
(a1) [38]+depth (a2) [44] (a3) Ours (a4) GT



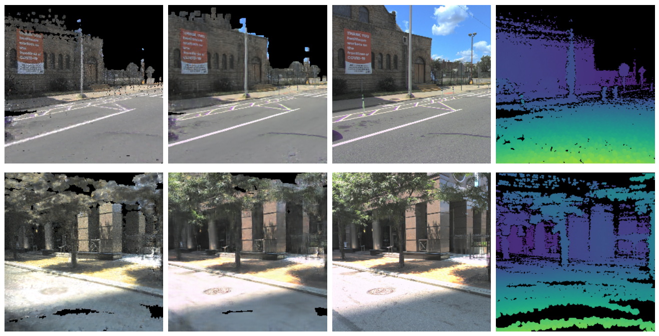
(a5) Car IoU



(b) Pose prediction errors



(c1) Season 1 (c2) Season 2 (c3) Season 3 (c4) GT



(d1) [44] (d2) Ours (d3) GT (d4) LiDAR

Figure 6: (a1-4) Object mask from [43]. (a5) Our rendered images get higher car IoU against GT than the baselines. (b) Data augmentation with our synthetic images significantly reduced the pose prediction errors of MapNet [3]. (c1-4) Synthesized images and GT with different appearance embedding t rendered from the same validation camera pose. (d1-4) Rendered images, ground truth rgb and depth from the noisy LiDAR map.

thetic images rendered by NeRF. This is especially useful for data-hungry deep networks such as pose regression networks, whose performance largely depend on the training set size [33]. Inspired by [24], we augmented our collected Argoverse 2 sequences (Sec. 4.1), and compared the MapNet [3] performance with the same real validation set before and after the augmentation. The MapNet network takes a RGB image as input and predicts a 6-DoF camera pose. For augmentation, we added uniformly random noises within range $[-0.5, +0.5]m$ in xz for translation, and $[-3, +3]$ degree in yaw for rotation. The results in Fig. 6 (b) show significantly lower pose errors in the cases with data augmentation, indicating that our synthetic images provide useful information for MapNet despite the black regions.

Changing seasons. If trained with seasonal information, the proposed system can perform season changing visual effect. With this ability, we can potentially use our system as a simulator with the control of seasonal appearance. We performed this experiment with the NCLT dataset [4], a long-term camera-LiDAR dataset that was collected through different seasons. We extracted multi-season sequences from NCLT, each with multiple passes of similar trajectories in different seasons. For the appearance embedding, we used a scalar $t \in [0, 1]$ to represent the normalized timestamp when the data was recorded (e.g. $t = 0.33, 0.66, 1$ means spring, summer and winter). We first encoded t with positional encoding, and used the encoded t, $\mathbf{t}_j$ with an AdaIN [14] module to incorporate $\mathbf{t}_j$ into $\mathbf{h}_j$ (the blue box in Fig. 2 (b)). After training, we applied different t values to a given validation view pose to obtain different seasonal appearances (Fig. 6 (c)). More details are in the supplementary.

6. Limitations

Our pipeline skips and returns black color at the pixels where the LiDAR depth is unavailable. The same limitation was presented in PointNeRF [44]. The black regions are mostly on the sky, top of tall buildings, and objects too far away. Another current limitation is that non-lambertian surfaces, such as transparent and reflective objects, are currently unmodeled. This can potentially be improved by introducing latent variables to model these materials in the volume rendering MLPs. For now our implementation relies on accurate camera-LiDAR calibration, therefore optimizing the calibration parameters with NeRF can be a future work. Also, here we focus on offline image rendering quality but runtime still has room to improve. For example, we expect the point embeddings to be repetitive and can be pruned and compressed. Besides, large moving objects with inaccurate 3D labels might cause bad results. Visualizations of failure cases are in the supplementary material.

7. Conclusion

The proposed pipeline performs NVS from outdoor camera-LiDAR datasets. Our system combines the strength of LiDAR sensors, NeRF, and cGAN, and outperforms the baselines significantly. We believe that the practical use of NeRF is an uprising research area, and look forward to future developments in this field.

References

[1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-Only Robot Navigation in a Neural Radiance World. In *arXiv*, 2021. 1, 7

[2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *Int. Conf. Comput. Vis.*, 2021. 1, 2

[3] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-Aware Learning of Maps for Camera Localization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 1, 2, 8

[4] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus Long-Term Vision and Lidar Dataset. In *Int. J. of Robotics Res.*, 2016. 8

[5] Alexandra Carlson, Manikandasriram Srinivasan Ramanagopal, Nathan Tseng, Matthew Johnson-Roberson, Ram Vasudevan, and Katherine A. Skinner. CLONeR: Camera-Lidar Fusion for Occupancy Grid-aided Neural Representations. In *IEEE Robotics and Automation Letters*, 2023. 2

[6] Eric R. Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. Pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2

[7] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer Views and Faster Training for Free. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1, 2, 6

[8] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. GRAM: Generative Radiance Manifolds for 3D-Aware Image Generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

[9] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, Zoey Yang, Aurelien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large Scale Interactive Motion Forecasting for Autonomous Driving : The Waymo Open Motion Dataset. In *Int. Conf. Comput. Vis.*, 2021. 1

[10] Guy Gafni, Justus Thies, Michael Zollhöfer, and Matthias Nießner. Dynamic Neural Radiance Fields for Monocular 4D Facial Avatar Reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 1, 2

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2012. 1

[12] Christopher Goodin, Daniel Carruth, Matthew Doude, and Christopher Hudson. Predicting the Influence of Rain on LIDAR in ADAS. In *Electronics*, 2019. 6

[13] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *Int. Conf. Comput. Vis.*, 2021. 2, 3, 4

[14] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. In *Int. Conf. Comput. Vis.*, 2017. 8

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 1, 2, 4

[16] Jeong-gi Kwak, Yuanming Li, Dongsik Yoon, Donghyeon Kim, David Han, and Hanseok Ko. Injecting 3D Perception of Controllable NeRF-GAN into StyleGAN for Editable Portrait Image Synthesis. In *Eur. Conf. Comput. Vis.*, 2022. 2

[17] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 1, 2

[18] Tianxiang Lin, Akshay Hinduja, Mohamad Qadri, and Michael Kaess. Conditional GANs for Sonar Image Filtering with Applications to Underwater Occupancy Mapping. In *IEEE Int. Conf. Robotics and Automation*, 2023. 3, 4

[19] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. In *Conf. Neural Inform. Process. Syst.*, 2020. 2

[20] Dominic Maggio, Marcus Abate, Jingnan Shi, Courtney Mario, and Luca Carlone. Loc-NeRF: Monte Carlo Localization Using Neural Radiance Fields. In *arXiv*, 2022. 1

[21] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 1, 2

[22] Quan Meng, Anpei Chen, Haimin Luo, Minye Wu, Hao Su, Lan Xu, Xuming He, and Jingyi Yu. GNeRF: GAN-based Neural Radiance Field without Posed Camera. In *Int. Conf. Comput. Vis.*, 2021. 2

[23] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Eur. Conf. Comput. Vis.*, 2020. 1, 2, 3, 6

[24] Arthur Moreau, Nathan Piasco, Dzmitry Tsishkou, Bogdan Stanciulescu, and Arnaud de La Fortelle. LENS: Localization Enhanced by NeRF Synthesis. In *Conf. on Robot Learning*, 2021. 1, 8

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding . In *arXiv*, 2022. 1, 2

[26] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2

[27] Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. Neural Point Light Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

[28] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable Neural Radiance Fields. In *Int. Conf. Comput. Vis.*, 2021. 1, 2

[29] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLP . In *Int. Conf. Comput. Vis.*, 2021. 2

[30] Konstantinos Rematas, Andrew Liu, Pratul Srinivasan, Jonathan Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1, 2, 6

[31] Barbara Roessle, Jonathan T. Barron, Ben Mildenhall, Pratul P. Srinivasan, and Matthias Nießner. Dense Depth Priors for Neural Radiance Fields from Sparse Input Views. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2, 6

[32] Antoni Rosinol, John J. Leonard, and Luca Carlone. NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields. In *arXiv*, 2022. 7

[33] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe´. Understanding the Limitations of CNN-based Absolute Camera Pose Regression. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 8

[34] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016. 2

[35] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Eur. Conf. Comput. Vis.*, 2016. 2

[36] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. In *Conf. Neural Inform. Process. Syst.*, 2020. 2

[37] Paloma Sodhi, Michael Kaess, Mustafa Mukadam, and Stuart Anderson. PatchGraph: In-hand Tactile Tracking with Learned Surface Normals. In *IEEE Int. Conf. Robotics and Automation*, 2022. 3, 4

[38] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable Large Scene Neural View Synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1, 2, 4, 5, 6, 7, 8

[39] Jingkang Wang, Sivabalan Manivasagam, Yun Chen, Ze Yang, Ioan Andrei Bârsan, Joyce Anqi Yang, Wei-Chiu Ma, and Raquel Urtasun. CADSim: Robust and Scalable in-the-wild 3D Reconstruction for Realistic and Controllable Sensor Simulation. In *Conf. on Robot Learning*, 2022. 4

[40] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. In *arXiv*, 2019. 6

[41] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. NerfingMVS: Guided Optimization of Neural Radiance Fields for Indoor Multi-view Stereo. In *Int. Conf. Comput. Vis.*, 2021. 2

[42] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting. In *Conf. Neural Inform. Process. Syst.*, 2021. 1, 5

[43] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 1, 2, 7, 8

[44] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based Neural Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1, 2, 3, 4, 6, 8

[45] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, and Qinhong Chen. Plenoxels: Radiance Fields without Neural Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2

[46] Huangying Zhan, Jiyang Zheng, Yi Xu, Ian Reid, and Hamid Rezatofighi. ActiveRMAP: Radiance Field for Active Mapping And Planning. In *arXiv*, 2022. 1

[47] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Int. Conf. Comput. Vis.*, 2017. 2, 4