MIMO-NeRF: Fast Neural Rendering with Multi-input Multi-output Neural Radiance Fields

Takuhiro Kaneko
NTT Communication Science Laboratories, NTT Corporation

Abstract

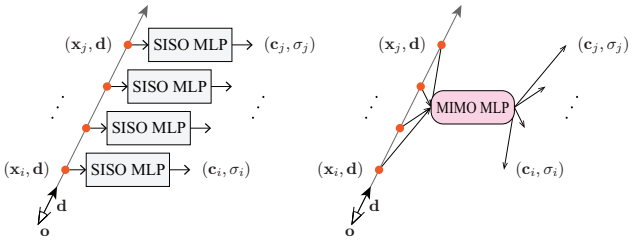
Neural radiance fields (NeRFs) have shown impressive results for novel view synthesis. However, they depend on the repetitive use of a single-input single-output multilayer perceptron (SISO MLP) that maps 3D coordinates and view direction to the color and volume density in a sample-wise manner, which slows the rendering. We propose a multi-input multi-output NeRF (MIMO-NeRF) that reduces the number of MLPs running by replacing the SISO MLP with a MIMO MLP and conducting mappings in a group-wise manner. One notable challenge with this approach is that the color and volume density of each point can differ according to a choice of input coordinates in a group, which can lead to some notable ambiguity. We also propose a self-supervised learning method that regularizes the MIMO MLP with multiple fast reformulated MLPs to alleviate this ambiguity without using pretrained models. The results of a comprehensive experimental evaluation including comparative and ablation studies are presented to show that MIMO-NeRF obtains a good trade-off between speed and quality with a reasonable training time. We then demonstrate that MIMO-NeRF is compatible with and complementary to previous advancements in NeRFs by applying it to two representative fast NeRFs, i.e., a NeRF with a sampling network (DONeRF) and a NeRF with alternative representations (TensoRF).[1]

1. Introduction

Images are two-dimensional (2D) projections of three-dimensional (3D) scenes. Solving the inverse problem, that is, learning 3D representations from 2D images and synthesizing novel views, is a fundamental concern in computer vision and graphics and has been extensively studied for various applications such as photo editing, content creation, virtual reality, and environmental understanding.

With the advent of implicit neural representations (e.g., [56, 44, 39, 28, 69, 71, 55]), substantial advancements have been made towards addressing this problem. Neural radiance fields (NeRFs) [39] have been noted as a successful approach. A NeRF represents a scene using a continuous function that maps 3D coordinates and view direction to the color and volume density and renders a pixel by integrating the outputs on a ray using volume rendering [34]. This formulation enables NeRF to learn to synthesize geometrically consistent and high-fidelity novel views with only 2D supervision.

Despite this advantage, a typical NeRF suffers from slow rendering because it uses a *single-input single-output (SISO)* MLP that calculates the RGB color and volume density in a *sample-wise* manner (Figure 1(a)). Although this architecture ensures the independent representation of each point, which is useful, for example, for learning view-independent volume density, its computational cost increases in proportion to the number of samples for each



(a) NeRF (b) MIMO-NeRF (proposed)

Figure 1. Comparison between NeRF and MIMO-NeRF (proposed). (a) A typical NeRF uses a SISO MLP that maps 3D coordinates and view direction to the color and volume density in a *sample-wise* manner. (b) In contrast, the proposed MIMO-NeRF uses a MIMO MLP that performs mappings in a *group-wise* manner. This change reduces the number of MLPs running and improves the rendering speed, but also requires addressing ambiguity in the color and volume density caused by the fact that these values are determined in a non-unique manner by a set of input coordinates that vary by viewpoint, grouping, and sampling. The main technical contribution of the present work is that of providing methods to mitigate this challenge. We demonstrate the impact of the proposed technique in Figure 2.

[1]The project page is available at https://www.kecl.ntt.co.jp/people/kaneko.takuhiro/projects/mimo-nerf/.

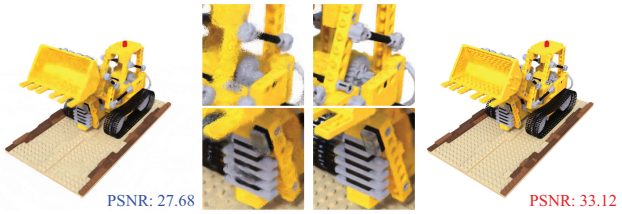
ray (e.g., on the order of hundreds). Several methods developed to address this issue can be roughly categorized into two approaches, including (1) *sample reduction* and (2) *alternative representations*.

A typical sample reduction strategy reduces the number of samples on a ray using a sampling network based on the depth [41] or density of a pretrained NeRF [47] or using a sampling network with an adaptive optimization mechanism [29, 14, 27]. These methods successfully accelerate the rendering process while retaining the quality of an image adequately. However, most of these techniques still use a SISO MLP to predict the colors and volume densities of selected samples; therefore, they still need to run MLPs many times in proportion to the number of selected samples. This issue can be alleviated by reducing the number of selected samples, although this deteriorates the quality of the synthesized images accordingly.

As alternative representations, various sophisticated and faster representations such as 3D voxel grids [17, 22, 30, 66, 60], sparse voxel-based octrees [70, 15], multiplane images [65], tri-planes [7], vector-matrix decomposition [9], hashes [40], NeRF-specific structures [24], and space-wise MLPs [50, 51] have been devised. These representations contribute to achieving fast rendering while retaining image quality moderately well. However, after powerful features are extracted using alternative representations, SISO MLPs are still commonly used for the final prediction of color or volume density owing to their memory-efficient and continuous nature. Hence, part of the calculation cost still increases depending on the number of samples.

Consequently, owing to its compact, continuous (i.e., resolution-free), and independent (e.g., view-independent) nature, a SISO MLP is commonly used in various NeRFs. However, as mentioned above, the calculation cost increases with the number of samples. This is not preferable when considering the improvement in rendering speed. Possible simple solutions include, for example, a reduction of the number of samples or a reduction of the size of a model with a corresponding sacrifice of image quality. However, these solutions are not necessarily the best for handling the trade-off between quality and speed.[2] Alternatively, we propose a *multi-input multi-output NeRF (MIMO-NeRF)*, which is a novel variant of NeRF that represents a scene using a *MIMO* MLP that conducts mappings in a *group-wise* manner (Figure 1(b)). This modification enables a reduction in the number of MLPs running according to the number of grouped samples and consequently improves rendering speed.

However, in this approach, the uniqueness of the color and volume density of each point is not ensured because they are determined not only by the coordinates of the corresponding point but also by the coordinates of the other points in a group, which vary by viewpoint, grouping, and sampling. This leads to some ambiguity and causes fluctuation artifacts as shown in Figure 2(a). In particular, this ambiguity can be problematic when learning a 3D representation using only 2D supervision because obtaining direct supervision that can resolve the ambiguity is difficult. One possible solution is to train a standard (i.e., SISO) NeRF first and then distill the model onto the corresponding MIMO-NeRF. However, this increases training time because both student and teacher NeRFs must be trained. Alternatively, we have also developed a novel *self-supervised learning* approach in which we reformulate a MIMO MLP in several ways (in particular, we use *group shift* (Figure 3) and *variation reduction* (Figure 4)) and impose a consistent regularization so that the reformulated MIMO MLPs produce the same outputs. Because each reformulated MIMO MLP can render a pixel faster than the original SISO MLP, we can prevent a large sacrifice of training time even when using multiple reformulated MIMO MLPs by adequately adjusting the reformulation configuration. Figure 2(b) shows an example of the effects of this learning scheme.



(a) MIMO-NeRF-naive (*without* self-supervised learning)

(b) MIMO-NeRF-self (*with* self-supervised learning)

Figure 2. Challenge of training MIMO-NeRF and the impact of the proposed self-supervised learning. (a) MIMO-NeRF-naive suffers from ambiguity in the color and volume density of each point and deteriorates image quality. (b) We propose self-supervised learning to address this problem without relying on a pretrained model. This technique improves image quality.

We investigated the benchmark performance of MIMO-NeRF by comparing it with possible alternatives (including the distillation of a pretrained NeRF, sample reduction, and reduction of the model size). We also performed ablation studies to examine the validity of each component of the proposed self-supervised learning method. We apply MiMO-NeRF to two representative fast NeRFs, including a NeRF with a sampling network (DONeRF [41]) and a model with alternative representations (TensoRF [9]) to demonstrate that it is compatible with and complementary to previous advancements in NeRFs.

The main contributions of this study are summarized as follows.

- To speed up the rendering of NeRF, we propose *MIMO-NeRF*, which represents a scene using a *MIMO*

[2] We discuss this trade-off in detail in Section 5.2.

MLP that maps the coordinates on a ray to the colors and volume densities in a *group-wise* manner.

- We introduce novel *self-supervised learning* to mitigate the ambiguity in the color and volume density of each point and enable MIMO-NeRF to be trained without relying on pretrained models.
- We examined the effectiveness of MIMO-NeRF through a comprehensive experimental evaluation, and the results demonstrate the versatility of MIMO-NeRF in applications to two representative fast NeRFs. We also provide more detailed analyses and extended results in the Supplementary Material.

2. Related work

Implicit neural representations. Implicit neural representations have attracted attention in 3D shape [45, 35, 11, 36, 52, 18, 3, 20] and 3D scene [25, 46, 6, 12, 54] reconstructions owing to their memory-efficient, continuous (i.e., resolution-free), and 3D-aware characteristics. Although research began with explicit 3D supervision, learning implicit 3D only from 2D supervision (i.e., inverse graphics) has also been achieved by incorporating differentiable rendering [56, 31, 44, 32, 39, 28, 69, 71, 55]. In this study, we focus on NeRFs as a representative example of the latter owing to their remarkable success in synthesizing geometrically consistent and high-quality novel view. However, applying our ideas to other implicit neural representations, such as those mentioned above, remains as an interesting direction for future research.

Advancements in NeRFs. Various extensions have been proposed since the emergence of NeRF. For example, representative research topics include (1) improving image quality and enhancing applicable scenes (e.g., [72, 33, 49, 16, 67, 4, 5, 37, 62, 10]), (2) incorporation into other models, e.g., deep generative models, such as generative adversarial networks (GANs) [19] and diffusion probabilistic models [58, 23] (e.g., [53, 8, 43, 42, 21, 7, 13, 68, 26, 57, 48]), and (3) accelerating NeRFs for fast inference or fast training (e.g., [41, 47, 29, 14, 27, 17, 22, 30, 66, 60, 70, 15, 65, 7, 9, 40, 24, 50, 51]). The present work falls into the third category. However, our proposed approach is complementary to previous studies, including most of the abovementioned works in all categories, because SISO MLPs have commonly been used as a partial or main network in previous studies and improving their rendering speed by replacing SISO MLPs with the proposed MIMO MLP is feasible. The results of the experimental evaluation validated this potential (Sections 5.4 and 5.5).

Acceleration of NeRFs. As discussed in Section 1, a typical NeRF is well known for its slow rendering because it use a SISO MLP. Several approaches have been developed to address this issue. These can be roughly categorized into two approaches, including (1) *sample reduction* and (2) *alternative representations*. (1) As described in Section 1, methods that reduce the number of samples using a sampling network can improve rendering speed by replacing a SISO MLP with the proposed MIMO MLP. We validated this statement during an experiment (Section 5.4) by applying our ideas to a representative NeRF in this category (DONeRF [41]). Another common approach in the first category is to render pixels using a light-field network [55, 2, 59, 63] instead of volume rendering [34]. This approach has been shown to achieve fast rendering by running only a single MLP for a given ray. However, owing to the lack of explicit geometry-aware representations driven by the use of volume densities, these methods suffer from limitations that are not faced by a standard NeRF in terms of restrictions on applicable scenes (e.g., toy datasets [55] and forward-facing datasets [2]), a requirement for high-capacity models (e.g., a deeper MLP [63] and a transformer [59]), and the need for extra modules (e.g., meta-learned priors [55], pretrained NeRFs [2, 63], or additional encoders [59]). (2) As explained in Section 1, alternative representations have the potential to accelerate the rendering speed by replacing the SISO MLP with the proposed MIMO MLP. We present an empirical investigation of this potential in Section 5.5 by incorporating MIMO-NeRF into TensoRF [9], a representative model in this category.

Learning of fast NeRFs. Knowledge distillation (or baking) methods are commonly used to train fast NeRFs. In these methods, a standard NeRF is first trained and then baked to faster representations [17, 22, 24, 50, 51]. However, this approach is disadvantageous in terms of training time because two separate models must be trained, i.e., teacher and student NeRFs. As an alternative, we consider a self-supervised learning approach in which we can train a model without a large increase in training time. We examine the performance differences between the proposed self-supervised learning scheme and a knowledge distillation scheme in Section 5.1.

3. Preliminaries: NeRF

We begin by explaining NeRFs as the basis for our model. As shown in Figure 1(a), a NeRF represents a point in a 3D space using a continuous SISO function f_{SISO} that maps the 3D position $\mathbf{x} \in \mathbb{R}^3$ and view direction $\mathbf{d} \in \mathbb{S}^2$ to the RGB color $\mathbf{c}(\mathbf{x}, \mathbf{d}) \in \mathbb{R}^3$ and volume density $\sigma(\mathbf{x}) \in \mathbb{R}^+$ in a sample-wise manner.

$$f_{\text{SISO}} : \mathbb{R}^3 \times \mathbb{S}^2 \to \mathbb{R}^3 \times \mathbb{R}^+, \quad (\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma). \tag{1}$$

Specifically, positional encoding [39, 61] is applied to $\mathbf{x}$ and $\mathbf{d}$ to represent the high-frequency details of an image. Subsequently, an MLP is applied to the encoded inputs to obtain $\mathbf{c}$ and σ. For simplicity, we represent these series of processes in a unified manner as f_{SISO}.

A NeRF is based on ray tracing, in which a camera ray is defined as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ and $\mathbf{d}$ respectively denote the origin and direction of the camera and t denotes a distance from the origin. A NeRF calculates the color of each pixel $\hat{\mathbf{C}}(\mathbf{r})$ by integrating the colors and volume densities on a ray $\mathbf{r}(t)$ within $t \in [t_n, t_f]$ using volume rendering [34]. In implementation, the calculation of the integral is intractable; therefore, a ray is discretized into N points; alternatively, the following discretized formulation can be used.

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i, \text{where } T_i = \prod_{j=1}^{i-1}(1-\alpha_j), \quad (2)$$

where the subscript i indicates that the variable corresponds to the i-th point on a ray, $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$ is an alpha value, and $\delta_i = t_{i+1} - t_i$ is the distance between the i-th and $(i+1)$-th points. f_{SISO} is optimized by minimizing the following pixel-wise loss.

$$\mathcal{L}_{\text{pixel}} = \|\hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r})\|_2^2, \quad (3)$$

where $\mathbf{C}(\mathbf{r})$ is the ground-truth color for a ray $\mathbf{r}$. In practice, this loss is calculated for $\mathbf{r} \in \mathcal{R}$, where $\mathcal{R}$ denotes a set of rays in each batch.

In practice, a NeRF uses coarse and fine networks. In the coarse network, a ray is discretized into N_c points using stratified sampling, whereas in the fine network, a ray is discretized into $N_c + N_f$ points using hierarchical sampling in which N_f additional points are sampled according to the output of the coarse network. The two networks are optimized by minimizing $\mathcal{L}_{\text{pixel}}$ for $\hat{\mathbf{C}}(\mathbf{r})$ predicted by each network. Hereafter, we omit a variable in parentheses (e.g., $(\mathbf{r})$) for simplicity.

4. MIMO-NeRF

4.1. MIMO formulation

There are several ways to group the input samples when constructing a MIMO MLP. For example, we can construct a general MLP that can accept any combination of samples in a 3D space, or we can construct a specific MLP that only accepts a group of nearby samples. In preliminary experiments (Appendix A.1), we found that the latter significantly outperformed the former because general models are more difficult to train than more specific models. Therefore, we adopted the latter in this study. In particular, we group neighboring samples on a ray as shown in Figure 1(b).[3]

More formally, given N samples on a ray, we group N_p samples from the sample nearest to the camera and create N/N_p groups. Subsequently, we apply a MIMO function f_{MIMO} to each group as follows.

$$f_{\text{MIMO}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \to (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p}, \\ (\mathbf{x}_i, \ldots, \mathbf{x}_j, \mathbf{d}) \mapsto (\mathbf{c}_i, \ldots, \mathbf{c}_j, \sigma_i, \ldots, \sigma_j), \quad (4)$$

where $i \in \{1, 1+N_p, \ldots, 1+N-N_p\}$ and $j = i + N_p - 1$. Assuming that the grouped samples are lined on a ray, we use a single direction $\mathbf{d}$ in the input of f_{MIMO}. In this formulation, the number of MLPs running to render a pixel (*# Run*) is equal to the number of groups and is calculated as # Run $= N/N_p$. Therefore, we can reduce the calculation cost, particularly that of # Run, by increasing N_p.

In inference, the only necessary modification to a NeRF is the simple replacement of f_{SISO} with f_{MIMO} and the same volume rendering (Equation 2) and sampling scheme (i.e., stratified and hierarchical sampling) can be used. With this modification strategy, MIMO-NeRF exhibits high compatibility and complementarity with previous NeRFs.

During training, $\mathcal{L}_{\text{pixel}}$ (Equation 3) can be used for $\hat{\mathbf{C}}$ obtained using f_{MIMO}. However, this loss does not necessarily suffice to address the ambiguity in the color and volume density of each point because this ambiguity occurs in a 3D space and the loss cannot regularize the 3D representations explicitly. Hence, we introduce self-supervised learning as discussed in subsequent sections.

4.2. MIMO reformulation

One possible solution to this ambiguity is to train a standard (i.e., SISO) NeRF first and then distill the model onto a corresponding MIMO-NeRF. However, this solution involves an increase in training time because it requires training not only a MIMO-NeRF but also a SISO NeRF. Consequently, this solution cannot entirely take advantage of the fast rendering of MIMO-NeRF in training.

Alternatively, we reformulate a MIMO MLP in multiple ways and impose a consistent regularization to produce the same RGB colors and alpha values. Because each reformulated MIMO MLP can render a pixel faster than the original SISO MLP, we can prevent a large increase in training time even when using multiple reformulated MIMO MLPs by adequately adjusting the reformulation configuration. When implementing this idea, the question arises as to how best to reformulate a MIMO MLP. To this end, we developed two methods, including (1) *group shift* and (2) *variation reduction*.

Group shift. We consider restricting this ambiguity by assessing each point in multiple ways using different groups and imposing consistency on the assessed results. More formally, we implement this by shifting groups and rewriting Equation 4 as follows.

$$f^{\text{shift}}_{\text{MIMO}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \to (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p}, \\ (\mathbf{x}_{i'}, \ldots, \mathbf{x}_{j'}, \mathbf{d}) \mapsto (\mathbf{c}_{i'}, \ldots, \mathbf{c}_{j'}, \sigma_{i'}, \ldots, \sigma_{j'}), \quad (5)$$

[3] One possible alternative is to group near samples on different rays. However, in NeRFs, searching near points across different rays is not trivial because points are sampled unevenly via hierarchical sampling. Therefore, we simply group neighboring samples on the same ray in this study.

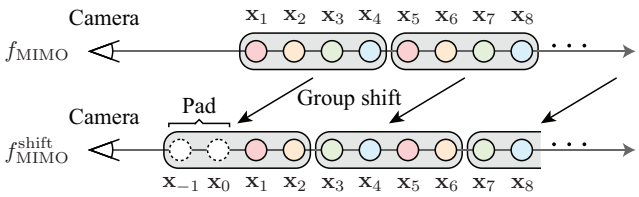


Figure 3. Example of the group shift when $N_p = 4$ and $s = 2$. We shift each group by s toward the camera after padding s samples before the front sample. This procedure enables assessing each sample in multiple ways using different groups.

where $i' \in \{k, k+N_p, \dots, k+N\}$ and $j' = i' + N_p - 1$. Here, k is the head index of the first group, which is shifted by $s \in \{1, \dots, N_p - 1\}$ toward the camera. Hence, $k \in \{2-N_p, \dots, 0\}$. In practice, it is randomly sampled during training. More strictly, we add padding before this process to represent a sample with an index exceeding the original index, i.e., $i' < 1$ or $j' > N$. For clarity, we present an example in which $N_p = 4$ and $s = 2$ in Figure 3.

Variation reduction. In the original MIMO formulation, the abovementioned ambiguity is caused by N_p different input samples. To mitigate this, we consider reducing the variation in the input by replacing Equation 4 with the following.

$$f^{\text{reduce}}_{\text{MIMO}} : (\mathbb{R}^3)^{N_p} \times \mathbb{S}^2 \to (\mathbb{R}^3 \times \mathbb{R}^+)^{N_p},$$
$$([\mathbf{x}_{i''}]^R, \dots, [\mathbf{x}_{j''}]^R, \mathbf{d}) \mapsto (\mathbf{c}_{i''_1}, \dots, \mathbf{c}_{i''_R}, \dots, \mathbf{c}_{j''_1}, \dots, \mathbf{c}_{j''_R}, \sigma_{i''_1}, \dots, \sigma_{i''_R}, \dots, \sigma_{j''_1}, \dots, \sigma_{j''_R}), \tag{6}$$

where $[\cdot]^R$ denotes the operation of repeating the given variable R times, $i'' \in \{1, 1+\frac{N_p}{R}, \dots, 1+N-\frac{N_p}{R}\}$, and $j'' = i'' + \frac{N_p}{R} - 1$. After applying $f^{\text{reduce}}_{\text{MIMO}}$, we average $(\mathbf{c}_{k''_1}, \dots, \mathbf{c}_{k''_R})$ and $(\sigma_{k''_1}, \dots, \sigma_{k''_R})$ to obtain $\mathbf{c}_{k''}$ and $\sigma_{k''}$, where $k'' \in \{i'', \dots, j''\}$. In this formulation, the mentioned ambiguity is reduced by decreasing input variation from N_p to $\frac{N_p}{R}$. Conversely, the number of MLPs running increases by factor of R, i.e., # Run $= N/N_p \times R$; therefore, we need to select R carefully in practice to avoid a large increase in training time. For clarity, we present an example case in which $N_p = 4$ and $R = 2$ in Figure 4.

4.3. MIMO objective

Through the above processes, we obtain M reformulated MIMO MLPs in which we use different s for each MIMO MLP and set R such that the total number of # Run is not larger than that of the original SISO MLP. Hereafter, we use the superscript $m \in \{1, \dots, M\}$ to denote the variable corresponding to the m-th reformulated MIMO MLP, e.g., $\hat{\mathbf{C}}^m$ and R^m. We train these MLPs using two loss functions, including pixel-wise and 3D consistency losses.

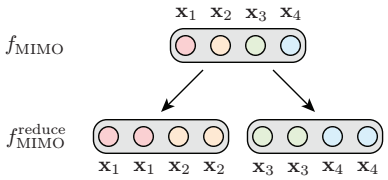


Figure 4. Example of variation reduction when $N_p = 4$ and $R = 2$. Samples with the same color correspond to the same coordinate. In $f^{\text{reduce}}_{\text{MIMO}}$, we reduce the number of unique samples in a group from N_p $(= 4)$ to $\frac{N_p}{R}$ $(= 2)$ by repeating each sample R $(= 2)$ times to reduce the variation in a group.

Pixel-wise loss. We apply the pixel-wise loss (Equation 3) to each $\hat{\mathbf{C}}^m$ as follows.

$$\mathcal{L}^{\text{MIMO}}_{\text{pixel}} = \sum_{m=1}^{M} \|\hat{\mathbf{C}}^m - \mathbf{C}\|_2^2. \tag{7}$$

3D consistency loss. The pixel-wise loss provides supervision in a 2D space; however, it cannot impose an explicit regularization in a 3D space. Hence, we introduce a 3D consistency loss that encourages the reformulated MIMO MLPs to produce the same colors and alpha values in the 3D space. The 3D consistency loss consists of a color 3D consistency loss $\mathcal{L}^{\text{color}}_{\text{3D}}$ and an alpha value 3D consistency loss $\mathcal{L}^{\text{alpha}}_{\text{3D}}$ as follows.

$$\mathcal{L}^{\text{color}}_{\text{3D}} = \sum_{m_1=1}^{M-1} \sum_{m_2=m_1+1}^{M} \frac{1}{N} \sum_{i=1}^{N} [\mu^{m_2}_{m_1} \|\mathbf{c}_i^{m_1} - \text{sg}(\mathbf{c}_i^{m_2})\|_2^2 + \mu^{m_1}_{m_2} \|\text{sg}(\mathbf{c}_i^{m_1}) - \mathbf{c}_i^{m_2}\|_2^2], \tag{8}$$

$$\mathcal{L}^{\text{alpha}}_{\text{3D}} = \sum_{m_1=1}^{M-1} \sum_{m_2=m_1+1}^{M} \frac{1}{N} \sum_{i=1}^{N} [\mu^{m_2}_{m_1} \|\alpha_i^{m_1} - \text{sg}(\alpha_i^{m_2})\|_2^2 + \mu^{m_1}_{m_2} \|\text{sg}(\alpha_i^{m_1}) - \alpha_i^{m_2}\|_2^2]. \tag{9}$$

where "sg" indicate a stop-gradient operation. The 3D consistency loss $\mathcal{L}_{\text{3D}}$ is calculated by $\mathcal{L}_{\text{3D}} = \mathcal{L}^{\text{color}}_{\text{3D}} + \mathcal{L}^{\text{alpha}}_{\text{3D}}$. We define $\mu^{m_j}_{m_i}$ as $\mu^{m_j}_{m_i} = \frac{\sqrt{R^{m_j}}}{\sqrt{R^{\max}}\sqrt{R^{m_i}}}$, where $R^{\max}$ is the maximum of R^m in $m \in \{1, \dots, M\}$. We use this asymmetric weight on the assumption that $\mathbf{c}_i^m$ and α_i^m with greater R^m have lower ambiguity and are more reliable. Hence, the effect of $\mathcal{L}_{\text{3D}}$ is reduced. We empirically investigated the importance of this effect through an ablation study as described in Section 5.3.

Full objective. The full objective is defined as follows.

$$\mathcal{L}_{\text{MIMO}} = \mathcal{L}^{\text{MIMO}}_{\text{pixel}} + \lambda \mathcal{L}_{\text{3D}}, \tag{10}$$

where λ is a hyperparameter that balances the pixel-wise loss and 3D consistency loss.

5. Experiments

We conducted five experiments to investigate the effectiveness of MIMO-NeRF. In the first three experiments, we conducted a comprehensive study, including an investigation of benchmark performance (Section 5.1), an investigation of the trade-off between speed and quality (Section 5.2), and ablation studies (Section 5.3). In the remaining two experiments, we examined the versatility of MIMO-NeRF by applying it to two representative fast NeRFs, including a NeRF with a sampling network, i.e., DONeRF [41] (Section 5.4), and a NeRF with alternative representations, i.e., TensoRF [9] (Section 5.5). The main results of these experiments are provided here, and detailed analyses are presented with extended results in Appendix A. The implementation details are presented in Appendix B.

5.1. Investigation of benchmark performance

First, we investigated the benchmark performance of MIMO-NeRF by applying our ideas to the original *NeRF* [39]. In particular, we examined three variants of MIMO-NeRF, including *MIMO-NeRF-naive*, which simply replaced f_{SISO} (Equation 1) with f_{MIMO} (Equation 4) and was trained with a standard pixel-wise loss (Equation 3).[4] *MIMO-NeRF-distill*, which is a student model distilled from a pretrained standard (i.e., SISO) NeRF. During training, we used a 3D consistency loss (Equations 8 and 9) that was adjusted for knowledge distillation, in addition to the pixel-wise loss. *MIMO-NeRF-self*, which is MIMO-NeRF that adopted the proposed self-supervised learning. We examined the performance when N_p was varied within $\{2, 4, 8\}$.

Datasets. We investigated the benchmark performance on two commonly-used datasets. (1) *Blender dataset* [39] includes eight scenes, each of which consists of 360° views of complex objects at a resolution of 800×800 pixels. We used 100 and 200 views for training and testing, respectively. (2) *Local Light Field Fusion (LLFF) dataset* [38, 39], which consists of eight complex real-world scenes, each of which includes 20–62 forward-facing views at 1008×756 pixels. One-eighth of the images were used for testing, and the remainder were used for training. Where not otherwise specified, we used half-sized images following the default settings of a widely-used source code for NeRF[5] to better investigate the various configurations.

Implementation. For a fair comparison, we implemented all the models with a commonly-used source code for NeRF[5] and trained the models using the default settings provided in the code. The number of samples was set as $N_c = 64$ and $N_f = 128$ for the Blender dataset and $N_c = 64$ and $N_f = 64$ for the LLFF dataset. For MIMO-NeRF-self, we used two formulations with $R^1 = R^2 = 1$ when $N_p = 2$, two formulations with $R^1 = 1$ and $R^2 = 2$ when $N_p = 4$, and three formulations with $R^1 = 1$, $R^2 = 2$, and $R^3 = 4$ when $N_p = 8$. MIMO-NeRF-self was trained individually depending on N_p. An investigation of different reformulation methods is presented in Appendix A.2. Group shifts were applied to all cases. We set λ to 1 and 0.4 for the Blender and LLFF datasets, respectively. The effect of λ is analyzed in Appendix A.3. The implementation details are presented in Appendix B.1.

Evaluation metrics. Following the original NeRF study [39], we used the peak signal-to-noise ratio (*PSNR*), structural similarity index (*SSIM*) [64], and learned perceptual image patch similarity (*LPIPS*) [73] to quantitatively evaluate the image quality. To assess the calculation cost of inference and training, we report inference time (*I-time*) measured with an NVIDIA GeForce RTX 3080 Ti GPU and training time (*T-time*) measured with an NVIDIA A100-SXM4-80GB GPU.[6] We also provide *# Run* $\left(= \frac{N_c + (N_c + N_f)}{N_p}\right)$ and the number of parameters (*# Params*) as supplementary information. # Params increases in MIMO-NeRF mainly because the total dimension of the positional embeddings is increased by N_p times according to the increase in the inputs.

Results. From Table 1, the following is observed.

Image quality. MIMO-NeRF-self outperformed not only MIMO-NeRF-naive but also MIMO-NeRF-distill in most cases in terms of PSNR, SSIM, and LPIPS. We conjecture that this occurred because the joint optimization of the teacher and student networks in MIMO-NeRF-self was more effective for training than the student-only optimization in MIMO-NeRF-distill. Even MIMO-NeRF-self suffered from a trade-off between speed and quality with increasing N_p; however, MIMO-NeRF-self performed better than or comparably to the original NeRF when $N_p = 2$. This may have occurred because the advantage of accumulating neighboring information and the disadvantage of handling the ambiguity were antagonistic in this case.

Inference speed. All MIMO-NeRFs with the same inference procedure showed inference times improved by a factor of 1.84–5.78 with increasing N_p.

Training speed. MIMO-NeRF-naive achieved the fastest training because it used only a single MIMO formulation during training. MIMO-NeRF-self required more training time because it uses multiple MIMO reformulations; however, each calculation cost is low. Therefore, it did not suffer from a large increase in training time compared

[4] For simplicity and a fair comparison, we only increased the input and output of the original SISO MLP and retained the other parameters (e.g., depth and width). Hence, the increase in model size was relatively small.

[5] https://github.com/yenchenlin/nerf-pytorch

[6] For simplicity and a fair comparison, we measured the calculation time using a standard PyTorch implementation[5] for all the models. Optimizing the implementation for faster rendering (e.g., using custom CUDA kernels) would be interesting for future research.

Model	N_p	Blender PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)	LLFF PSNR↑	SSIM↑	LPIPS↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)
NeRF	1	31.04	0.951	0.055	256	9.60	4.70	1.19	27.72	0.871	0.150	192	8.38	3.39	1.19
MIMO-NeRF-naive	2	30.18	0.944	0.065	128	5.15	3.09	1.26	27.31	0.860	0.167	96	4.55	2.12	1.26
MIMO-NeRF-distill	2	30.76	0.949	0.058	128	5.15	9.46	1.26	27.50	0.863	0.169	96	4.55	6.81	1.26
MIMO-NeRF-self	2	31.26	0.953	0.054	128	5.15	5.36	1.26	27.70	0.870	0.155	96	4.55	3.97	1.26
MIMO-NeRF-naive	4	28.62	0.927	0.091	64	2.79	2.02	1.39	26.29	0.824	0.218	48	2.46	1.57	1.39
MIMO-NeRF-distill	4	30.22	0.946	0.065	64	2.79	8.42	1.39	27.37	0.861	0.172	48	2.46	6.25	1.39
MIMO-NeRF-self	4	30.94	0.950	0.058	64	2.79	4.68	1.39	27.51	0.865	0.162	48	2.46	3.44	1.39
MIMO-NeRF-naive	8	26.34	0.895	0.133	32	1.66	1.66	1.65	25.10	0.774	0.284	24	1.45	1.24	1.65
MIMO-NeRF-distill	8	29.39	0.937	0.075	32	1.66	8.07	1.65	27.01	0.851	0.184	24	1.45	5.91	1.65
MIMO-NeRF-self	8	30.40	0.945	0.065	32	1.66	5.86	1.65	26.97	0.851	0.180	24	1.45	4.43	1.65

Table 1. Benchmark performance of MIMO-NeRFs. MIMO-NeRF-self outperformed MIMO-NeRF-naive and MIMO-NeRF-distill in terms of PSNR, SSIM, and LPIPS in most cases with shorter training time than MIMO-NeRF-distill. All MIMO-NeRFs outperformed the original NeRF in terms of inference time owing to the reduction of # Run.

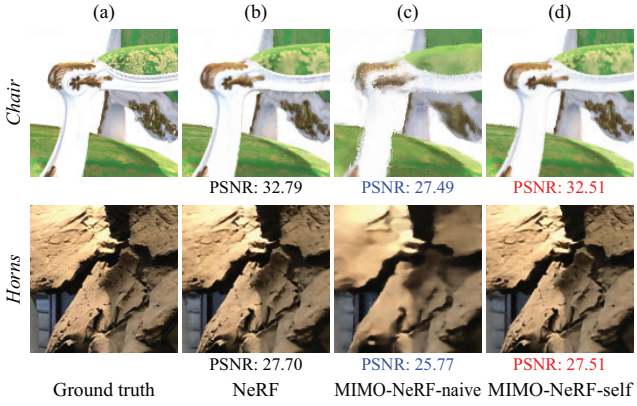


Figure 5. Qualitative comparison between NeRF and MIMO-NeRFs with $N_p = 8$. The models were trained using full-sized images. MIMO-NeRF-naive (c) produced some artifacts owing to the ambiguity in the color and volume density of each point. MIMO-NeRF-self (d) was useful for addressing this issue, and its results were close to those of NeRF (b), while inference time were improved by a factor of 5.8.

with MIMO-NeRF-distill, which requires training not only a MIMO-NeRF but also a SISO NeRF.

Summary. From these results, we found that when $N_p = 2$, MIMO-NeRF-self improved the inference speed of NeRF without compromising image quality, and when N_p was larger, there was a trade-off between speed and quality. We examine the validity of this trade-off in Section 5.2.

Qualitative results. Figure 5 shows a qualitative comparison between NeRF, MIMO-NeRF-naive, and MIMO-NeRF-self. In this experiment, we used full-sized images to train the models and set N_p to 8 for MIMO-NeRFs. MIMO-NeRF-naive produced some artifacts, whereas MIMO-NeRF-self adequately addressed this issue. The additional results are provided in Appendix A.6.

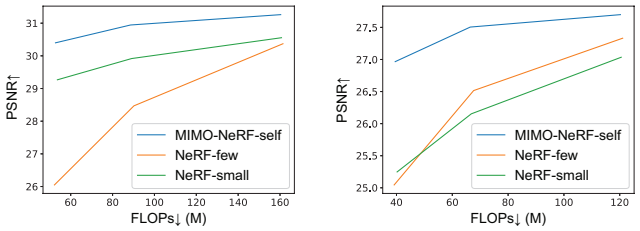


Figure 6. Relationship between FLOPs and PSNR. Higher values indicate better the image quality. Faster speeds are shown to the left. MIMO-NeRF-self achieves the best trade-off between speed and quality.

5.2. Investigation of speed-quality trade-off

We compared MIMO-NeRF with possible alternatives to investigate whether it achieved a good trade-off between speed and quality. In particular, we focused on methods that are general and applicable to various NeRFs, similar to MIMO-NeRF, and examined two variants, including *NeRF-few*, which reduced the number of samples on a ray, and *NeRF-small*, which reduced the number of features in the hidden layers. We adjusted the parameters such that their FLOPs were comparable to those of MIMO-NeRF.[7]

Results. The relationship between the FLOPs and PSNR is plotted in Figure 6. We found that MIMO-NeRF-self obtained a better trade-off between speed and quality than NeRF-few and NeRF-small. We provide other relationships (e.g., relationships between FLOPs/inference time and PSNR/SSIM/LPIPS) in Appendix A.4.[8]

[7]We tuned the models based on FLOPs because the performance of different methods for improving speed in terms of inference time may vary depending on the calculation tools used, such as GPU processor hardware. As a reference, we provide the relationship between the inference time and image quality in Appendix A.4.

[8]During training, the calculation cost of MIMO-NeRF-self was larger than those of NeRF-small and NeRF-few because multiple MIMO reformulations were used. To confirm this effect, we examined the performance of NeRF-few and NeRF-small with increasing batch sizes such that the calculation costs became almost the same as that of MIMO-NeRF-self. We

N_p	GS	CL	AW	Blender PSNR↑	Blender SSIM↑	Blender LPIPS↓	LLFF PSNR↑	LLFF SSIM↑	LLFF LPIPS↓
2		✓		30.17	0.943	0.067	27.21	0.856	0.170
	✓			30.54	0.945	0.065	27.48	0.865	0.161
	✓	✓		31.26	0.953	0.054	27.70	0.870	0.155
4		✓	✓	30.84	0.949	0.058	27.39	0.862	0.166
	✓			29.48	0.936	0.077	26.46	0.832	0.206
	✓	✓		30.87	0.949	0.060	27.44	0.864	0.164
	✓	✓	✓	30.94	0.950	0.058	27.51	0.865	0.162
8		✓	✓	29.81	0.941	0.069	26.97	0.851	0.179
	✓			27.39	0.907	0.116	24.29	0.734	0.332
	✓	✓		30.16	0.942	0.071	26.69	0.843	0.192
	✓	✓	✓	30.40	0.945	0.065	26.97	0.851	0.180

Table 2. Results of ablation studies. Check marks in GS, CL, and AW indicate the use of group shift, 3D consistency loss, and asymmetric weights. In $N_p = 2$, asymmetric weights were not used in the full model because $R^1 = R^2 = 1$. Hence, it was not ablated.

5.3. Ablation studies

We also conducted ablation studies to better understand the performance of each element of the proposed self-supervised learning method. Specifically, we investigated the importance of group shift, 3D consistency loss, and asymmetric weights. When asymmetric weights were ablated, $\mu_{m_i}^{m_j}$ was set to 1.

Results. The results are listed in Table 2. We found that the full model achieved the best performance in most cases. The results validate the importance of each technique.

5.4. Application to DONeRF

We incorporated MIMO-NeRF into DONeRF [41], a representative NeRF with a sampling network, to demonstrate that MIMO-NeRF can complement existing fast NeRFs. DONeRF uses a sampling network called a depth oracle network to select samples and calculates the colors and volume densities of the selected samples using a shading network. It handles the trade-off between speed and image quality by adjusting the number of selected samples (N_s). We examined whether MIMO-NeRF could be used as an alternative to handle this trade-off.

Datasets. We evaluated the performance using the *DONeRF dataset* [41] comprising six synthetic indoor and outdoor scenes. Each scene included 300 forward-facing views at a resolution of 800×800 pixels. 70%, 10%, and 20% of the images were used for training, validation, and testing, respectively.

Implementation. We implemented the models according to the source code of DONeRF[9] and trained them using the default settings. We applied MIMO-NeRF with $N_p = 4$ to *DONeRF-16* (i.e., DONeRF with $N_s = 16$). In the self-supervised learning process, we used two formulations with $R^1 = R^2 = 1$ and group shifts. This model is referred to as *MIMO-DONeRF-16/4*. We set λ to 0.001. As a baseline, we examined DONeRF-4, which had the same # Run as MIMO-DONeRF-16/4. The implementation details are presented in Appendix B.2.

found that MIMO-NeRF-self achieved a better trade-off between speed and quality than the other variants. Detailed results are provided in Appendix A.4.

[9] https://github.com/facebookresearch/DONERF

Model	PSNR↑	FLIP↓	# Run↓	I-time↓ (s)	T-time↓ (h)	# Params (M)
DONeRF-16	33.06	0.061	17	0.429	3.79	0.94
DONeRF-4	31.21	0.070	5	0.140	3.23	0.94
MIMO-DONeRF-16/4-naive	32.30	0.063	5	0.155	3.26	0.99
MIMO-DONeRF-16/4-self	32.72	0.061	5	0.155	3.56	0.99

Table 3. Comparison of quantitative scores between DONeRFs and MIMO-DONeRFs. MIMO-DONeRF-16/4-naive outperformed DONeRF-4 in terms of PSNR and FLIP with a small increase in I-time and T-time. The performance of MIMO-DONeRF-16/4-self was close to DONeRF-16 in terms of PSNR and FLIP with shorter I-time and T-time.

Evaluation metrics. Following the study on DONeRF [41], we assessed the image quality using the *PSNR* and *FLIP* [1]. In addition, we used the *# Run*, *I-time*, *T-time*, and *# Params* described in Section 5.1. In DONeRF, # Run was calculated as $1 + \frac{N_s}{N_p}$.

Results. The results are summarized in Table 3. It is observed that MIMO-DONeRF-16/4-naive outperformed DONeRF-4 in terms of PSNR and FLIP with a small increase in I-time and T-time. MIMO-DONeRF-16/4-self enhanced the image quality with an increase in T-time, and its image quality approached that of DONeRF-16 in terms of PSNR and FLIP with faster inference and training. These results suggest that the increase in N_p (i.e., the replacement of the SISO MLP by the MIMO MLP) can be used as an alternative to the reduction in N_s (the number of selected samples) to obtain a better trade-off between speed and quality. A detailed analysis is presented in Appendix A.8.

5.5. Application to TensoRF

A NeRF with alternative representations is another representative fast approach. To demonstrate that MIMO-NeRF is also compatible with this model, we applied it to TensoRF [9], a representative model in this category. TensoRF uses a vector-matrix decomposition to calculate the volume densities and color features and applies a SISO MLP to the color features to decode the RGB colors. The corresponding ambiguity was relatively limited because the volume densities were extracted using an explicit representation. Hence, we simply replaced the SISO MLP with a MIMO MLP without modifying the training process while prioritizing training speed. These models are denoted as *MIMO-TensoRF-N_p*, where N_p was varied among $\{2, 4, 8\}$.

Dataset. We examined the performance of our approach on the *Blender* [39] and *LLFF* [38] datasets described in Section 5.1. Full-size images were used in this experiment.

Model	PSNR↑	SSIM↑	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF	33.23	0.963	9.95	1.25	11.50	18.8
MIMO-TensoRF-2	33.26	0.963	4.76	1.18	10.89	18.8
MIMO-TensoRF-4	32.98	0.961	2.40	1.15	10.67	18.8
MIMO-TensoRF-8	32.37	0.956	1.27	1.14	10.57	18.9

(a) Blender

Model	PSNR↑	SSIM↑	# Run↓	I-time↓ (s)	T-time↓ (m)	# Params (M)
TensoRF	26.73	0.837	126.73	6.64	23.41	46.8
MIMO-TensoRF-2	26.72	0.837	62.14	6.18	21.63	46.8
MIMO-TensoRF-4	26.72	0.836	30.16	5.76	21.15	46.8
MIMO-TensoRF-8	26.64	0.835	14.52	5.52	20.68	46.9

(b) LLFF

Table 4. Comparison of quantitative scores between TensoRF and MIMO-TensoRF. MIMO-TensoRF improved I-time and T-time while retaining PSNR and SSIM when $N_p \leq 2$ and $N_p \leq 4$ on the Blender and LLFF datasets, respectively.

Implementation. We implemented the models based on the official source code of TensoRF[10] and trained all the models using the same default settings for a fair comparison. The implementation details are presented in Appendix B.3.

Evaluation metrics. Following the study on TensoRF [9], we measured the image quality using PSNR and SSIM [64]. In addition, we used the *# Run*, *I-time*, *T-time*, and *# Params* described in Section 5.1. In TensoRF, # Run is determined adaptively for each pixel. Therefore, we report the average value.

Results. The results are presented in Table 4. It is observed that MIMO-TensoRF improved I-time and T-time with similar image quality when N_p was set within an adequate range (in particular, $N_p \leq 2$ on the Blender dataset and $N_p \leq 4$ on the LLFF dataset). These results suggest that MIMO-TensoRF can strengthen the inference and training speed of TensoRF without negative effects by adequately selecting N_p. A detailed analysis is presented in Appendix A.9.

6. Discussion

The results of these experiments in various situations demonstrate that MIMO-NeRF achieved a good trade-off between speed and quality. However, we also found that the quality degradation became significant with increasing N_p. One possible reason for this is that we did not modify the baseline network except for its input and output and did not increase the capacity of the models. It might be natural to implement a model of larger capacity to handle larger combinations of inputs and outputs. We did not adopt this strategy to ensure a fair comparison. However, searching for the best configurations considering the number of samples, the number of groups (the proposed new searching area), and the size of the model remain as a practically imperative and promising direction for further research.

[10] https://github.com/apchenstu/TensoRF

7. Conclusion

In this study, we have proposed MIMO-NeRF to improve the rendering speed of NeRF. Our core idea is that of replacing the SISO MLP used in standard NeRFs with a MIMO-MLP. We have developed a novel self-supervised learning method to address the ambiguity in the color and volume density of each point without relying on pretrained models. The results of an experimental evaluation have shown that MIMO-NeRF achieves a good trade-off between speed and quality with a reasonable training time. Although we have demonstrated the versatility of MIMO-NeRF by applying it to various NeRFs, many implicit neural representations aside from NeRFs also partially or primarily use SISO MLPs. We expect our ideas to be utilized with a few modifications to speed up the execution of such models.

References

[1] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. FLIP: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), 2020. 8

[2] Benjamin Attal, Jia-Bin Huang, Michael Zollhoefer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding networks. In *CVPR*, 2022. 3

[3] Matan Atzmon and Yaron Lipman. SAL: Sign agnostic learning of shapes from raw data. In *CVPR*, 2020. 3

[4] 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 *ICCV*, 2021. 3

[5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 3

[6] Rohan Chabra, Jan E. Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *ECCV*, 2020. 3

[7] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*, 2022. 2, 3

[8] 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 *CVPR*, 2021. 3

[9] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, 2022. 2, 3, 6, 8, 9

[10] Xingyu Chen, Qi Zhang, Xiaoyu Li, Yue Chen, Ying Feng, Xuan Wang, and Jue Wang. Hallucinated neural radiance fields in the wild. In *CVPR*, 2022. 3

[11] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 3

[12] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *NeurIPS*, 2020. 3

[13] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. GRAM: Generative radiance manifolds for 3D-aware image generation. In *CVPR*, 2022. 3

[14] Jiemin Fang, Lingxi Xie, Xinggang Wang, Xiaopeng Zhang, Wenyu Liu, and Qi Tian. NeuSample: Neural sample field for efficient view synthesis. *arXiv preprint arXiv:2111.15552*, 2021. 2, 3

[15] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 2, 3

[16] Guy Gafni, Justus Thies, Michael Zollhofer, and Matthias Nießner. Dynamic neural radiance fields for monocular 4D facial avatar reconstruction. In *CVPR*, 2021. 3

[17] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200FPS. In *ICCV*, 2021. 2, 3

[18] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, 2019. 3

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 3

[20] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *ICML*, 2020. 3

[21] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. StyleNeRF: A style-based 3D-aware generator for high-resolution image synthesis. In *ICLR*, 2022. 3

[22] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 2, 3

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 3

[24] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. EfficientNeRF: Efficient neural radiance fields. In *CVPR*, 2022. 2, 3

[25] Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3D scenes. In *CVPR*, 2020. 3

[26] Takuhiro Kaneko. AR-NeRF: Unsupervised learning of depth and defocus effects from natural images with aperture rendering neural radiance fields. In *CVPR*, 2022. 3

[27] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. AdaNeRF: Adaptive sampling for real-time rendering of neural radiance fields. In *ECCV*, 2022. 2, 3

[28] Chen-Hsuan Lin, Chaoyang Wang, and Simon Lucey. SDF-SRN: Learning signed distance 3D object reconstruction from static images. In *NeurIPS*, 2020. 1, 3

[29] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. In *CVPR*, 2021. 2, 3

[30] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 2, 3

[31] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3D supervision. In *NeurIPS*, 2019. 3

[32] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. DIST: Rendering deep implicit signed distance function with differentiable sphere tracing. In *CVPR*, 2020. 3

[33] 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 *CVPR*, 2021. 3

[34] Nelson Max. Optical models for direct volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 1(2), 1995. 1, 3, 4

[35] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 3

[36] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *ICCV*, 2019. 3

[37] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. NeRF in the dark: High dynamic range view synthesis from noisy raw images. In *CVPR*, 2022. 3

[38] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 38(4), 2019. 6, 8

[39] 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 *ECCV*, 2020. 1, 3, 6, 8

[40] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), 2022. 2, 3

[41] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards real-time rendering of compact neural radiance fields using depth oracle networks. *Comput. Graph. Forum*, 40(4), 2021. 2, 3, 6, 8

[42] Michael Niemeyer and Andreas Geiger. CAMPARI: Camera-aware decomposed generative neural radiance fields. In *3DV*, 2021. 3

[43] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 3

[44] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *CVPR*, 2020. 1, 3

[45] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 3

[46] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020. 3

[47] Martin Piala and Ronald Clark. TermiNeRF: Ray termination prediction for efficient neural rendering. In *3DV*, 2021. 2, 3

[48] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *ICLR*, 2023. 3

[49] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *CVPR*, 2021. 3

[50] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. DeRF: Decomposed radiance fields. In *CVPR*, 2021. 2, 3

[51] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *ICCV*, 2021. 2, 3

[52] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*, 2019. 3

[53] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3D-aware image synthesis. In *NeurIPS*, 2020. 3

[54] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020. 3

[55] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *NeurIPS*, 2021. 1, 3

[56] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019. 1, 3

[57] Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. EpiGRAF: Rethinking training of 3D GANs. In *NeurIPS*, 2022. 3

[58] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, 2019. 3

[59] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *CVPR*, 2022. 3

[60] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 2, 3

[61] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020. 3

[62] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 3

[63] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. R2L: Distilling neural radiance field to neural light field for efficient novel view synthesis. In *ECCV*, 2022. 3

[64] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4), 2004. 6, 9

[65] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 2, 3

[66] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. DIVeR: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *CVPR*, 2022. 2, 3

[67] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *CVPR*, 2021. 3

[68] Yang Xue, Yuheng Li, Krishna Kumar Singh, and Yong Jae Lee. GIRAFFE HD: A high-resolution 3D-aware generative model. In *CVPR*, 2022. 3

[69] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *NeurIPS*, 2020. 1, 3

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

[71] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 1, 3

[72] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 3

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