Exploring 3D-aware Latent Spaces for Efficiently Learning Numerous Scenes

Antoine Schnepf [*1,3] Karim Kassab [*1,2] Jean-Yves Franceschi[1] Laurent Caraffa[2]
Flavian Vasile[1] Jeremie Mary[1] Andrew Comport[3] Valérie Gouet-Brunet[2]

[*] Equal contributions
[1] Criteo AI Lab, Paris, France
[2] LASTIG, Université Gustave Eiffel, IGN-ENSG, F-94160 Saint-Mandé
[3] Université Côte d'Azur, CNRS, I3S, France

Abstract

We present a method enabling the scaling of NeRFs to learn a large number of semantically-similar scenes. We combine two techniques to improve the required training time and memory cost per scene. First, we learn a 3D-aware latent space in which we train Tri-Plane scene representations, hence reducing the resolution at which scenes are learned. Moreover, we present a way to share common information across scenes, hence allowing for a reduction of model complexity to learn a particular scene. Our method reduces effective per-scene memory costs by 44% and per-scene time costs by 86% when training 1000 scenes. Our project page can be found at https://3da-ae.github.io.

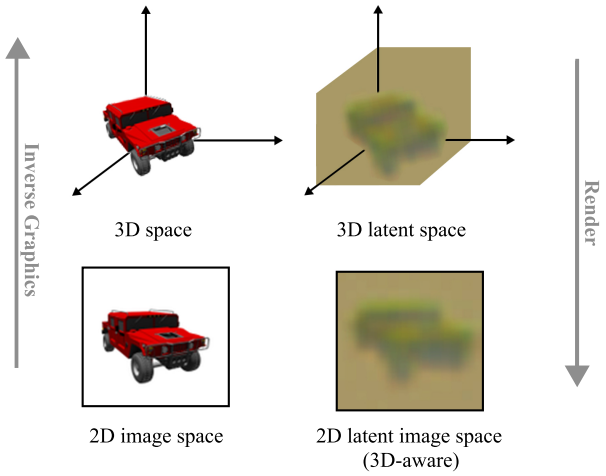


Figure 1. **3D-aware latent space.** We draw inspiration from the relationship between the 3D space and image space and introduce the idea of a 3D latent space. We propose a 3D-aware autoencoder that encodes images into a 3D-aware (2D) latent image space, in which we train our scene representations.

1. Introduction

The inverse graphics problem has proven to be a challenging quest in the domain of Computer Vision. While many methods have historically emerged [9, 21, 24, 25], the question has mostly remained unchanged: how to model an object or scene, using only its captured images? While this question continues to be an active area of research, our work targets a scaled version of the original problem: how to model abundantly many objects at once? This context is dissimilar to independently learning scenes, as one could exploit the inherent nature of the problem — learning *many* scenes — to mitigate per-scene optimization costs. Although modeling 3D objects and scenes from captured images has seen various applications (e.g. Virtual Reality, Robotics, Autonomous Navigation), scaling this problem to large amounts of objects unlocks new ways in which 3D modeling techniques could be leveraged (e.g. modeling an inventory of products for commerce, integrating real artifacts in virtual spaces).

In this paper, we introduce a novel technique enabling the scaling of the inverse graphics problem. To do so, our work aims to compress scene representations to model only essential information within a particular scene. To achieve this, we build on two main ideas: we learn a **3D-aware (2D) latent space** in which we train our scene representations, and we introduce **cross-scene feature sharing** to avoid redundantly learning similar information.

Latent spaces enable the representation of high-resolution RGB images in a compact form, through the process of encoding and decoding. Particularly, Auto-Encoders (AEs) learn a lower-dimensional latent space able to capture the underlying structure and diversity of a dataset of images. However, as their optimization is devoid of geometrical constraints, latent spaces generally lack 3D structure, which is particularly crucial in 3D applications as this promotes

a discrepancy between the latent space and the real world images it represents. One particular downfall of this discrepancy is the absence of 3D-consistency between two latent images encoding two 3D-consistent images. In this work, we present a novel approach to regularize a latent space with geometric constraints by leveraging Tri-Planes representations [5]. This not only adds 3D consistency to the latent space, but also enables the use of recent neural rendering techniques within this latent space, which accelerates their optimization and rendering times, and lowers their memory footprint. Fig. 1 illustrates the intuition behind our 3D latent space, and the analogy with the natural 3D space.

To go further, we also minimize the amount of information we learn per scene representation by introducing the notion of **globally shared representations**. Here, we train M representations that are shared among scenes, and that learn *global* information about the scenes within the dataset at hand. This avoids learning redundant information across scenes and hence minimizes both the per-scene compute time and memory cost when learning numerous scenes.

A summary of our contribution can be found below:

- We build a 3D-aware latent space in which neural scene representations can be trained,
- We present an approach to minimize the capacity needed to model a latent scene by sharing common globally-trained scene representations across scenes,
- Our work can learn 1000 scenes with 86% less time and 44% less memory than our base representation.

2. Related Work

NeRF resource reductions. Neural Radiance Fields [21, NeRF] achieve impressive performances on the task of Novel View Synthesis (NVS) by adopting a purely implicit representation to model scenes. On one end of the spectrum, some NeRF methods [2, 3] achieve exceptional quality while requiring low memory capacity to store scenes, as they represent scenes through the weights of neural networks. This however comes with the sacrifice of high training and rendering times due to bottlenecks in volume rendering. To alleviate these issues, some representations trade-off compute time for memory usage by explicitly storing proxy features for the emitted radiances and densities in a 3D data structure (e.g. voxel based representations [8, 23, 29, 34] or plane-based representations [4, 5, 12]). This allows for the use of a significantly smaller neural network as compared to purely implicit representations. On the other end of the spectrum, Fridovich-Keil et al. [11], Kerbl et al. [14] completely forgo the use of neural networks, achieving real-time rendering, but at high memory costs.

In contrast to previous works, we propose a method orthogonal to the aforementioned spectrum and sidestepping the time-memory trade-off; by accelerating scene learning all the while lowering the memory footprint of individual scenes. This is partly done by training our scene representations on a compressed version of training images. To do so, we present a novel 3D-aware latent feature space.

Neural Feature Fields. Neural Feature Fields extend Neural Radiance Fields to render feature images instead of color images, while using the same volume rendering equations. First methods exploring feature fields propose the 3D distillation of features by jointly training a radiance field and a feature field. This allows for modeling an *individual* scene in a feature field rather than a radiance field. This is particularly interesting, as this representation unlocks many subsequent applications, like 3D object detection and segmentation [31], 3D editing [16, 18] and semantic understanding of scenes [15, 19]. Metzer et al. [20], Seo et al. [27] train a feature field that renders in the latent space of Stable Diffusion [26]. These feature fields are optimized such that their renderings match the posterior distribution of Stable Diffusion under a descriptive text prompt, hence enabling text-to-3D generation. Chan et al. [6], Ye et al. [32], Yu et al. [33] use a pre-trained encoder to generate feature fields from single images, hence enabling single-image-to-3D generation. Aumentado-Armstrong et al. [1] are closest to our work, as they model individual scenes in a latent feature space with a decoder neural network, thus accelerating volume rendering thanks to the reduced latent-space dimension. However, this training is individually done for each scene, and no common latent space is learned across scenes.

Our work expands [1] in a different scope, more particularly by leveraging both an encoder and a decoder to learn a *common* 3D-consistent latent space in which numerous scene representations can be trained.

Meta-Learning base networks for INRs. To reduce model complexity, previous works have explored learning and modulating shared base networks to represent common structure within a set of signals modeled by Implicit Neural Representations (INRs). More particularly, to create a *functaset* of NeRFs, Dupont et al. [10] learn a shared base network thanks to which NeRFs can be trained by only optimizing modulations of the base network. Our work draws inspiration from this and introduces global Tri-Planes, which consist of shared scene representations storing common information and structure across a set of scenes. This enables in our case the reduction of per-scene model complexity. Previous works have also shown that meta-learning such shared networks enables fitting Neural Radiance Fields and Signed Distance Functions in only a few optimization steps [28, 30]. More generally, previous works have also shown the advantage of learning priors over a subset of scenes. [10, 17, 22, 25]. Our work parallels this by pre-training parts of our pipeline on the training set, hence learning a prior over the scene distribution before applying it on new scenes.

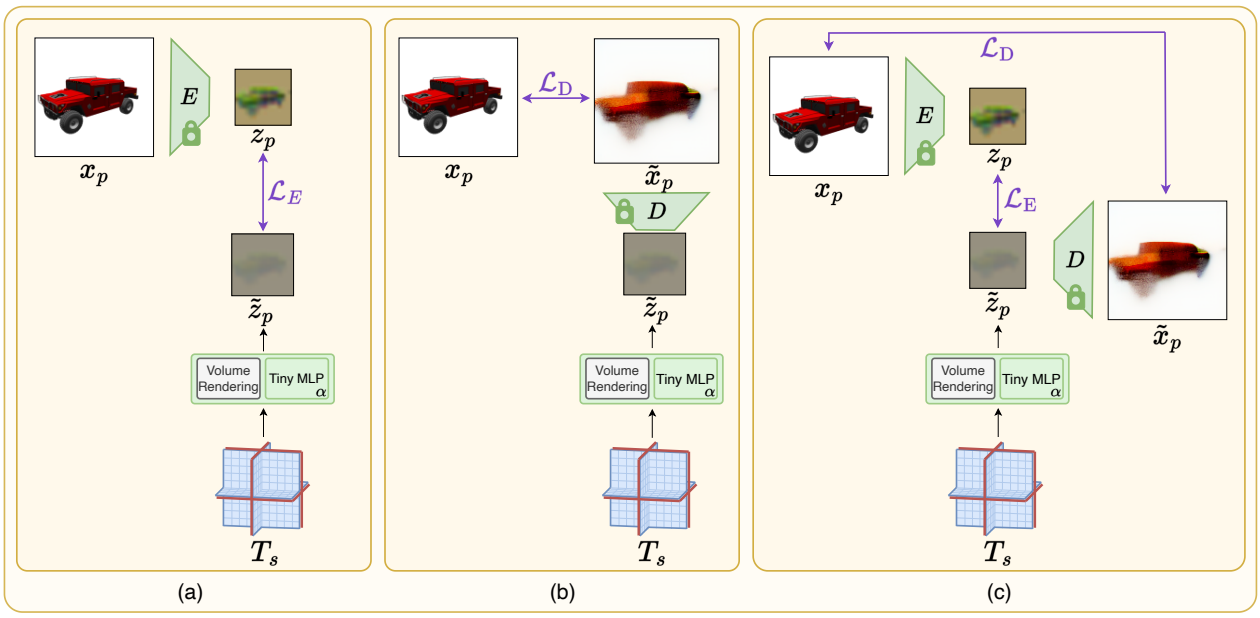


Figure 2. **Methods for learning scenes in a 3D-aware latent space.** Diagrams for (a) Encode-Scene, (b) Decode-Scene, and (c) Encode-Decode-Scene, the proposed methods to train Tri-Plane scene representations in a 3D-aware latent space.

3. Method

In this section, we present the main components of our method and elucidate the intuition behind our choices. We start by presenting three methods with which one could train a scene representation in a 3D-aware latent space: **Encode-Scene**, **Decode-Scene**, and **Encode-Decode-Scene**. We subsequently present our 3D-aware Autoencoder (**3Da-AE**) which builds upon these methods to learn a 3D-aware latent space. We also present **Micro-Macro representations**, which constitute an additional component in our pipeline that further accelerates training while lowering its memory footprint thanks to information sharing. Finally, we show how to use 3Da-AE and Micro-Macro Tri-Planes to solve the problem of learning numerous scenes.

3.1. Prerequisites

Autoencoder. An autoencoder (AE) is a compression model trained to learn efficient, low-resolution representations of images. To achieve this, the model is trained via a reconstruction loss where images are passed through an information bottleneck:

$$\begin{aligned} z &= E_\psi(x)\,, \\ \hat{x} &= D_\phi(z)\,, \\ \mathcal{L}_{\text{ae}}(\psi,\phi) &= \mathbb{E}_x \|x-\hat{x}\|_2^2\,, \end{aligned} \tag{1}$$

where x is an image, E_ψ and D_ϕ respectively represent the encoder and decoder with trainable parameters ψ and ϕ, and z has a lower resolution than x. In this work, we use the autoencoder of Stable Diffusion [26] as a baseline. It works with a large range of input image resolutions, while reducing the resolution by a factor of 64 in the latent space.

3D consistency. The notion of 3D consistency refers to the underlying 3D geometry of 2D images. Formally, 3D consistency involves ensuring that corresponding points or features in different images represent the same physical point or object in the scene, despite variations in viewpoint, lighting or occlusion. Note that while 3D consistency is natural for a set of posed images $\mathcal{X}_s = \{p, x_p\}_{p\in\mathcal{P}}$ obtained from a scene s in the image space, it does not naturally extend to the latent space, as latent representations of two 3D-consistent images are not necessarily 3D consistent (Fig. 3, bottom row).

Tri-Plane representation. Tri-Plane representations [5] are explicit-implicit scene representations enabling scene modeling in three axis-aligned orthogonal feature planes, each of resolution $T\times T$ with feature dimension F. To query a 3D point $x \in \mathbb{R}^3$, it is projected onto each of the three planes to retrieve bilineraly interpolated feature vectors F_{xy}, F_{xz} and F_{yz}. These feature vectors are then aggregated via summation and passed into a small neural network to retrieve the corresponding color and density, which are then used for volume rendering [13]. We adopt Tri-Plane representations for their relatively fast training times, as well as their explicit nature enabling their modularity, an essential property for our Micro-Macro decomposition (Sec. 3.4).

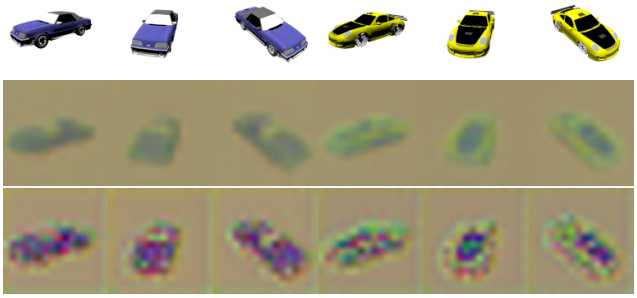


Figure 3. **Latent space comparison.** Top: ground truth image. Middle: latent image obtained with the 3D-aware encoder. Bottom: latent image obtained with the baseline encoder. Qualitative results show that our 3D-aware encoder better preserves 3D consistency and geometry in the latent space.

3.2. Latent NeRFs

We define a latent scene representation similarly to a classical scene representation, except that it is trained on latent encodings of the scene images. In this section, we assume the existence of a 3D-aware latent space, and present three approaches to learn latent scene representations: **Encode-Scene**, **Decode-Scene**, and **Encode-Decode-Scene**, respectively utilizing the encoder, decoder and both modules of a 3D-aware autoencoder. Fig. 2 illustrates these methods.

3.2.1 Encode-Scene

Encode-Scene takes the most similar approach compared to classical NeRF training in order to train NeRFs in a latent space. It is a two-step process where latent images are first obtained from training images using the encoder, and are then used to train the latent NeRF with the usual NeRF photometric loss:

$$\min_{\alpha,T} \mathcal{L}_E(\alpha, T) \triangleq \mathbb{E}_{x_p} \|E(x_p) - \mathcal{R}_\alpha(T, p)\| \,, \qquad (2)$$

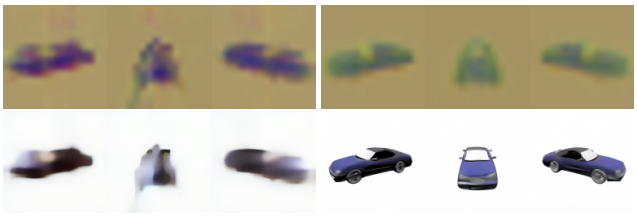
where $E(x_p)$ is the encoding of an image x_p with camera pose p, and $\mathcal{R}_\alpha(T, p)$ represents the rendered image from the Tri-Plane T queried at pose p. Note that the main advantage of this approach is its accelerated training and rendering procedures, as all the training images can be encoded into their latent representations once, and then cached. As these latent representations are 64 times smaller in resolution, the rendering algorithm has to query 64 times less pixels, which greatly accelerates rendering and hence the training.

3.2.2 Decode-Scene

Decode-Scene takes a different approach to train NeRFs in the latent space. Here, we use the decoder and supervise the pipeline with RGB images, all while keeping the NeRF in the latent space. More particularly, the NeRF here is tasked to find a 3D-consistent latent object that, when rendered into latent images, decodes to its corresponding RGB images. Hence, the NeRF here is supervised via a photometric loss computed in the RGB space:

$$\min_{\alpha,T} \mathcal{L}_D(\alpha, T) \triangleq \mathbb{E}_{x_p} \|x_p - D(\mathcal{R}_\alpha(T, p))\| \,, \qquad (3)$$

where D represents the decoder. While rendering is still applied in the latent space in Decode-Scene, this is generally a slower method compared to Encode-Scene. This is because no latent images can be cached, and a gradient step requires differentiation through all the parameters of the decoder.



(a) Standard AE (b) 3D-aware AE

Figure 4. **Latent scenes comparison**. Visualization of Tri-Planes renderings and their corresponding decodings after learning scenes in the latent space of a standard AE and our 3D-aware AE. All Tri-Planes are trained using the Encode-Scene pipeline.

3.2.3 Encode-Decode-Scene

Encode-Decode-Scene is a mixture of the former approaches, where both $\mathcal{L}_E$ and $\mathcal{L}_D$ are used to train the latent NeRF:

$$\min_{\alpha,T} (1-t)\mathcal{L}_D(\alpha, T) + t\mathcal{L}_E(\alpha, T) \,, \qquad (4)$$

where $t \in (0, 1)$. This method takes the best of both worlds by supervising both the rendered latent and the decoded image to train a scene representation, ensuring good latent scene representations and high-fidelity color images.

Qualitative results obtained when training Tri-Planes with the baseline AE and our 3D-aware AE are illustrated in Fig. 4. While learning scenes in a standard latent space is feasible, it results in poor scene quality, as the latent images used for inverse graphics are not 3D-consistent (Fig. 3, bottom row).

3.3. 3D-aware AE for a 3D-aware latent space

As presented, NeRFs trained in a standard latent space suffer from its 3D inconsistencies. To fix this issue, we fine-tune the autoencoder of Stable Diffusion [26] to obtain a 3D-aware Autoencoder (3Da-AE).

To enforce 3D consistency in the latent space, we regularize it by training NeRFs on its latent images. Intuitively, this encourages the encoder and the decoder to preserve 3D

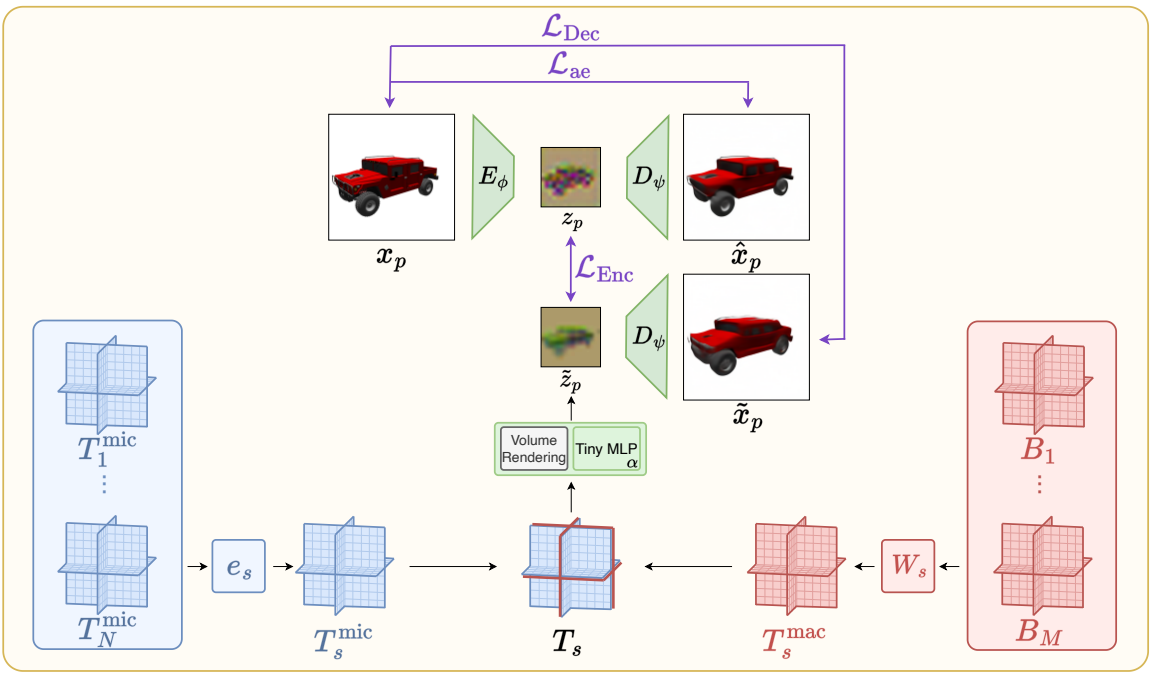


Figure 5. **3Da-AE training.** We learn a 3D-aware latent space by regularizing its training with 3D constraints. To this end, we jointly train the encoder E_ϕ, the decoder D_ψ and N scenes in this latent space. For each scene s, we learn a Tri-Planes representation T_s, built from the concatenation of local Tri-Planes T_s^{mic} and global Tri-Planes T_s^{mac}. T_s^{mic} is retrieved via a one-hot vector e_s from a set of scene-specific planes stored in memory. T_s^{mac} is computed from a summation of M globally shared Tri-Planes, weighted with weights W_s.

consistency, thanks to the NeRF's innate 3D-consistent nature. To do so, we present a joint training process where we optimize N_{train} Tri-Planes modeling latent scenes, as well as the encoder and decoder. Fig. 3 (middle row) illustrates the preservation of consistency of our method.

To achieve this joint training, we implement three losses, inspired from Sec. 3.2, in order to simultaneously train the encoder, the decoder, as well as the latent NeRFs. First, an autoencoder preservation loss $\mathcal{L}_{\text{ae}}$ ensures the correct reconstruction of images after encoding and decoding. Second, $\mathcal{L}_{\text{Enc}}$ reassembles $\mathcal{L}_E$ from Encode-Scene, but additionally supervises the encoder so that it produces 3D consistent latent images. Third, $\mathcal{L}_{\text{Dec}}$ reassembles $\mathcal{L}_D$ from Decode-Scene, but additionally trains the decoder to ensure that two 3D-consistent latent images are decoded into their corresponding 3D-consistent RGB images. Hence, the training objective for the 3Da-AE is written as:

$$\min_{\phi,\psi,\alpha,T} \lambda_{\text{ae}}\mathcal{L}_{\text{ae}}(\phi,\psi) + \lambda_{\text{Enc}}\mathcal{L}_{\text{Enc}}(\phi,\alpha,T) + \lambda_{\text{Dec}}\mathcal{L}_{\text{Dec}}(\psi,\alpha,T)\,,$$

$$\text{with} \begin{cases} \mathcal{L}_{\text{ae}}(\phi,\psi) = \mathbb{E}_{x_p}\|x_p - D_\psi(E_\phi(x_p))\|\,, \\ \mathcal{L}_{\text{Enc}}(\phi,\alpha,T) = \mathbb{E}_{x_p}\|E_\phi(x_p) - \mathcal{R}_\alpha(T,p)\|\,, \\ \mathcal{L}_{\text{Dec}}(\psi,\alpha,T) = \mathbb{E}_{x_p}\|x_p - D_\psi(\mathcal{R}_\alpha(T,p))\|\,, \end{cases} \tag{5}$$

where ϕ, ψ and α are shared parameters among all scenes. T consists of Tri-Plane parameters which we divide into scene-specific local parameters and globally-shared computed parameters. For an overview of the full 3Da-AE pipeline, we refer the reader to Fig. 5.

3.4. Micro-Macro Tri-Plane Decomposition

In this section, we present an additional approach to reduce the capacity needed to learn individual scenes through Tri-Plane scene representations. As most of our scenes share similar structure, we sidestep repeatedly learning redundant information across scenes by integrating globally shared information into our scene representations, and modulating this shared information via scene-dependent weights. While learning scenes in the latent space achieves complexity reductions through minimizing spatial resolutions, we aim to achieve this here by decomposing a scene representation into "globally" shared information and "locally" learned features.

Formally, we decompose a Tri-Plane representation T_s modeling a scene s into a locally trained Tri-Plane representation T_s^{mic} and a globally learned representation T_s^{mac}:

$$T_s = T_s^{\text{mic}} \oplus T_s^{\text{mac}}\,, \tag{6}$$

where $\oplus$ concatenates two Tri-Planes along the feature dimension. We denote by F^{mic} the number of local feature

Algorithm 1: 3Da-AE Training.

Input: $\mathcal{X}_{\text{train}}$, E_ϕ, D_ψ, $\mathcal{R}_\alpha$, N, λ_{ae}, λ_{Enc}, λ_{Dec}, optimizer

Random initialization: $T^{\text{mic}}, W, T^{\text{mac}}$

for N *steps* **do**
  for $\{s, p, x_p\}$ *in* shuffle($\mathcal{X}_{\text{train}}$) **do**
    // Compute local planes
    T_s^{mic} , $T_s^{\text{mac}} \leftarrow e_s T^{\text{mic}}$, $W_s B$
    $T_s \leftarrow T_s^{\text{mic}} \oplus T_s^{\text{mac}}$
    // Encode, Decode, Render
    $z_p \leftarrow E_\phi(x_p)$
    $\hat{x}_p \leftarrow D_\psi(z_p)$
    $\tilde{z}_p \leftarrow \mathcal{R}_\alpha(T_s, p)$
    $\tilde{x}_p \leftarrow D_\psi(\tilde{z}_p)$
    // Compute losses and optimize
    $\mathcal{L}_{\text{ae}} \leftarrow \|x_p - \hat{x}_p\|_2^2$
    $\mathcal{L}_{\text{Enc}} \leftarrow \|z_p - \tilde{z}_p\|_2^2$
    $\mathcal{L}_{\text{Dec}} \leftarrow \|x_p - \tilde{x}_p\|_2^2$
    $\mathcal{L} \leftarrow \lambda_{\text{ae}}\mathcal{L}_{\text{ae}} + \lambda_{\text{Enc}}\mathcal{L}_{\text{Enc}} + \lambda_{\text{Dec}}\mathcal{L}_{\text{Dec}}$
    $T_s^{mic}, W_s, B, \alpha, \phi, \psi \leftarrow$ optimizer.step($\mathcal{L}$)

in T_s^{mic} and by F^{mac} the number of global feature in T_s^{mac}, with the total number of feature $F = F^{\text{mic}} + F^{\text{mac}}$.

For T_s^{mac} to represent globally captured information, it is computed for each scene from globally learned Tri-Plane representations $\{B_i\}_{i=1}^M$ by the means of a weighted sum:

$$T_s^{\text{mac}} = W_s B = \sum_{i=1}^{M} w_s^i B_i \ , \tag{7}$$

where W_s are learned coefficients for scene s, and B_i are jointly trained with every scene. This approach accelerates our method and reduces its memory footprint, as we asymptotically reduce the number of trainable features by a factor of $\frac{F^{\text{mac}}}{F^{\text{mic}}+F^{\text{mac}}}$.

3.5. Scaling 3Da-AE

In order to leverage our 3Da-AE pipeline to train a large number of NeRFs, we utilize the ideas presented in Secs. 3.3 and 3.4 within two stages: "**Training 3Da-AE**" (Sec. 3.5.1) and "**Exploiting 3Da-AE**" (Sec. 3.5.2). The objective of the first stage is to train 3Da-AE and the global planes in order to obtain our 3D-aware latent space. The goal in the second stage is to learn numerous scenes by using the 3D-aware latent space obtained in the first step, as well as the global planes. We detail the two stages in the following sections.

3.5.1 Training 3Da-AE

In this stage, we learn a 3D-aware autoencoder. Our training dataset $\mathcal{X}_{\text{train}} = \{(s, p, x_p)\}_{s\in\mathcal{S},\ p\in\mathcal{P}}$, where $\mathcal{S}$ denotes the set of scene indices and $\mathcal{P}$ the set of poses, is composed of

Algorithm 2: 3Da-AE Exploitation.

Input: $\mathcal{X}_{\text{exploit}}$, E_ϕ, D_ψ, B, $\mathcal{R}_\alpha$, N_1, N_2, optimizer

Random initialization: T^{mic}, W

// Encode-Scene
for N_1 *steps* **do**
  for $\{s, p, x_p\}$ *in* shuffle($\mathcal{X}_{\text{exploit}}$) **do**
    // Compute local planes
    T_s^{mic} , $T_s^{\text{mac}} \leftarrow e_s T^{\text{mic}}$, $W_s B$
    $T_s \leftarrow T_s^{\text{mic}} \oplus T_s^{\text{mac}}$
    // Encode, Render
    $z_p \leftarrow E_\phi(x_p)$
    $\tilde{z}_p \leftarrow \mathcal{R}_\alpha(T_s, p)$
    // Compute losses and optimize
    $\mathcal{L}_{\text{E}} \leftarrow \|z_p - \tilde{z}_p\|_2^2$
    $T_s^{mic}, W_s, B, \alpha \leftarrow$ optimizer.step($\mathcal{L}_{\text{E}}$)
// Decoder finetuning
for N_2 *steps* **do**
  for $\{s, p, x_p\}$ *in* shuffle($\mathcal{X}_{\text{exploit}}$) **do**
    // Compute local planes
    T_s^{mic} , $T_s^{\text{mac}} \leftarrow e_s T^{\text{mic}}$, $W_s B$
    $T_s \leftarrow T_s^{\text{mic}} \oplus T_s^{\text{mac}}$
    // Encode, Decode, Render
    $\tilde{z}_p \leftarrow \mathcal{R}_\alpha(T_s, p)$
    $\tilde{x}_p \leftarrow D_\psi(\tilde{z}_p)$
    // Compute losses and optimize
    $\mathcal{L}_{\text{Dec}} \leftarrow \|x_p - \tilde{x}_p\|_2^2$
    $T_s^{mic}, W_s, B, \alpha, \psi \leftarrow$ optimizer.step($\mathcal{L}_{\text{Dec}}$)

N_{train} scenes from ShapeNet [7]. We initialize the local Tri-Planes $\{T_s^{\text{mic}}\}_{s\in\mathcal{S}}$, as well as our global Tri-Planes $\{B_i\}_{i=1}^M$ and the scene-specific coefficients $\{W_s\}_{s\in\mathcal{S}}$. For each scene s, we compute the corresponding representation T_s with the Micro-Macro decomposition (Eq. (7)).

Given a posed image (p, x_p) of a scene s, we use the autoencoder to obtain the latent image z_p and reconstructed image $\hat{x}_p$. Subsequently, we render the triplane T_s from pose p to obtain the rendered latent $\tilde{z}_p$, which we decode to obtain $\tilde{x}_p$. The losses $\mathcal{L}_{\text{ae}}$, $\mathcal{L}_{\text{Dec}}$ and $\mathcal{L}_{\text{Enc}}$ are estimated on a mini-batch and used to optimize the encoder, decoder, local Tri-Planes, global Tri-Planes and the scene-specific coefficients. Algorithm 1 details our training procedure.

Note that in practice, we begin this training with a warmup stage, during which only the local triplanes, global triplanes, and scene-specific coefficients are optimized. Here, we keep the autoencoder frozen as random gradients would back-propagate into E_ϕ and D_ψ.

3.5.2 Exploiting 3Da-AE

In this stage, we learn scenes using the 3D-aware latent space and the global Tri-Planes obtained in the former stage. Our exploitation dataset $\mathcal{X}_{\text{exploit}}$ is composed of N_{exploit}

	t_{scene} (min)	$t^{\text{eff}}_{\text{scene}}$ (min)	m_{scene} (MB)	$m^{\text{eff}}_{\text{scene}}$ (MB)	Rendering Time (ms)	Rendering Resolution
Encoder	—	—	0	0.13	—	—
Decoder	—	—	0	0.19	9.7	128×128
Tri-Planes (RGB)	32	32	1.5	1.5	23.3	128×128
Our method	2	4.5	0.48	0.84	11.0	128×128

Table 1. **Cost comparison.** Per scene cost comparison with Tri-Planes trained in the image space. Here, we consider N_{train} = 500, $N_{\text{exploit}} = 1000$, t_{EC} = 40 hours, $M = 50$, $F^{\text{mac}} = 22$. Our method reduces the effective training time by 86% per scene, and the effective memory cost by 44% per scene.

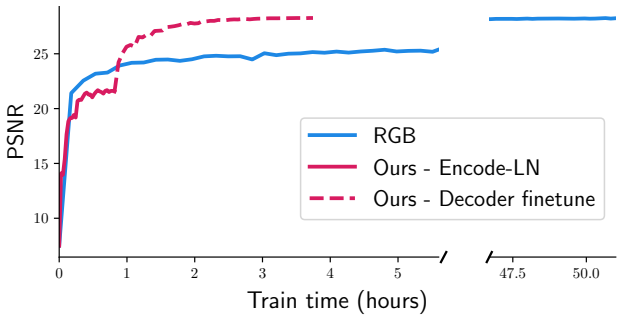


Figure 6. **Quality evolution.** Evolution of the average test-view PSNR demonstrated in the exploit phase of our method compared to RGB Tri-Planes ($N_{\text{exploit}} = 100$). Our method achieves comparable quality in less training time.

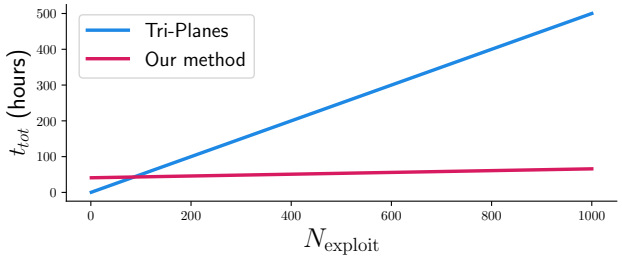


Figure 7. **Time cost evolution.** Total train time evolution when scaling the number of trained scenes N_{exploit}. The entry training cost t_{EC} is taken into account. Our method demonstrates more favorable scalability properties as compared to Tri-Planes (RGB).

scenes. We start by randomly initializing our local Tri-Planes and training our scene representations with **Encode-Scene**. Finally, to maximize the quality of the decoded RGB images, we finish this stage by a small fine-tuning of the decoder with an $\mathcal{L}_{\text{Dec}}$ loss, similarly to **Decode-Scene**, but with a trainable decoder. Algorithm 2 details our exploitation procedure.

4. Experiments

In this section, we present the resource costs and quality evaluations of our method. We also present an ablation study to assess the added value of each element of our pipeline.

Dataset. We adopt the ShapeNet-Cars [7] dataset. Each scene s is rendered at resolution 128×128 with 200 different camera poses, from which we take 90% for training and 10% for testing. We use two subsets of scenes respectively for training our 3Da-AE, and exploiting it to learn numerous scenes. For each scene, we divide the views into train and test views as to evaluate the NVS performances of our method.

Implementation details. We train the 3Da-AE on $N_{\text{train}} = 500$ scenes from ShapeNet-Cars. For the exploitation phase, we learn $N_{\text{exploit}} = 1000$ scenes. For all our experiments, we take $F^{\text{mic}} = 10$, $F^{\text{mac}} = 22$, and $M = 50$.

4.1. Resource costs

In this section, we detail the resource costs in terms of time and memory of the various stages of our method, and illustrate how we compare it to classical Tri-Plane training.

Time costs. As presented, our method starts by warming up the Tri-Planes and training the 3Da-AE with N_{train} scenes, for which we respectively allocate the times $t^{\text{warmup}}_{\text{train}}$ and t_{train}. Subsequently, we exploit the 3Da-AE to train N_{exploit} scenes. We call t_{exploit} and $t^{\text{ft}}_{\text{exploit}}$ respectively the time to train the Tri-Planes in the exploit phase, and the time to fine-tune the decoder at the end of this phase. Note that $t^{\text{warmup}}_{\text{train}}$ and t_{train} represent a time entry cost for our method, as this training is done only once, and it is independent of the number of scenes N_{exploit} which we wish to learn. We denote this entry cost time by $t_{\text{EC}} = t^{\text{warmup}}_{\text{train}} + t_{\text{train}}$. Thus, our total training time is written as:

$$t_{\text{tot}} = t_{\text{EC}} + N_{\text{exploit}} t_{\text{scene}} \ , \tag{8}$$

where $t_{\text{scene}} = \frac{1}{N_{\text{exploit}}}(t_{\text{exploit}} + t^{\text{ft}}_{\text{exploit}})$ is the training time per scene in the exploit phase. Finally, for a fair comparison to training in the RGB space, we define $t^{\text{eff}}_{\text{scene}}$, the effective time taken per scene in our method. This takes into account

Experiment	Latent Space	Micro-Planes	Macro-Planes	Train scenes	Exploit scenes
Ours-Micro	✓	✓	✗	26.52	26.95
Ours-Macro	✓	✗	✓	25.67	26.10
Tri-Planes-Macro (RGB)	✗	✗	✓	27.84	28.00
Tri-Planes (RGB)	✗	✓	✗	**28.24**	28.40
Ours-No-Prior	✓	✓	✓	27.72	28.13
Ours	✓	✓	✓	28.05	**28.48**

Table 2. **Quality comparison.** Average PSNR demonstrated by our method with a comparison to Tri-Planes and ablations of our pipeline. All metrics are computed on never-seen test views. Here, we consider $N_{\text{train}} = 500$, $N_{\text{exploit}} = 100$, and $M = 50$. For compute constraints, Tri-Planes metrics are averaged on 50 scenes.

t_{EC}, the entry cost to our method. $t^{\text{eff}}_{\text{scene}}$ is written as:

$$t^{\text{eff}}_{\text{scene}} = \frac{t_{\text{EC}}}{N_{\text{exploit}}} + t_{\text{scene}} \ . \tag{9}$$

Indeed, our method is more beneficial when N_{exploit} is large.

Memory costs. In terms of memory footprint, our method presents an advantage compared to its baseline as it requires less local Tri-Plane features per scene. We denote m_E, m_D and m_B, the memory size required to respectively save the encoder, decoder, and the global planes. We also define $m_{\text{EC}} = m_E + m_D + m_B$, the memory entry cost needed for our method. Additionally, saving the scenes requires saving their local Tri-Planes of size m_T and their coefficients of size m_W. Thus, our total memory footprint is written as:

$$m_{\text{tot}} = m_{\text{EC}} + N_{\text{exploit}} m_{\text{scene}} \ , \tag{10}$$

where $m_{\text{scene}} = \frac{1}{N_{\text{exploit}}}(m_T + m_W)$ is the memory size needed to save one scene. Lastly, we define $m^{\text{eff}}_{\text{scene}}$, the effective memory size needed per scene, taking into account the entire pipeline. $m^{\text{eff}}_{\text{scene}}$ is written as:

$$m^{\text{eff}}_{\text{scene}} = \frac{m_{\text{EC}}}{N_{\text{exploit}}} + m_{\text{scene}} \ . \tag{11}$$

Similarly to the time costs, our method is also more advantageous with larger N_{exploit}.

4.2. Evaluations

We apply our exploitation phase on two sets of scenes: scenes from the training set, and held-out scenes from the exploit set. We compare our results with a classical training of Tri-Planes in the image space, denoted **"Tri-Planes (RGB)"**. For a fair comparison, we use the same plane resolutions $T = 64$ and the same number of plane features $F = 32$ in all our experiments. All results are obtained on never-seen test views belonging to scenes coming from both the train and exploit sets. Note that, due to compute constraints, we only train Tri-Planes (RGB) on a reduced version of the datasets.

(a) Tri-Planes (RGB) (b) Ours (c) Ground truth

Figure 8. **Visual comparison**. Visual comparison of novel view synthesis quality for our method and Tri-Planes (RGB).

Results. As seen in Tabs. 1 and 2 and Fig. 6, our method reach the same PSNR as Tri-Planes (RGB) while reducing the the training time by 86% and the memory cost by 44% when training $N_{\text{exploit}} = 1000$ scenes. In addition, rendering novel views using our latent Tri-Planes requires 53% less time. Qualitatively, Fig. 8 compares the novel view synthesis quality of our method with Tri-Planes (RGB) and ground truth test views. Additionally, Fig. 7 shows how our method scales in training time compared to Tri-Planes (RGB).

4.3. Ablations

To justify our choices and explore further, we present an ablation study of our method. The first ablation, "**Ours-Micro**", eliminates the Micro-Macro decomposition, and consequently global information sharing (*i.e.* $F^{\text{mac}} = 0$, $F^{mic} = F$). The second ablation, "**Ours-Macro**", eliminates local features from Tri-Planes and relies only on global features (*i.e.* $F^{mic} = 0$, $F^{\text{mac}} = F$). The third ablation, "**Tri-Planes-Macro**" exclusively trains globally shared Tri-Planes in the image space instead of the latent space. Finally, "**Ours-No-Prior**" refers to an ablation where we reset our globally shared Tri-Planes before the exploitation phase. Note that ablating the latent space as well as information sharing is equivalent to the vanilla "**Tri-Planes (RGB)**" setting. The results of our ablation study can be found in Tab. 2.

5. Conclusion

In this paper, we introduce a novel approach for efficiently learning abundantly many scenes. We propose a 3D-aware autoencoder that enables the training of scene representations in its latent space, drastically speeding-up rendering and training times. Additionally, we present a Micro-Macro Tri-Planes scene decomposition enabling cross-scene information sharing and lighter scene representations. We show that our pipeline reduces resource costs required to learn an individual scene in both time and memory, while showing no quality loss. We envision this work as an essential milestone towards a foundation 3D-aware latent space.

References

[1] Tristan Aumentado-Armstrong, Ashkan Mirzaei, Marcus A Brubaker, Jonathan Kelly, Alex Levinshtein, Konstantinos G Derpanis, and Igor Gilitschenski. Reconstructive latent-space neural radiance fields for efficient 3d scene representations. *arXiv preprint arXiv:2310.17880*, 2023. 2

[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, 2021. 2

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5470–5479, 2022. 2

[4] Ang Cao and Justin Johnson. HexPlane: A Fast Representation for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–141, 2023. 2

[5] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16123–16133, 2022. 2, 3

[6] E. R. Chan, K. Nagano, M. A. Chan, A. W. Bergman, J. Park, A. Levy, M. Aittala, S. De Mello, T. Karras, and G. Wetzstein. Generative novel view synthesis with 3d-aware diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4194–4206, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 2

[7] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 6, 7

[8] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[9] Michael F. Cohen and Richard Szeliski. *Lumigraph*, pages 462–467. Springer US, Boston, MA, 2014. 1

[10] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *Proceedings of the 39th International Conference on Machine Learning*, pages 5694–5725. PMLR, 2022. 2

[11] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields Without Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5501–5510, 2022. 2

[12] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-Planes: Explicit Radiance Fields in Space, Time, and Appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12479–12488, 2023. 2

[13] James T. Kajiya and Brian Von Herzen. Ray tracing volume densities. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984. 3

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4), 2023. 2

[15] J. Kerr, C. Kim, K. Goldberg, A. Kanazawa, and M. Tancik. Lerf: Language embedded radiance fields. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19672–19682, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 2

[16] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *Advances in Neural Information Processing Systems*, pages 23311–23330. Curran Associates, Inc., 2022. 2

[17] Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokra, and Danilo Jimenez Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *Proceedings of the 38th International Conference on Machine Learning*, pages 5742–5752. PMLR, 2021. 2

[18] Verica Lazova, Vladimir Guzov, Kyle Olszewski, Sergey Tulyakov, and Gerard Pons-Moll. Control-nerf: Editable feature volumes for scene rendering and manipulation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 4340–4350, 2023. 2

[19] Nur Muhammad Mahi Shafiullah, Chris Paxton, Lerrel Pinto, Soumith Chintala, and Arthur Szlam. Clip-fields: Weakly supervised semantic fields for robotic memory. *arXiv e-prints*, pages arXiv–2210, 2022. 2

[20] Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12663–12673, 2023. 2

[21] 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, 2

[22] Pol Moreno, Adam R. Kosiorek, Heiko Strathmann, Daniel Zoran, Rosalia Galiazzi Schneider, Björn Winckler, Larisa Markeeva, Theophane Weber, and Danilo Jimenez Rezende. Laser: Latent set representations for 3d generative modeling, 2023. 2

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 2

[24] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1

[25] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2

[26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022. 2, 3, 4

[27] Hoigi Seo, Hayeon Kim, Gwanghyun Kim, and Se Young Chun. Ditto-nerf: Diffusion-based iterative text to omni-directional 3d model. *arXiv preprint arXiv:2304.02827*, 2023. 2

[28] Vincent Sitzmann, Eric Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *Advances in Neural Information Processing Systems*, pages 10136–10147. Curran Associates, Inc., 2020. 2

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

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

[31] Vadim Tschernezki, Iro Laina, Diane Larlus, and Andrea Vedaldi. Neural feature fusion fields: 3d distillation of self-supervised 2d image representations. In *2022 International Conference on 3D Vision (3DV)*, pages 443–453, 2022. 2

[32] J. Ye, N. Wang, and X. Wang. Featurenerf: Learning generalizable nerfs by distilling foundation models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8928–8939, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 2

[33] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. *https://arxiv.org/abs/2012.02190*, 2020. 2

[34] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5752–5761, 2021. 2