CALVET Rémi

ENSAE 3e année
Stage de fin d'études
Année scolaire 2024–2025

Aligning 3D-aware and Stable Diffusion Latent Spaces for Latent NeRFs

Criteo
Paris, FRANCE

Maîtres de stage : Karim KASSAB
Antoine SCHNEPF
Jean-Yves FRANCESCHI
02/06/2025 - 28/11/2025

Contents

Acknowledgements

I would like to thank Karim Kassab, Antoine Schnepf and Jean-Yves Franceschi for their continuous support and engagement during my internship. I am also grateful to the other PhD students, researchers, engineers, and interns at the AI lab, whose scientific and informal exchanges contributed to making this internship a great academic and personal experience.

1 Introduction

Internship context. As part of a double-degree curriculum at ENSAE Paris (National School of Statistics and Economic Administration) and the Master in Data Science at Institut Polytechnique de Paris, I am completing a six-month internship at the Criteo AI Lab in Paris, from June to November 2025. My internship is supervised by Karim Kassab (PhD student at Criteo and National Institute of Geographic and Forest Information - IGN, Gustave Eiffel University - UGE), Antoine Schnepf (PhD student at Criteo and the Laboratory of Computer Science, Signal, and Systems of Sophia Antipolis - I3s, Côte d'Azur University), and Jean-Yves Franceschi (researcher at the Criteo AI Lab).

Company presentation. Criteo is an AdTech company with an in house artificial intelligence laboratory, the Criteo AI lab, that brings together academic researchers and engineers in machine learning. Its activities cover a wide range of fields, including deep learning, computer vision, generative modeling, bandit theory, recommender systems, and natural language processing. The lab has two main missions: contributing to the advancement of machine learning by publishing in top conferences, and transferring academic knowledge into Criteo's products. With the recent breakthroughs in 3D computer vision with the invention of Neural Radiance Fields (NeRFs) [13] in 2020 and 3D Gaussian Splatting [9] in 2023 , the Criteo AI lab has invested in 3D computer vision research with the long-term objective of exploring 3D applications such as 3D advertising, which could offer more engagement than traditional 2D ads.

Internship subject. Neural Radiance Fields enable the synthesis of photorealistic novel views of a 3D scene from a collection of images of the same 3D scene taken from multiple viewpoints. At the same time, 2D image generation models have shown immense progress with the introduction of diffusion models [8] in 2020 and the development of latent diffusion models [16] performing diffusion in lower-dimensional latent spaces. The public release of Stable Diffusion's weights in 2022 made the generation of photorealistic 2D images widely accessible. The latter is particularly interesting for researchers, who can now incorporate massive pretrained models like Stable Diffusion into the pipelines of their new models. My internship project lies at the intersection of 3D computer vision and latent spaces. It continues the work started by Schnepf et al. in *Bringing NeRFs to the Latent Space: Inverse Graphics Autoencoder* [17] and aims to leverage the 2D capabilities of Stable Diffusion to make applications in 3D computer vision tasks like 3D scene editing or generation.

Problem statement. We cannot directly use Stable Diffusion with NeRFs because they do not operate in the same space. While NeRFs are designed for modelling 3D scenes in the RGB space, they can operate in latent spaces. But, they show poor performance in the latent space of pretrained diffusion models. For NeRFs to work correctly, it is important for this latent space

to have a coherent underlying 3D structure, which [17] dub *3D-aware*. The latent space of Stable Diffusion lacks this coherent 3D structure, creating the need for a mapping between it and a 3D-aware latent space. Therefore, the objective of the internship is to develop a method to map the latent space of Stable Diffusion to a 3D-aware latent space and vice versa. This would enable the extension of the 2D capabilities like editing or generation of large pretrained diffusion models to 3D.

2 Prerequisites on 3D Computer Vision

Even though scenes lie in three-dimensional spaces, the most convenient representations are in two dimensions for example on a sheet of paper or a screen. The operation of mapping a 3D scene to a 2D image is called rendering. In this section we present the intuition of the pinhole camera model that we use for the ray casting and ray marching rendering techniques that are essential in NeRFs.

2.1 Pinhole Camera Model

Figure 1: First drawing of a pinhole camera by Gemma Frisius in 1545[1]

The pinhole camera model is a simple model in which light passes through a very small hole to form an image. The phenomenon was first described around 500 BC by the Chinese philosopher Mozi. The drawing by Gemma Frisius [7] shown in figure 1 is the first known printed illustration of this image formation process. It shows the inverted projection of a solar eclipse on a wall through a small hole pierced in a parallel wall facing the Sun. The position of the pinhole is called the camera origin and is denoted $\mathbf{o}$. The distance between the camera origin and the surface where the image is projected, called the image plane, is referred to as the focal distance. Formally, the pinhole camera model can be thought of as a model that projects points with 3D coordinates in the image plane. This model is simple, widely understood, and simplifies numerical computations, which is why it is used in all models presented in this report. A key feature of the pinhole camera model is that each point or pixel is associated with a single ray of light. This means that to determine the colour of a pixel, we only need to consider the light

[1]Image source: Gemma Frisius. De radio astronomico et geometrico liber. See fol. 39r for the first printed illustration of a camera obscura. Louvain: Apud G. Rescium, 1545. url: `https://archive.org/details/bub_gb_Ftk5AAAAcAAJ`

coming from the direction of that single ray. This simplification is justified when the pinhole is infinitesimal, but in practice this approximation simplifies the rendering process and reduces the computational cost.

2.2 Ray casting

We presented the pinhole camera model and highlighted its advantage of facilitating the computation of pixel colours. In this subsection we present *Ray Casting*, a technique to compute pixels from a 3D scene using the pinhole camera model. In ray casting, for each pixel of the image to be rendered, we cast a ray $\mathbf{r}$ from the camera origin $\mathbf{o}$ through the corresponding pixel and into the 3D scene. The line from the camera origin $\mathbf{o}$ to the pixel in the image plane defines the direction vector $\mathbf{d}$. As we are using the pinhole camera model, each pixel is associated with exactly one ray. We can march along this ray $\mathbf{r}(t) = \mathbf{o} + t * \mathbf{d}$ through the scene by increasing the value of the scalar t from 0 onward. The colour of each pixel is determined by the intersection point between the ray and the 3D scene. Ray casting has the advantage of being simple and it is sufficient to render explicit surface geometry but it has no knowledge of what is below the surface. For example Ray Casting is unable to accurately render semi-translucent materials. NeRFs are volumetric representations so *Ray Casting* is not a suitable rendering technique for our project, but we present a solution named *Ray Marching* in the next subsection.

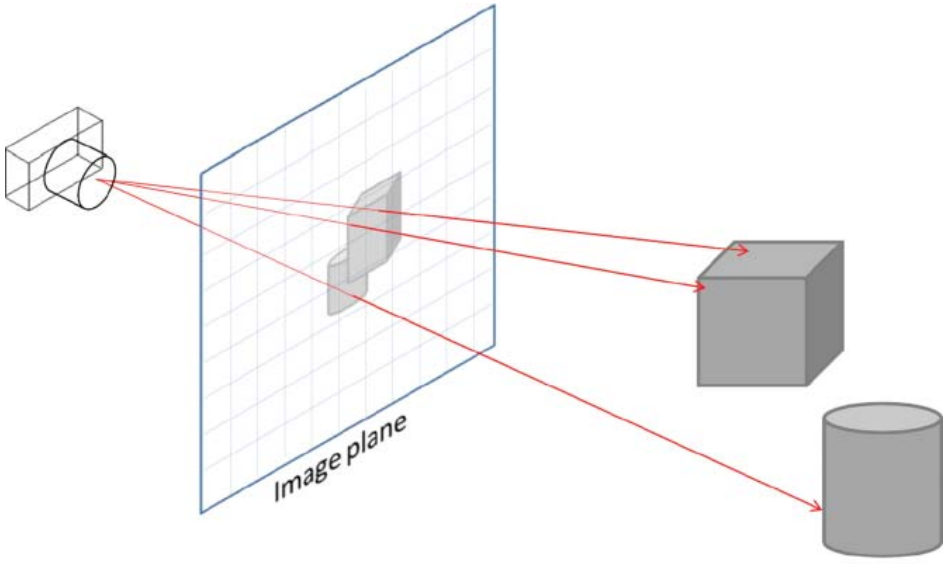

Figure 2: Illustration of ray casting: each pixel corresponds to a ray cast from the camera origin through the image plane into the 3D scene. Rays are stopping at the first contact with the objects. [3]

[3] Image source: Hao Zhang et al. "Full parallax three-dimensional display with occlusion effect using computer generated hologram". In: Optical Engineering - OPT ENG 50 (July 2011). doi: 10.1117/1.3599871.

2.3 Ray marching

NeRFs are volumetric representations, which is why they rely on a technique called *Ray Marching*. This method aggregates the colour and opacity along the entire ray associated with a pixel instead of stopping at the surface of the first object hit like in ray casting. This technique enables to take into account what is below the surface and permits realistic rendering of semi-transparent materials, smoke, glass or other non-opaque materials.

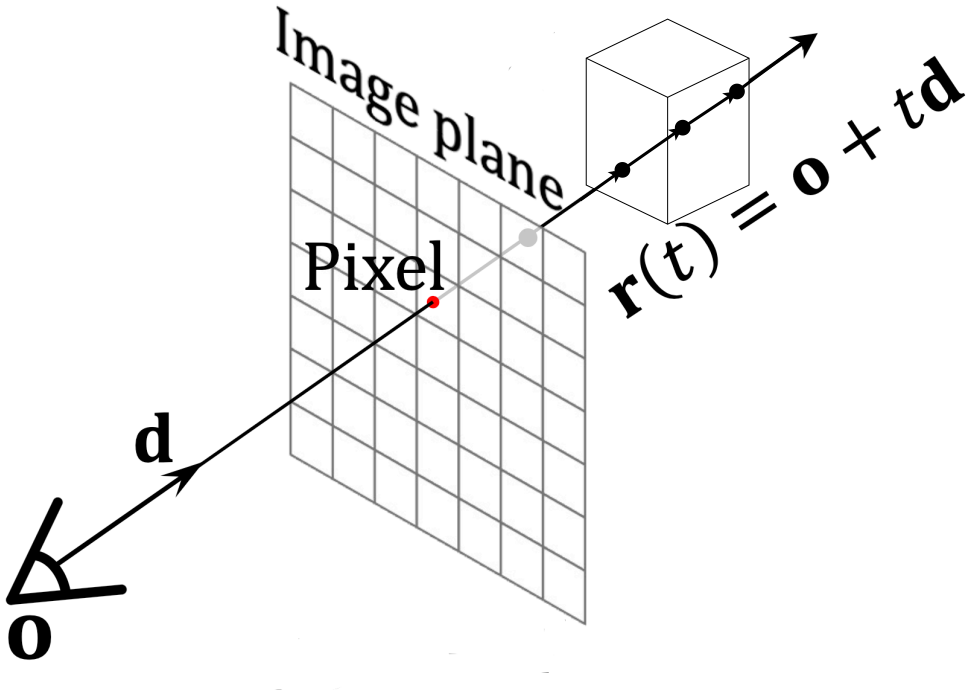

Figure 3: Illustration of ray marching: the colour of a pixel is determined by aggregating the colour and opacity along the ray associated with the pixel. The ray does not stop at the first contact with objects.[5]

[5]Modified image from: Wenbo Hu et al. "Tri-MipRF: Tri-Mip Representation for Efficient Anti-Aliasing Neural Radiance Fields".

3 Inverse Graphics with Neural Radiance Fields

Rendering transforms a 3D scene into a 2D image, inverse graphics is the reverse procedure. It starts from 2D images and tries to reconstruct the 3D scene. In this section we present how to do inverse graphics using deep learning methods through differentiable rendering techniques that trains an implicit neural representation of a scene.

In the subsection 3.1, we describe how a 3D scene is implicitly represented by a neural network. 3.2 details the crucial *volume rendering equation*, while 3.3 explains how it can be numerically approximated. Subsection 3.4 presents the algorithm to train the implicit representation, and 3.5 discusses the inference process. 3.6 provides additional remarks on Neural Radiance Fields.

3.1 Implicit Neural Representation

Formally, a NeRF is a function F that takes as input the coordinates of a point $\mathbf{x} = (x, y, z)$ and a direction (θ, ϕ), and outputs a volume density σ at $\mathbf{x}$ and a colour $\mathbf{c} = (r, g, b)$ emitted by $\mathbf{x}$ in the direction (θ, ϕ). In practice, instead of using angles (θ, ϕ), we represent the direction by a 3D Cartesian unit vector $\mathbf{d}$. This function F implicitly representats the 3D scene, and we approximate it with a neural network parametrised by θ. Thus, a NeRF parametrised by θ is written as

$$F_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$$

where F_θ is an MLP (Multi-Layer Perceptron).

An advantage of implicit representations is their compactness: for the same 3D scene, [13] reports that the memory storage is about 5 megabytes versus 15 gigabytes for an explicit voxel representation.

3.2 Volume Rendering Equation

In this subsection, we assume to have a trained NeRF representation F_θ of a 3D scene and we aim to render 2D views of the 3D scene. NeRF models make the pinhole camera assumption, each pixel in a 2D image corresponds to exactly one ray. Thus, we need to compute the colour value of each ray, which determines the corresponding pixel value. To synthesize a 2D view of the 3D scene we apply the following process to each pixel:

1. We cast a ray $\mathbf{r}(t) = \mathbf{o} + t * \mathbf{d}$ from the camera origin $\mathbf{o}$, going into the 3D scene by varying the value of t, where $\mathbf{d}$ is the direction associated with the pixel as explained in the ray casting subsection 2.2.

2. We do ray marching as in 2.3 and aggregate the opacities and colours along the ray to compute the expected colour of the ray and take it as the pixel value. The aggregation is done via the volume rendering integral defined below.

The expected colour of a ray $\mathbf{r}$ is defined by the volume rendering equation :

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\,\sigma(\mathbf{r}(t))\,\mathbf{c}(\mathbf{r}(t),\mathbf{d})\,dt, \quad \text{where} \quad T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\,ds\right), \qquad (1)$$

and where $\mathbf{c}$ and σ are the outputs of the NeRF F_θ. This integral is fundamental in NeRF and each term can be interpreted as follows:

- t_n and t_f are respectively the near and far bound of the 3D scene. We don't need to integrate over $\mathbb{R}_+$ because we assume that the scene is bounded.
- $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\,ds\right)$ is the accumulated transmittance, i.e the probability that the ray did not hit any other particle between t_n and t. A large value of $T(t)$ means that nothing opaque on the ray $\mathbf{r}$ is occluding the point $\mathbf{r}(t)$. On the contrary, a small value of $T(t)$ means that there is something opaque along the ray $\mathbf{r}$ between the camera origin $\mathbf{o}$ and the point $\mathbf{r}(t)$. $T(t)$ is an non increasing function of t, so if the value of $T(t)$ is low, the contribution of the part of the ray behind t to the expected colour of the ray $\mathbf{C}(\mathbf{r})$ is low.
- $\sigma(\mathbf{r}(t))$ is the volume density output by the NeRF F_θ and represents the opacity of the point $\mathbf{r}(t)$ along the ray. A large value of $\sigma(\mathbf{r}(t))$ means that the point is opaque and the value 0 represents the absence of matter or a fully transparent material. By looking at the integrand of the volume rendering equation, we understand that the higher the value of $sigma(\mathbf{r}(t))$, the higher the contribution of the point $\mathbf{r}(t)$ to the expected colour of the ray will be.

Hence, the integrand in $C(\mathbf{r})$ can be understood as the product of the probability that the ray did not hit any other particle between t_n and t, the opacity of the point $\mathbf{r}(t)$ and the colour emitted by $\mathbf{r}(t)$ in direction $\mathbf{d}$. The volume rendering equation integrates these contributions along $[t_n, t_f]$ to render the expected colour of the ray r in a way that takes into account semi-transparency and occlusions via σ and T.

3.3 Computing the Volume Rendering Equation

The integral of the volume rendering equation is intractable because it would require integrating over neural networks, so [13] approximate it by Monte Carlo. To compute the approximation $\hat{C}(\mathbf{r})$, they divide the ray $\mathbf{r}$ into bins and uniformly sample points $\mathbf{r}(t_i)$ (or their associated scalars t_i) within these bins. The neural network is then evaluated at those points to output colour and density.

This sampling method, called stratified sampling, is highly inefficient since we might sample at places that will not contribute to the rendered colour because they represent empty space or occluded regions. To avoid this, Mildenhall et al. [13] use a second sampling stage called

hierarchical sampling. They use two networks, a 'coarse' network that is evaluated with the t_i uniformly sampled in the bins to find the regions of the ray that contribute most to the rendered pixel colour. Then, a 'fine' network will be evaluated with additional t_i sampled from a distribution biased towards the most important regions found by the 'coarse network'. Here is the procedure in more detail (the training of the two networks will be discussed in the next subsection):

1. Stratified sampling: uniformly sample N_c points t_i along the ray (within bins).

2. Compute the coarse approximation

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \, c_i, \qquad w_i = T_i \, (1 - \exp(-\sigma_i \delta_i)) \, . \tag{2}$$

3. Normalise the weights $\hat{w}_i = \dfrac{w_i}{\sum_{j=1}^{N_c} w_j}$, which defines a discrete probability distribution function over the coarse sample points.

4. Hierarchical sampling: Transform this discrete distribution into a piecewise-constant PDF by assuming the probability density is constant between each pair of coarse samples. Sample N_f additional t_i from the distribution of step 4 using the inverse-CDF sampling technique.

5. Compute the refined approximation using the $N_c + N_f$ t_i:

$$\hat{C}_f(\mathbf{r}) = \sum_{i=1}^{N_c+N_f} w_i \, c_i, \qquad w_i = T_i \, (1 - \exp(-\sigma_i \delta_i)) \, . \tag{3}$$

At inference time, only the fine network is used. The coarse network is included during training to guide the hierarchical sampling process by producing meaningful weights.

3.4 Training algorithm

Up to this point, we assumed access to a trained NeRF F_θ. We now explain the training pipeline (Figure 4) of this implicit representation. The approximations $\hat{C}_c(\mathbf{r})$ and $\hat{C}_f(\mathbf{r})$ are fully differentiable, which allows us to use gradient descent to optimise network parameters. Here is the procedure in more detail, for each optimisation iteration:

1. Randomly sample a batch of pixels, cast their corresponding rays as in section 2.3, and store them in the set $\mathcal{R}$.

2. Sample along the rays with the hierarchical sampling procedure described in the section 3.3.

3. Compute pixels colour with volume rendering integrals as in section 3.3.

4. Compute the mean squared errors of the coarse and fine neural networks:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right].$$

5. Perform a gradient descent step to update network parameters.

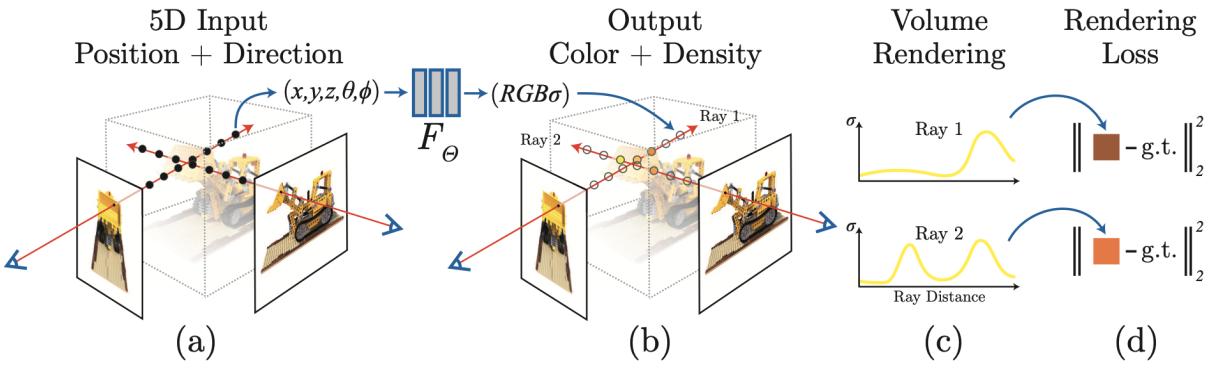

Figure 4: **Vanilla NeRF training pipeline.** (a) Casting rays and sampling along them. (b) Querying the neural network to predict colours and densities at sampled points. (c) Computing pixel colours via volume rendering. (d) Computing the loss between the prediction and the ground truth. [7]

3.5 Inference algorithm

NeRF can render photorealistic views of a 3D scene from angles never seen during training, this is called novel view synthesis. The inference pipeline is similar to the training pipeline, except that we can chose an arbitrary position and viewing direction, even if they are not included in the training set. Here is the inference pipeline:

1. Chose a camera position with its viewing angle.

2. Cast rays.

3. Query the NeRF along the rays, and compute the volume rendering equations.

3.6 Remarks on NeRFs

We explained the full pipelines of vanilla NeRF but the following remarks are also important:

[7] Image source: Ben Mildenhall et al. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. 2020. arXiv: 2003.08934 [cs.CV]. url: `https://arxiv.org/abs/2003.08934`.

1. For a given point $\mathbf{x}$, the volume density $\sigma(\mathbf{x})$ should be independent of the viewing direction. To enforce this, the MLP F_θ only processes $\mathbf{x}$ to predict σ along a 256 dimensional feature vector. The latter is passed to an additional fully connected layer that outputs the colour $\mathbf{c}$. It is important to note that with this setup, the output colour $\mathbf{c}$ does depend on the viewing direction of the point $\mathbf{x}$. This is realistic because for many objects, a point looks different when we look at it from different angles. Objects that verify this property are called non-Lambertian and include polished metal surfaces or water, and are supported by NeRF models.

2. A trained NeRF can only be used for a single 3D scene. If we want to sample new views from another 3D scene we must train another model from scratch, there is no transfer of knowledge between scenes.

3. To achieve photorealistic results, the authors trained their model for 1-2 days on a single NVIDIA V100 GPU.

4 Related work

The prohibitive cost of training NeRFs motivated the development of many variants to accelerate the process. In this section we will review three different paradigms tackling this issue: the explicit grid-based representation, the factorisation, and the latent approaches.

4.1 Grid-based approach

Computing the volume rendering equation requires many neural network evaluations causing a bottleneck during both training and inference. The idea of the grid-based approach is to replace the neural networks implicitely representating the 3D scene which are expensive to evaluate by an explicit representation queried efficiently.

This is what Plenoxel [6] is doing, it replaces the implicit neural network representation by an explicit representation consisting of a sparse voxel grid, conceptually similar to the Plenoctree [18] data structure, where each voxel stores opacity and spherical harmonics containing view dependent colour information. The overall pipeline of Plenoxel is very similar to the vanilla NeRF pipeline. Cast rays, query the representation via a trilinear representation along rays, compute a differentiable volume rendering equation, compute the mean squared error and backpropagate as described in the figure 5. Experiments from the authors of Plenoxel show that their method is two orders of magnitude faster to train than a vanilla NeRF.

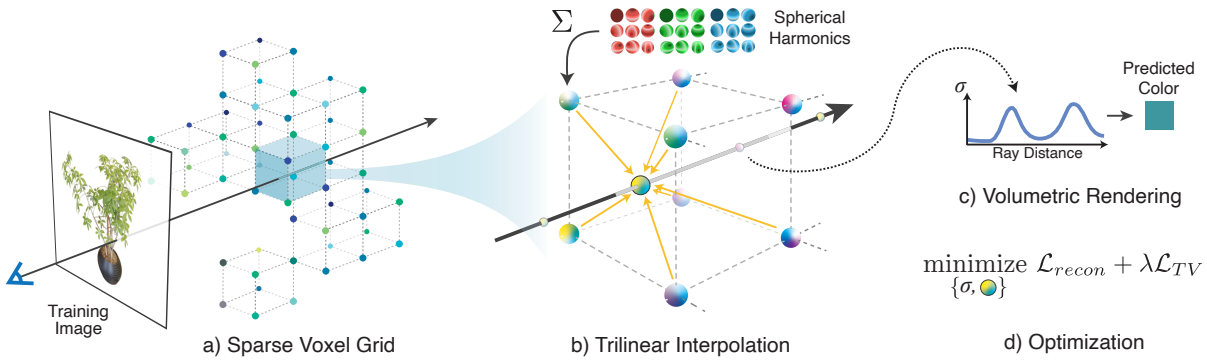

Figure 5: **Plenoxel training pipeline.** (a) Casting rays into the voxel grid. (b) Querying colour and opacity of sampled points via trilinear interpolation. (c) Rendering pixel colours via volume rendering. (d) Computing the loss and backpropagating gradients. [9]

Instant-NGP [14] is not purely grid-based, but adopts an hybrid representation guided by the intuition that, if we can design good encodings from an explicit grid, these can serve as an efficient input to a small neural network. The method replaces the positional encoding used in vanilla NeRF [13] with a multiresolution hash grid of trainable feature vectors. The feature vectors do not directly store density or colour, but abstract features learned by backpropagation, as illustrated in figure 6. The encoded features are then passed to a small neural network that

[9] Figure and caption source: Alex Yu et al. Plenoxels: Radiance Fields without Neural Networks. 2021. arXiv: 2112. 05131 [cs.CV]. url: `https://arxiv.org/abs/2112.05131`.

predicts color and densities. The rest of the pipeline is similar to vanilla NeRF [13] and include ray marching and differentiable volume rendering. The lightweight MLP together with the extensive cuda optimisation by the authors make Instant-NGP highly optimised. It achieves up to one thousand times faster training than vanilla NeRF, enabling training in just a few seconds.

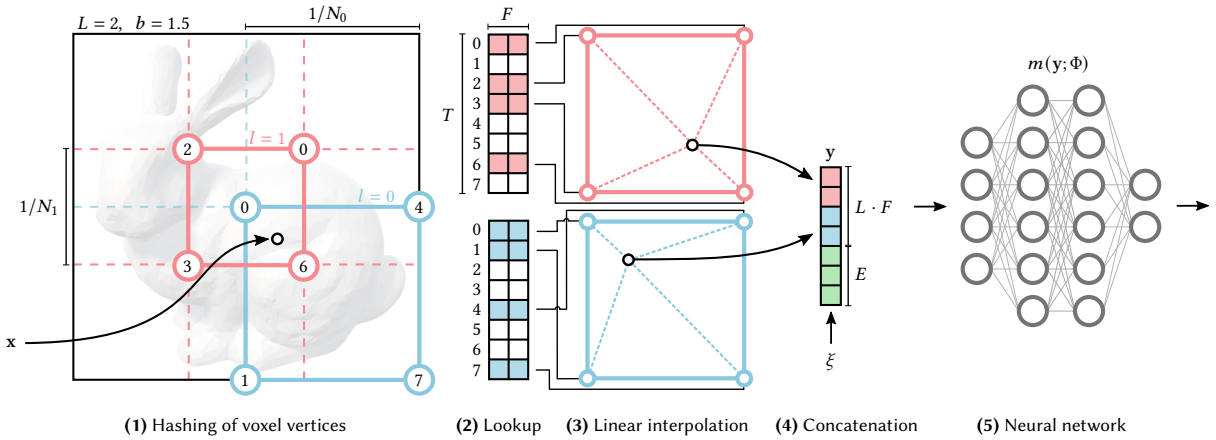

Figure 6: Illustration of the multiresolution hash encoding in 2D. (1) For a given input coordinate $\mathbf{x}$, we find the surrounding voxels at L resolution levels and assign indices to their corners by hashing their integer coordinates. (2) For all resulting corner indices, we look up the corresponding F-dimensional feature vectors from the hash tables θ_l and (3) linearly interpolate them according to the relative position of $\mathbf{x}$ within the respective l-th voxel. (4) We concatenate the result of each level, as well as auxiliary inputs $\boldsymbol{\xi} \in \mathbb{R}^E$, producing the encoded MLP input $\mathbf{y} \in \mathbb{R}^{LF+E}$, which (5) is evaluated last. To train the encoding, loss gradients are backpropagated through the MLP (5), the concatenation (4), the linear interpolation (3), and then accumulated in the looked-up feature vectors.[11]

4.2 Factorisation based approach

Explicit representations have the disadvantage of occupying more memory space than implicit representation. For example, the authors of the vanilla NeRF paper show that for the same 3D scene, a NeRF occupies five meagabytes of storage versus fifteen gigabytes for an explicit voxel representation [13]. There is a trade-off between a fast to render explicit representation and the memory it occupies. In this subsection, we present work models that are faster than vanilla NeRFs but maintain a reasonable memory occupancy by factorising a complex representation of the 3D scene into multiple simpler representations.

TensoRF [4] accelerates the training and reduces the memory usage compared to Plenoxel by factorising a scene in multiple compact low-rank tensors. They split a 3D grid into a geometry grid for the density and, an appearance grid for the colour. The two subgrids are modeled as a sum of vector-matrix outer products factorising tensors [4].

[11] Figure and caption source: Thomas Müller et al. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. 2022. arXiv: 2201.05989 [cs.CV]. url: `https://arxiv.org/abs/2201.05989`.

K-Planes [5] represent d dimensional scenes by making a factorisation with all pairs of dimensions, each pair forming a two dimensional plane. For example, a 3D scene will be factorised in $\binom{3}{2} = 3$ two-dimensional planes. To obtain the value of a point, project the point into each plane, bilinearly interpolate at different scales, concatenate the features over the scales and decode the features with a small MLP and use the volume rendering equation to get a colour [5]. The training pipeline for a four dimensional scene is described in figure 7 but it can be adapted to a scene of any finite dimension. K-Planes offer low memory and fast training, the authors report a 35 min training time on a single GPU using their model versus eight GPUs for two days with NeRF-W [11], an adaptation of vanilla NeRF, for a similar quality [5].

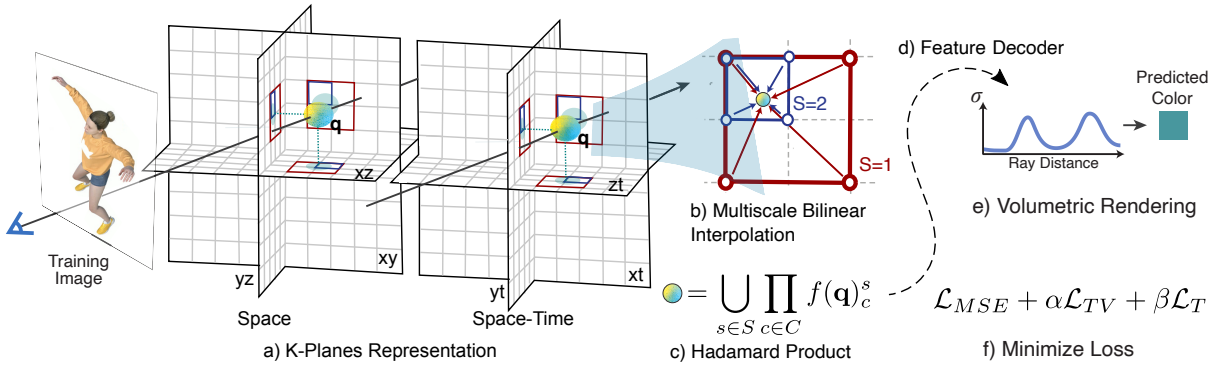

Figure 7: **K-Planes training pipeline for a four-dimensional scene.** (a) Casting rays and projecting sampled points into planes. (b) Interpolating features at different scales. (c) Combining features across scales. (d) Decoding the obtained features. (e) Rendering pixel colours via volume rendering. (f) Computing the loss and backpropagating gradients. [13]

Tri-Planes [2] is closely related to K-planes which were inspired by it. We present it here because we will use them in our models, presented later in this report. This representation is both efficient and expressive as it has been developed as part of a Generative Adversarial Network pipeline for real-time 3D scene generation. In this approach, the 3D scene is factorised into three two-dimensional planes (X, Y), (X, Z), (Z, Y) combined with a small fully connected neural network. It also enables feature planes to be generated with an off-the-shelf 2D generator [2]. To obtain the color and density of a point $p \in \mathbb{R}^3$, we project p onto each of the three feature planes, retrieving the corresponding feature vector (F_{xy}, F_{xz}, F_{yz}) via bilinear interpolation, and aggregating the three feature vectors via summation [2]. The aggregated features are then passed to a small MLP that outputs colour and density. To synthesize an image we use the volume rendering equation as in K-Planes and vanilla NeRF.

4.3 Latent approach

Inspired by the generative modeling literature, the latent approach aims to accelerate NeRFs training by performing the heavy volume rendering computations in a lower dimensional latent

[13] Image source: Sara Fridovich-Keil et al. "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance". In: CVPR. 2023.

space. Training a NeRF requires rendering many images, and we must compute the volume rendering integral for each pixel of each image. Since each computation of the volume rendering equation requires evaluating a neural network several times, training a high-resolution NeRF becomes computationally prohibitive. The number of neural network evaluations per epoch is $Batch\ size * Resolution * (N_c + N_f)$, where the batch is a batch of images, not rays. For a batch of eight images of resolution 480×480 and $N_c + N_f = 100$ this results in $184,320,000$ neural network evaluations. Computing the volume rendering integral in a lower-dimensional latent space instead of a high-dimensional pixel space requires much less neural network evaluations, leading to faster training and inference. Another important advantage of the latent approach is the possibility to use pretrained latent models to perform 3D scene generation [12] or editing [15]. The last article [17] presented in this subsection adresses the lack of geometric coherence in latent spaces.

4.3.1 Latent-NeRF

Latent-NeRF [12] is the first paper training NeRFs in latent spaces for 3D scene generation. The training pipeline has two main steps, first the NeRF is trained in a latent space and then it can be refined in the RGB space by adding a linear layer converting latents to RGB space.

Here are the main steps of the latent training loop visible in figure 8:

1. Randomly choose a camera view and render the NeRF to get a latent z.
2. Noise the latent and predict the noise with the pretrained latent diffusion model Stable Diffusion [16].
3. Compute the difference between the predicted noise and the noise and back-propagate to update the NeRF.

The key point to remember from this paper is that the volume rendering equation is computed in the compact latent space of a pretrained autoencoder and that the task of this model is 3D scene generation [12].

4.3.2 ED-NeRF

ED-NeRF [15] introduces a pipeline to train a NeRF in a latent space and another to do 3D scene editing, we focus on the analysis of latent training. The authors of ED-NeRF [15] observed that naïvely training NeRFs in a latent space leads to a significant drop in performance for novel view synthesis. They explain that the latent space of Stable Diffusion lacks 3D geometric consistency and that the poor reconstruction performance can be explained by analysing the architecture of Stable Diffusion's encoder and decoder. They note that pixel values have an influence on

[15]Image source: Gal Metzer et al. Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures. 2022. arXiv: 2211.07600 [cs.CV]. url: https://arxiv.org/abs/2211.07600.

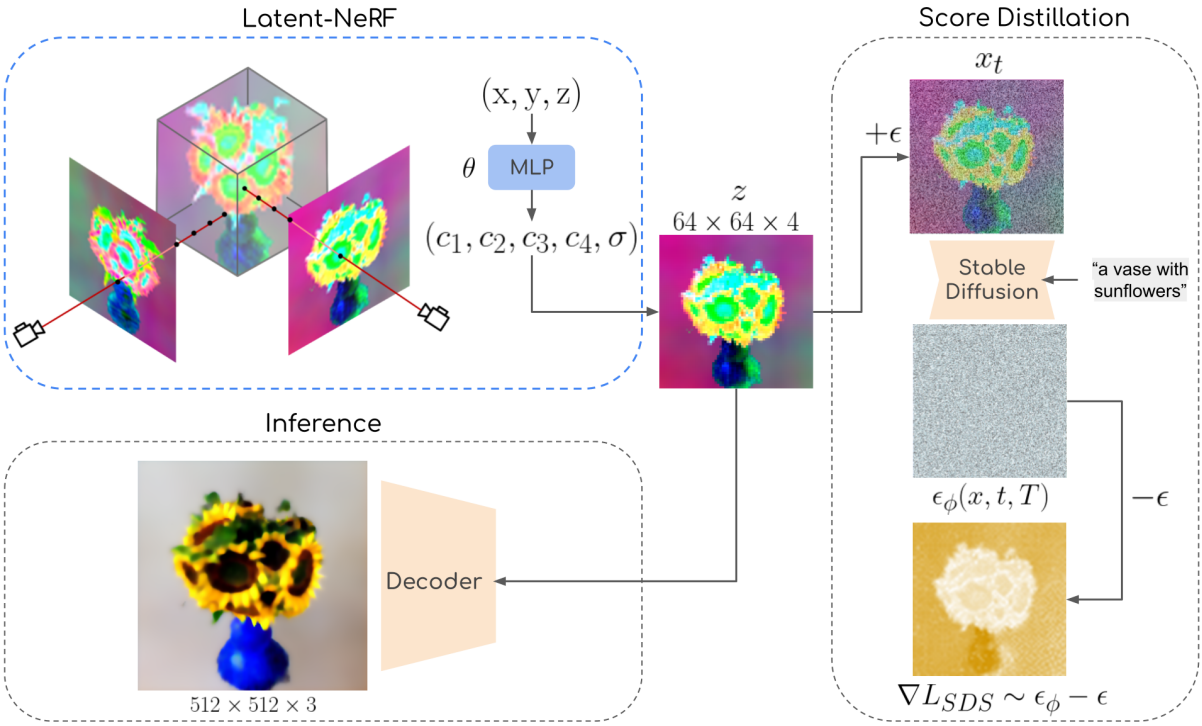

Figure 8: **Latent-NeRF training pipeline.** (a) The Latent-NeRF block performs volume rendering in a latent space. (b) The Score-Distillation block noise and denoise rendered latent images and backpropagate in the NeRF. (c) The Inference block is only used after training to obtain RGB images. [15]

one another, locally via convolutions in ResNet blocks and globally via self-attention layers in the encoder and decoder. This creates an inconsistency with NeRFs that render each pixel independently without taking into account the interdependent relations existing between latent pixels. This problem is addressed by the authors with a refinement layer placed after the NeRF (figure 9). This solution achieves good reconstruction quality but it requires retraining the refinement layer for each scene. The next subsection introduces a different approach that aims to overcome this limitation.

4.3.3 IG-AE

Our model, introduced in section 5, is based on the IG-AE framework presented in this subsection. The authors of IG-AE [17] formalise the idea of geometric coherence. The underlying intuition is that when rotating an object in latent space, the views should change accordingly. This property is referred to as the 3D consistency property and will be used throughout this report.

Schnepf et al. [17] observe significant flickering effects when naïvely training NeRFs in the latent space of Stable Diffusion. They explain it by the fact that latent spaces are not naturally

[17] Image source: Jangho Park, Gihyun Kwon, and Jong Chul Ye. ED-NeRF: Efficient Text-Guided Editing of 3D Scene with Latent Space NeRF. 2024. arXiv: 2310.02712 [cs.CV]. url: https://arxiv.org/abs/2310.02712.

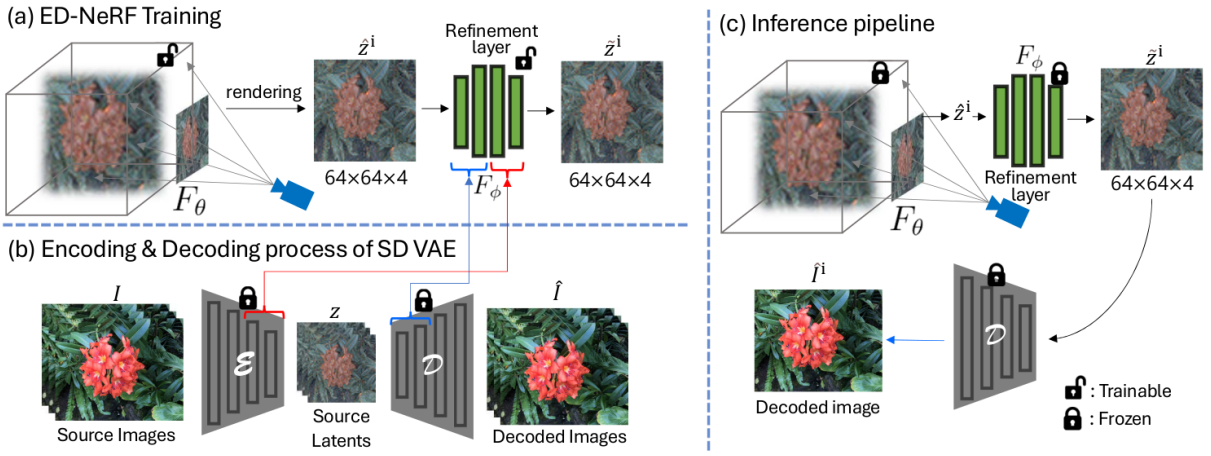

Figure 9: **ED-NeRF pipeline** (a) Training a NeRF in latent space together with a refinement layer. (b) Initialising the refinement layers from Stable Diffusion's encoder-decoder. (c) Rendering the NeRF, refining the output, and decoding to RGB. [17]

3D-awarere, and that the volume rendering equation requires a 3D-aware space to work correctly. The latter is due to the fact that NeRFs are 3D representation that are supposed to learn the underlying 3D geometry of the training images. The poor results come from the vain efforts of the NeRF to learn a geometry that does not exist.

While ED-NeRF [15] relies on scene-specific refinement layers, IG-AE [17] instead proposes to train a universal autoencoder that creates a 3D-aware latent space usable across different scenes, without the need for refinement layers. The main idea to train this 3D-consistency preserving autoencoder is to add a regularisation term that distils 3D consistent knowledge from Tri-Planes (4.2) to the autoencoder as shown in the figure 10.

The authors chose Tri-Planes for their efficiency in memory footprint and speed that enables training on many 3D scenes, which is a requirement for the autoencoder to generalise. They train their autoencoder on five hundred 3D scenes from the Objaverse dataset and then, use the pipeline in figure 11 to train NeRFs in the 3D-aware latent space. They achieve faster training with the lower computational cost of latent space training with significantly better quality compared to naïve latent NeRF training. Their results are best illustrated in their videos on their project web page[20].

[19] Antoine Schnepf *et al.* *Bringing NeRFs to the Latent Space: Inverse Graphics Autoencoder*. In: *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. Available at: `https://openreview.net/forum?id=LTDtjrv02Y`.

[20] IG-AE video results: `https://ig-ae.github.io/`

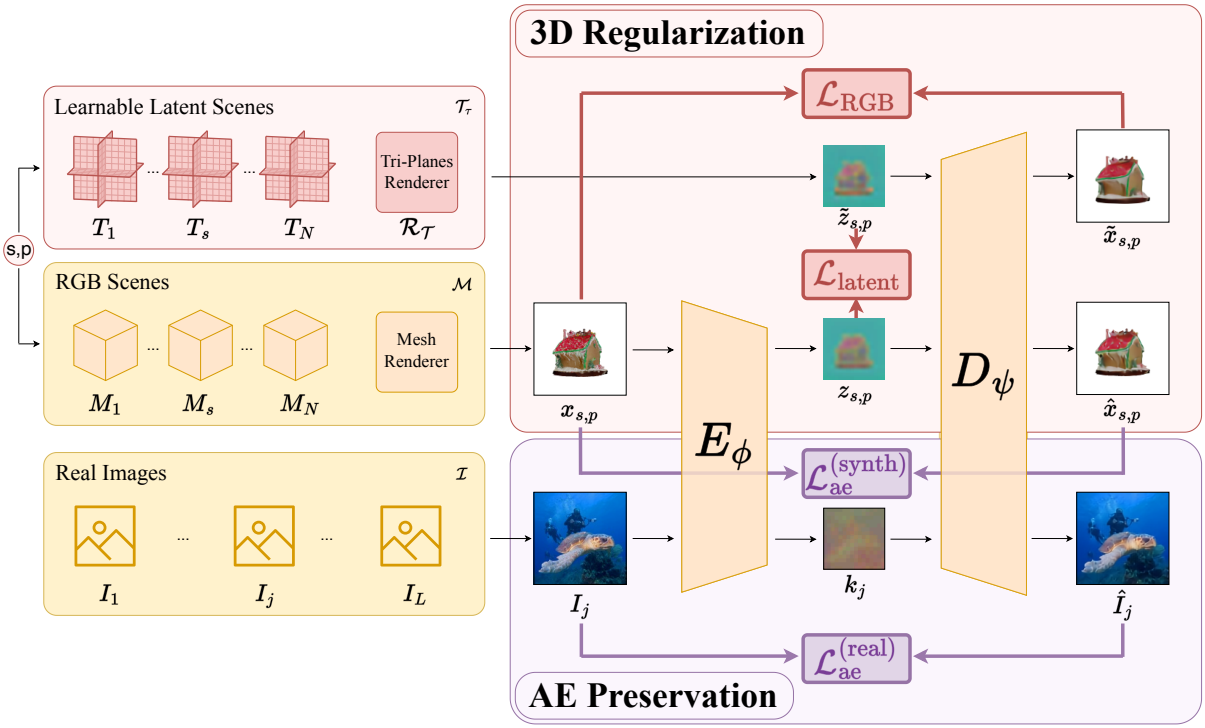

Figure 10: **Pipeline of the IG-AE 3D-awarere autoencoder training.** (a) The 3D Regularisation block makes the latent space 3D-awarere. (b) The AE Preservation block enhances real images performance. [19]

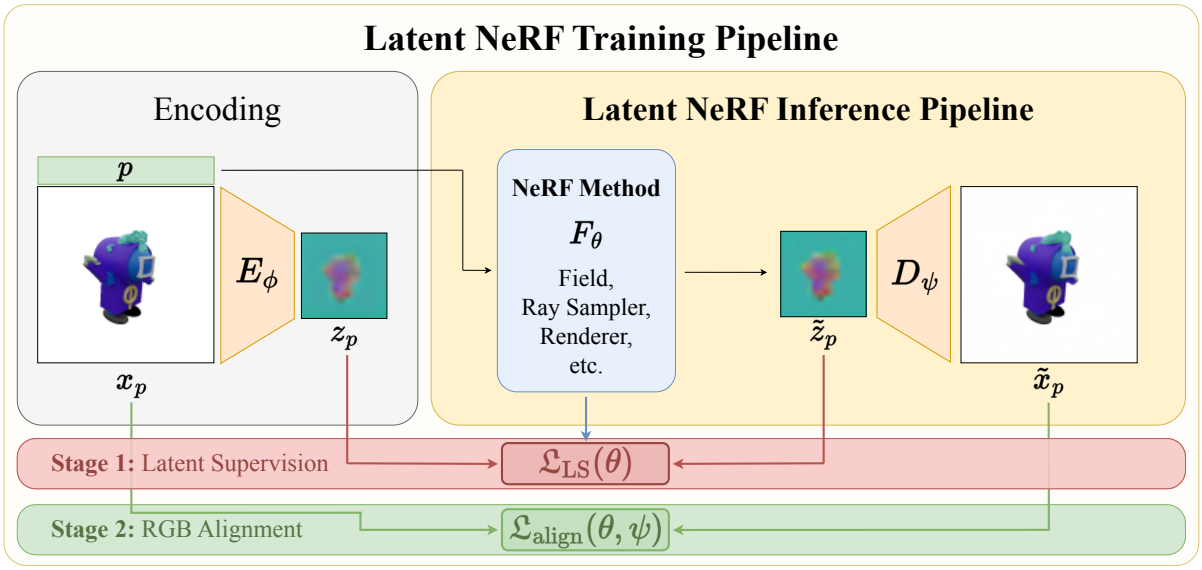

Figure 11: **Pipeline of IG-AE to train NeRFs in latent spaces.** (a) Encoding images with a 3D preserving encoder E_ϕ. (b) Rendering the NeRF in the latent space. (c) Decoding the latents to RGB. [19]

5 Method: Aligning 3D-aware and Stable Diffusion latent spaces with E2D2

5.1 Motivation

IG-AE gives us a 3D-aware latent space allowing better latent NeRFs training than with a naïve non 3D-aware latent space. The encoder and decoder of IG-AE are initialized with a pretrained variant of Stable Diffusion named Ostris [1] and during the training the encoder and decoder weights are updated to make a 3D-aware latent space. Unfortunately, this procedure breaks the Stable Diffusion latent space because the latter is incompatible with the 3D consistency property. Ideally, we would like to have a latent space that is both 3D-aware to train NeRFs efficiently and, that is compatible with Stable Diffusion to leverage its powerful capacities in various image tasks such as editing, inpainting and generation.

5.2 Intuition of the new E2D2 method

We extend the pipeline of IG-AE and take inspiration from the refinement layers of ED-NeRF to propose a new pipeline named E2D2. Our model circumvents the impossibility to have both the desired properties in the same latent space by constructing one latent space for each property and, a way to go from one latent space to the other as illustrated in figure 12. Once E2D2 is trained, it does not theoriticaly require to retrain for new scenes unlike ED-NeRF [15] where refinement layers need to be retrained for each scene.

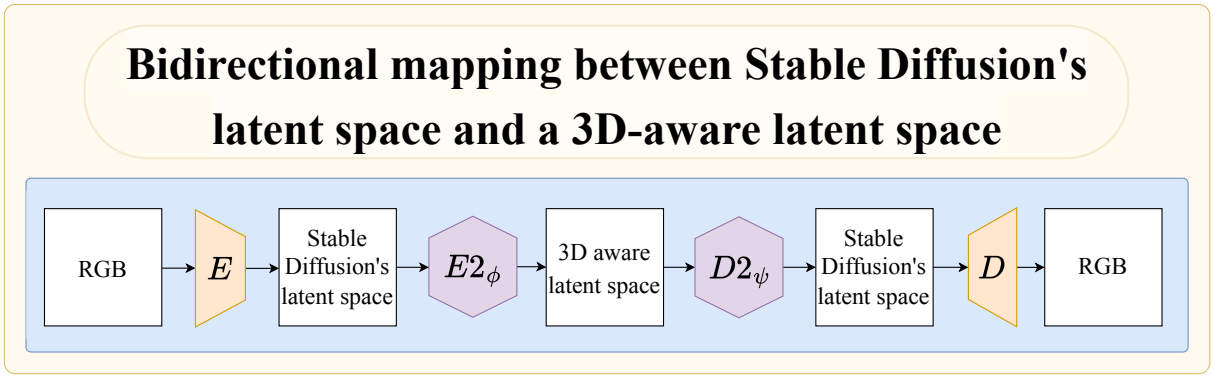

Figure 12: **Bidirectional Mapping Between Latent Spaces.** The figure illustrates the mapping from Stable Diffusion's latent space to a 3D-aware latent space via $E2_\phi$, and the reverse mapping via $D2_\psi$. E and D are respectively Stable Diffusion's encoder and decoder.

5.3 Architecture and pipeline of E2D2

In this subsection we will present the main components of the E2D2 pipeline (Figure 13) and, underline the key components to ensure 3D awareness and alignment with Stable Diffusion's latent space.

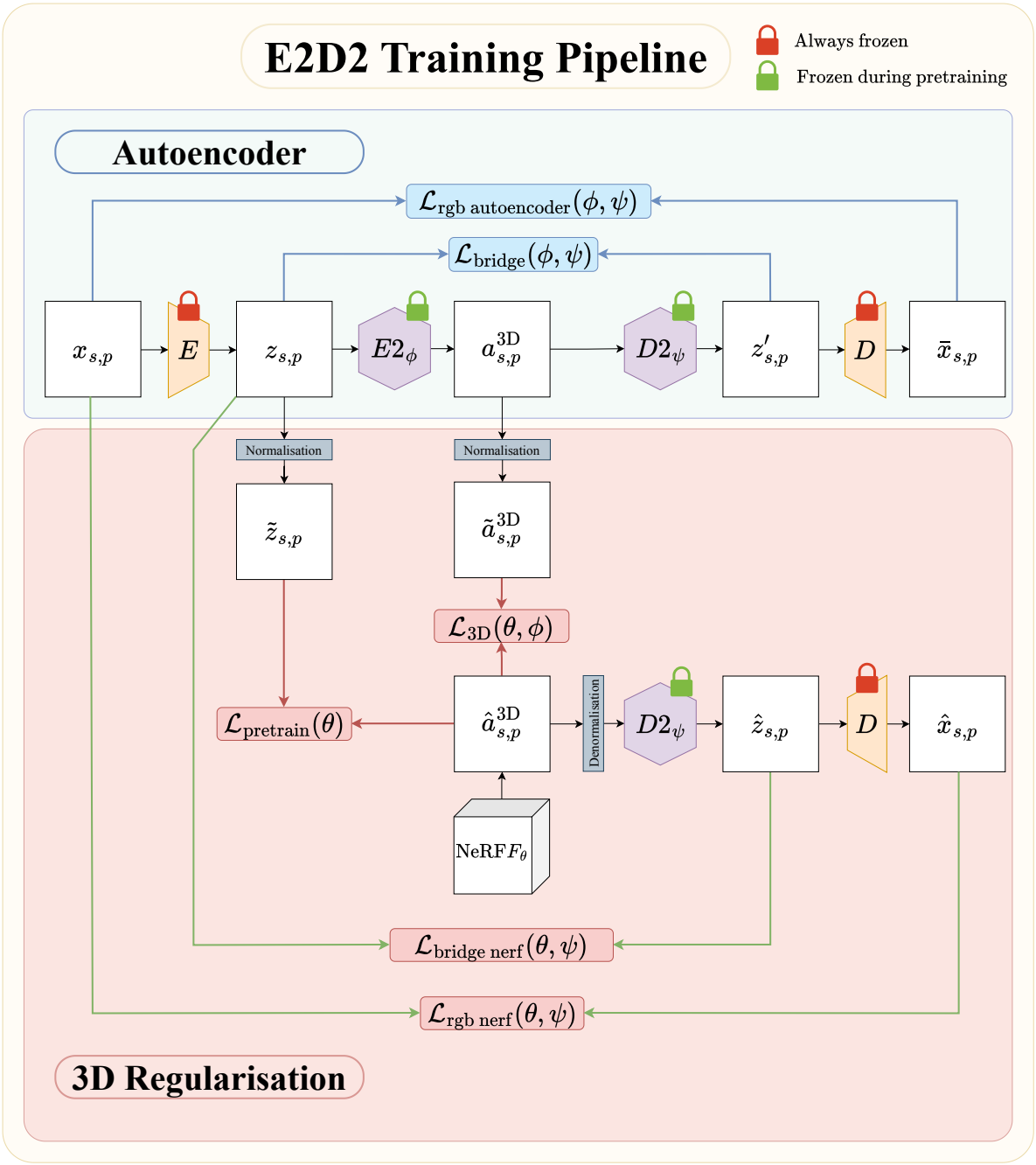

Figure 13: **Pipeline of E2D2** (a) The Autoencoder block in blue enforces the reconstruction of the RGB and latent images. (b) The 3D Regularisation block in red enforces the 3D awareness of E2's latent space. The NeRF module contains one Tri-Plane per scene; in our experiments we train on 50 scenes, so this block contains 50 Tri-Planes.

5.3.1 Global view of the E2D2 pipeline

The pipeline of E2D2 is divided in two parts, an autoencoder pipeline (in blue) at the top and, a 3D regularisation pipeline (in red) at the bottom. The most important blocks are $E2_\phi$ and $D2_\psi$ which are decoder-encoder blocks. The idea is to feed $E2_\phi$ a latent image from Stable Diffusion's encoder E to produce a latent image coherent with the underlying geometry of the original 3D scene. Then, this latent is passed through $D2_\psi$ that will project the latent back

into Stable Diffusion's latent space to allow its decoding by the decoder of Stable Diffusion noted D. Once $E2_\phi$ and $D2_\psi$ are trained, we have a bidirectional mapping between a 3D-aware latent space and Stable Diffusion's latent space. This circumvents the incompatibility of the 3D awareness and Stable Diffusion properties by using two seperate latent spaces, one for each property.

5.3.2 From Stable Diffusion's Latent Space to a 3D-aware Latent Space via $E2_\phi$

The training of the block $E2_\phi$ to create a 3D-aware latent space is not trivial and requires a 3D-aware signal to supervise the training. We take inspiration from the IG-AE [17] pipeline (Figure 10) where the 3D-aware signal is given by the Tri-Planes, that are 3D-aware by construction. This supervision corresponds to the loss $\mathcal{L}_{3D}$ between the output of the NeRF $\hat{a}^{3D}_{s,p}$ and the normalized output $\tilde{a}^{3D}_{s,p}$ of $E2_\phi$.

5.3.3 From a 3D-aware latent space to Stable Diffusion's latent space via $D2_\psi$

Ideally, $D2_\psi$ should be able to retrieve the Stable Diffusion latent fed to $E2_\phi$. In other words, $D2_\psi$ should act like an inverse of $E2_\phi$ and map a latent from the 3D-aware space to the non-3D-aware latent space of Stable Diffusion. $D2_\psi$ is directly trained to inverse $E2_\phi$ via the loss $\mathcal{L}_{bridge}$. $D2_\psi$ is also trained via the losses $\mathcal{L}_{rgb\ NeRF}$ and $\mathcal{L}_{rgb\ autoencoder}$, to make sure that decoded latents look close to the RGB images fed to Stable Diffusion's encoder E.

5.3.4 NeRF pretraining

Note that the NeRF is initialised randomly and needs to be pretrained to give a relevant 3D-aware signal and not pure noise, this is the role of the loss $\mathcal{L}_{pretrain}$. The consequences of not pretraining the NeRF before training the other blocks of the pipeline are presented in the ablations of the results section. It is also important to note that $\mathcal{L}_{pretrain}$ is disabled during the training because it would lead to contradictory signals, $\mathcal{L}_{pretrain}$ is encouraging the optimiser to make the NeRF closer to the latent space of Stable Diffusion hence, corrupting the signal propagated through $\mathcal{L}_{E2}$ and making the latent space of $E2_\phi$ non-3D-aware but, close to Stable Diffusion which is not the role of $E2_\phi$.

5.3.5 Normalisation layers

The normalisation layers in $E2D2$ are necessary to align the output domains of the different blocks. NeRF ouputs are normalised and lie in $[-1, 1]$, the output latents of Stable Diffusion's encoder E and $E2_\phi$ must be normalised to lie in the same domain of the NeRF. Normalisation is done via rescaled hyperbolic tangents

$$\text{Normalisation}_1(z_{s,p}) = \tanh(\alpha_1 * z_{s,p}) = \tilde{z}_{s,p} \tag{1}$$

and

$$\text{Normalisation}_2(a_{s,p}^{3D}) = \tanh(\alpha_2 * a_{s,p}^{3D}) = \tilde{a}_{s,p}^{3D} \tag{2}$$

independently for the latent output of E and $E2_\phi$ as they may not be distributed the same. Rescaling factors α_1 and α_2 are adjusted such that most of the points are mapped in the linear region of the hyperbolic tangents (Figure 14), otherwise the points will all be mapped very close to -1 and 1 becoming indistinguishable and leading to numerical instabilities.

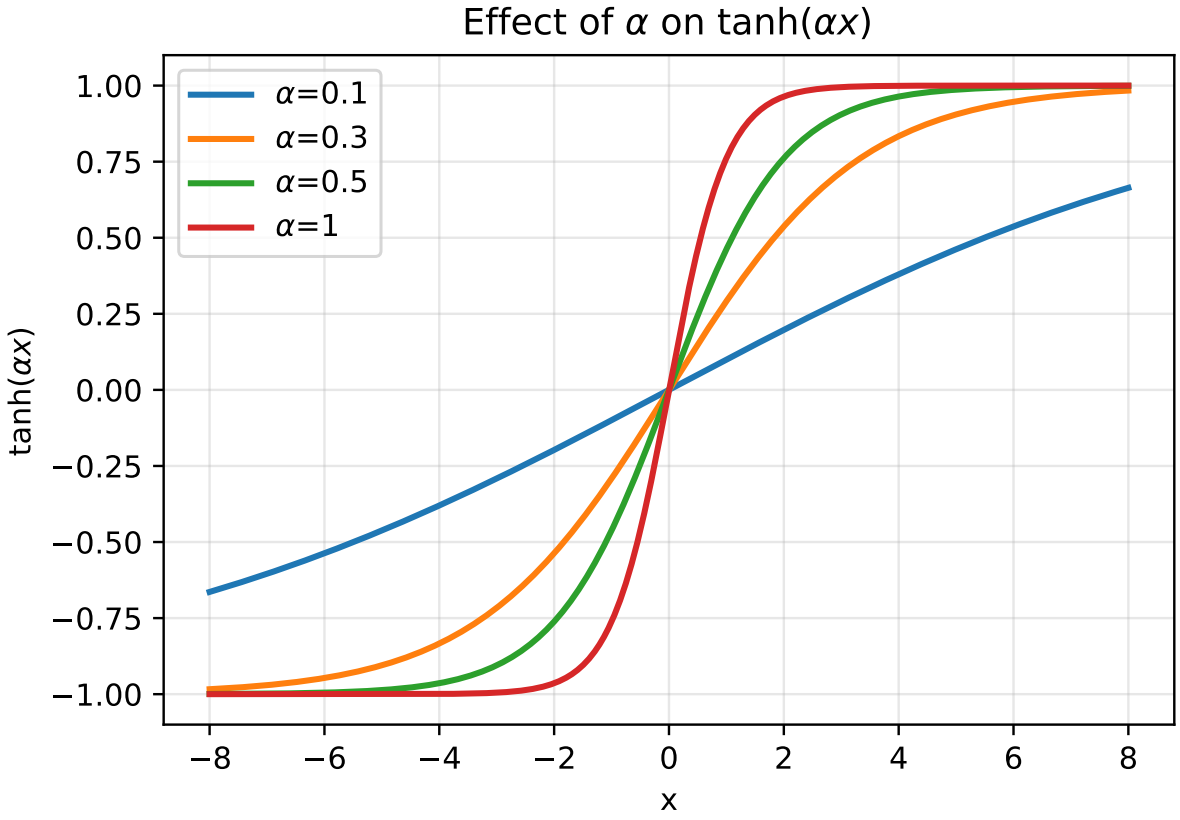

Figure 14: **Hyperbolic Tangent Normalisation.** The more the data is spread, the lower the rescaling factor α should be.

5.3.6 Remarks

The encoder E and decoder D in the E2D2 pipeline (Figure 13) are frozen to preserve Stable Diffusion's latent space. The only trainable blocks during trainining are $E2_\phi$, $D2_\psi$ and the NeRF.

The $E2_\phi$ and $D2_\psi$ blocks have the same architecture, they follow the same principle as the refinement layer in ED-NeRF [15]. Its first layers consists of the first layers of Stable Diffusion's decoder D and the next layers are the last layers of Stable Diffusion's encoder. The intuition of the architecture of this block is to slightly decode the image and to slightly reencode it to another type of space (3D-aware or Stable Diffusion's). This type of initialization allows the modules

$E2_\phi$ and $D2_\psi$ to begin the training with plausible weights, from the pretrained Stable Diffusion encoder and decoder, making them cheaper to train.

5.4 Adaptation of the latent NeRF pipeline for E2D2

We adapt the latent NeRF training pipeline of IG-AE [17] to integrate the trained E2D2 blocks enabling the training of NeRFs in a 3D-aware latent space while maintaining compatibility with Stable Diffusion's latent space.

The blue part of the pipeline is responsible for encoding the RGB views of a 3D scene into latent images coherent with the underlying 3D geometry of the scene. This is done by passing the RGB images through Stable Diffusion's encoder E and the trained decoder-encoder block $E2_\phi$. Those latent images serve as supervision to train a NeRF F_θ via the loss $\mathcal{L}_{\text{NeRF}}$ in figure 15. It is important to note that only the NeRF parameters θ are trainable and that all the other blocks are frozen.

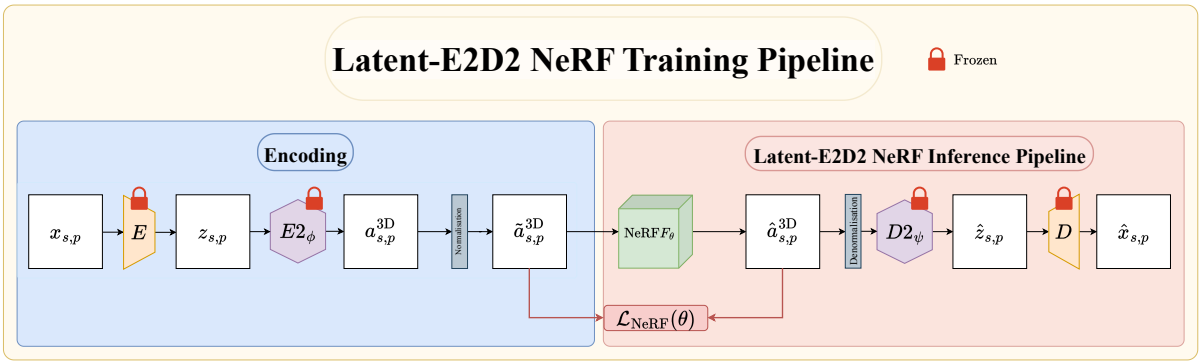

Figure 15: **Latent-E2D2 NeRF Training Pipeline** (a) The Autoencoder part in blue provides supervision for the NeRF, it does not contain trainable components. (b) The red part trains the NeRF, everything else is frozen.

Once E2D2 is trained, the latent NeRF training pipeline (Figure 15) enables efficient NeRF training in a 3D-aware latent space while remaining compatible with Stable Diffusion's latent 2D generative model. This compatibility could enable applications such as editing or generation of latent NeRFs from textual prompts.

6 Experiments

6.1 Experimental setup

6.1.1 Training dataset

All our experiments are conducted using a preprocessed version of the ShapeNet dataset [3], prepared by Antoine Schnepf and Karim Kassab. The dataset consists of sets of 2D images captured from virtual cameras placed at various viewpoints around synthetic 3D car models. Each model is rendered from 160 viewpoints uniformly distributed over the upper hemisphere of the scene, with an image resolution of 128×128 pixels. Views from the lower hemisphere are omitted to focus on the most visually relevant parts of the cars and to reduce computational cost, though they could easily be included if needed.

6.1.2 Pretraining

We pretrain the Tri-Planes only once and use the same pretraining checkpoint for all the ablations to ensure fair comparison between the baseline and the ablated models.

6.1.3 Training

For both the training and pretraining, we use the ADAM [10] optimiser with the default Pytorch implementation hyperparameters. We train our baseline with 50 scenes for 50 epochs, and all the losses enabled except the loss $\mathcal{L}_{\text{pretrain}}$ that is only used during pretraining. All the losses are optimised jointly with the same weight. The ablations are also conducted with 50 scenes.

6.1.4 Hardware

Except for an experiment with 500 scenes, we trained all our models on a single Nvidia $L4$ GPU. The pretraining lasted 2 hours and the training of the baseline 11 hours. The experiment with 500 scenes instead of 50 was conducted on 4 Nvidia $L4$ GPUs.

6.2 Results

In this subsection we present the results of the baseline, where each loss is enabled.

6.2.1 3D-aware signal

We observed that the latents $\hat{a}^{3D}_{s,p}$ outputed by the Tri-Planes F_θ are smoother before they are passed through $D2_\psi$ as we can see in the figure 16. This result is encouraging and expected since the smoothness makes the latents less sensible to flickering effects and thus more 3D consistent across the different views. The latents $\hat{z}_{s,p}$, outputed by the Tri-Planes and passed through $D2_\psi$

are less smooth, which is expected since the role of $D2_\psi$ is to project its input on the latent space of Stable Diffusion that is not 3D-aware as noted by Schnepf et al. in [17].

Those smooth latents, $\hat{a}^{3D}_{s,p}$, are the 3D-aware signal passed to the block $E2_\phi$ through the loss $\mathcal{L}_{3D}(\theta, \phi)$ and we can control the strength of this signal by modifying the value of the weight of this loss. It is important that the latents outputed by the Tri-Planes are smooth, because otherwise we would not have a 3D-aware signal to supervise $E2_\phi$ that we train to create a 3D-aware latent space.

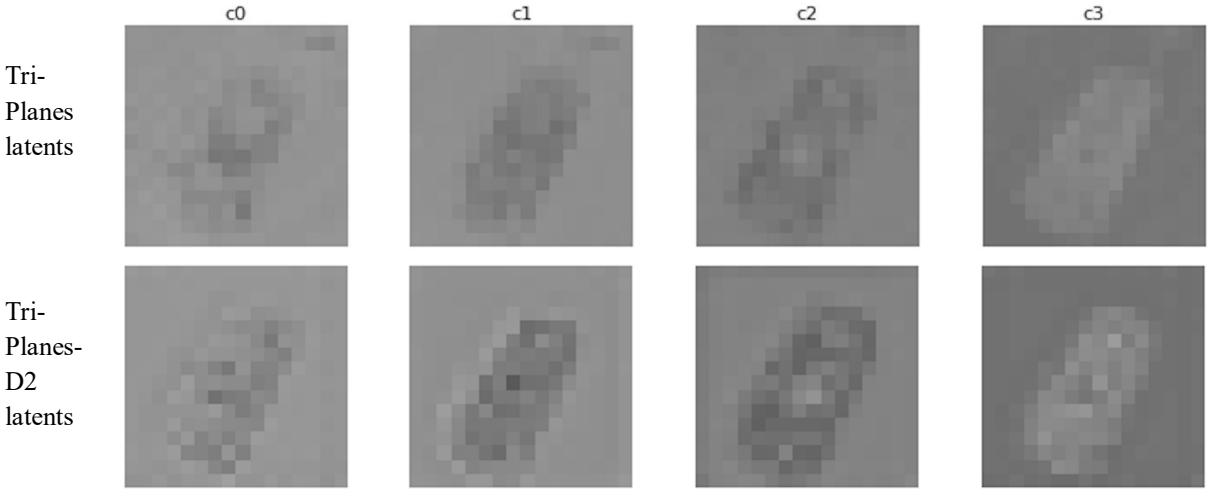

Figure 16: **Comparison of Tri-Planes Latents before and after being passed through $D2_\psi$.** The first row shows the latents directly outputed by the Tri-Planes, while the second row displays the same latents after passing through $D2_\psi$. The columns are representing the four channels of the latent images. The latents are smoother before passing through $D2_\psi$, indicating a higher degree of 3D consistency which is expected as $D2_\psi$ maps the 3D-aware latent space to Stable Diffusion's latent space.

6.2.2 $E2_\phi$ and the 3D-aware latent space

Our results show that $E2_\phi$ is able to learn to produce a latent space that is more 3D-aware than the one of Stable Diffusion. In fact, we can see in the figure 17 that the normalised output of $E2_\phi$, $\tilde{a}^{3D}_{s,p}$, is much more smooth than the latent $z_{s,p}$ outputed by Stable Diffusion's encoder E. This result is encouraging since the role of $E2_\phi$ is to project the latents from Stable Diffusion's latent space to a 3D-aware latent space. The outputs of $E2_\phi$ being smoother than the ones of Stable Diffusion's encoder, they are less prone to flickering effects when rotating the camera around the object. That makes the latent space produced by $E2_\phi$ more 3D-aware than Stable Diffusion's latent space.

By multiplying the weight of the loss $\mathcal{L}_{3D}$ by ten, we observe that the latents produced by $E2_\phi$ become oversmooth as visible in the third row in figure 17. There is a tradeoff between the smoothness that makes the latent space more 3D-aware and losing details due to oversmoothing.

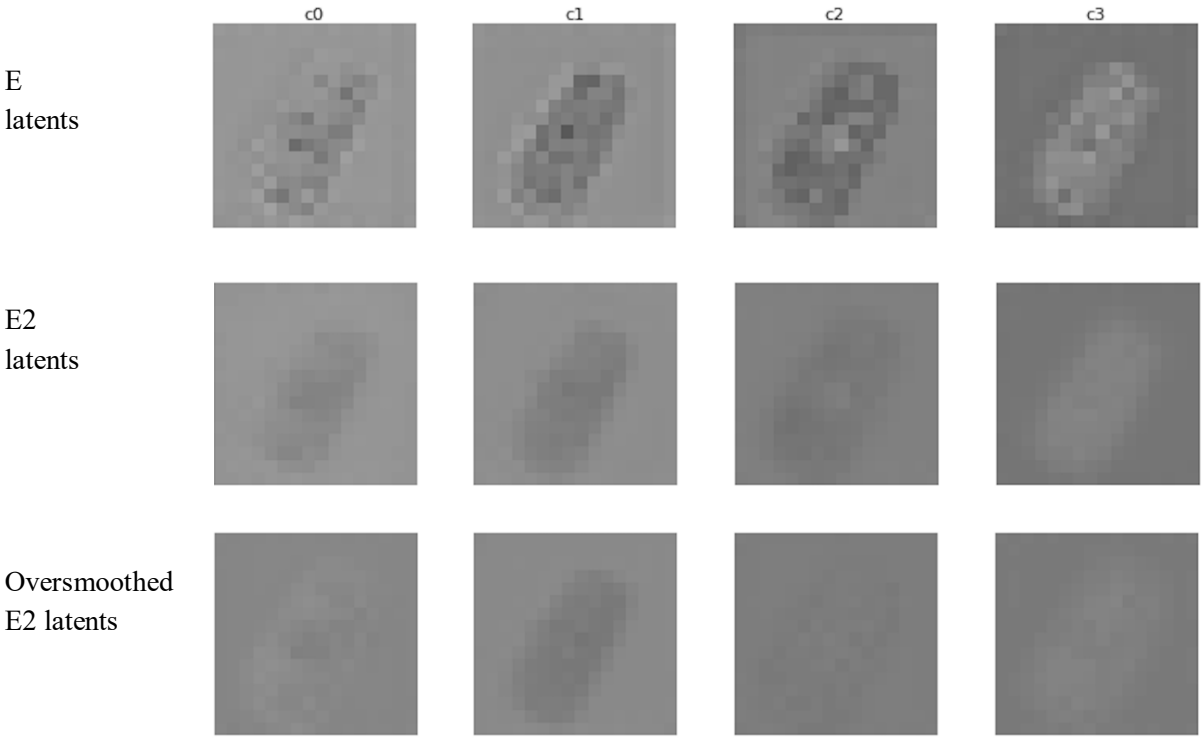

Figure 17: **Comparison of Stable Diffusion and E2 Latents.** The first row shows the latents produced by Stable Diffusion's encoder E, while the second row displays the latents outputed by the $E2_\phi$ block. The third row shows the latents outputed by $E2_\phi$, when we increase the weight of the loss $\mathcal{L}_{3D}$ by ten. The columns represent the four channels of the latent images. The smoother latents produced by $E2_\phi$ indicate a higher degree of 3D consistency, learned from the Tri-Planes, than Stable Diffusion.

6.2.3 $D2_\psi$ reverses $E2_\phi$

We observe that the block $D2_\psi$ correctly maps the latents from the 3D-aware latent space produced by $E2_\phi$ back to Stable Diffusion's latent space. Qualitatively, the latents passed through $D2_\psi$ (second and third row in the figure 18) are close to the latents produced by Stable Diffusion's encoder E (first row in the figure 18). Quantitatively, we have an excellent reconstruction $z'_{s,p}$, with a PSNR consistently above 40, of the latents $z_{s,p}$ produced by the encoder E of Stable diffusion. It means that we have a bidirectional mapping between the latent space of Stable Diffusion and the 3D-aware latent space. In other words, we can map a latent image from the latent space of Stable Diffusion to a 3D-aware latent space and map it back to Stable Diffusion's latent space.

6.2.4 Poor quality of RGB images

As we can see in the figure 19, the RGB images obtained by decoding the latents produced by $D2_\psi$, by Stable Diffusion's decoder D are meaningful and we clearly recognise the groundtruth image. We match the quality obtained by the autoencoder of Stable Diffusion, but unfortunately, the visual quality of the RGB images are poor and some details, like the windows in figure 19,

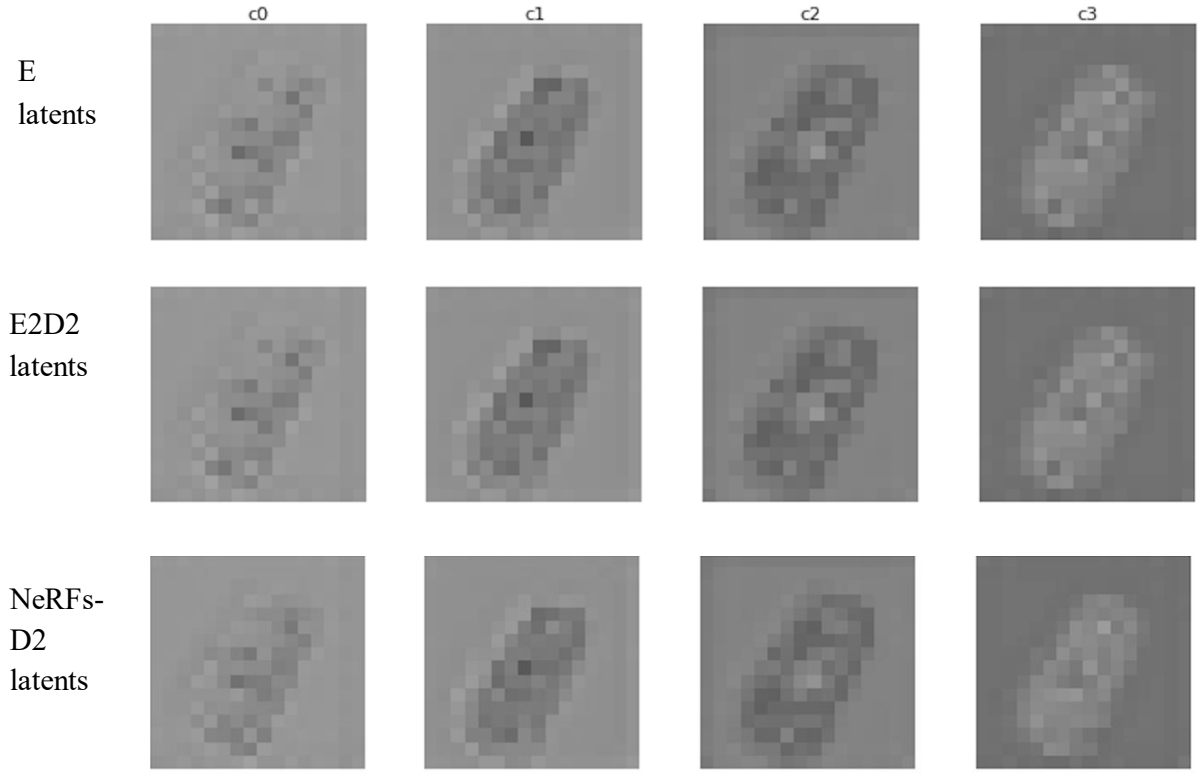

Figure 18: **Latent Space Alignment with E2D2.** The first row shows the latents produced by Stable Diffusion's encoder E, while the second row displays the latents outputed by the $D2_\psi$ block in the autoencoder pipeline in blue in figure 13. The last row also represents the latents outputed by $D2_\psi$, but in the 3D Regularisation pipeline in red in figure 13. The columns represent the four channels of the latent images.

are not reconstructed accurately.

The RGB images $\bar{x}_{s,p}$ obtained by the autoencoder pipeline shows a poor reconstruction of the details. When increasing the number of training scenes from 50 to 500 and increasing the weight of the latent reconstruction loss $\mathcal{L}_{\text{bridge}}(\phi, \theta)$, the PSNR of the RGB image $\bar{x}_{s,p}$ did not increase as visible in the column Full AE in the table 6.2.4. This likely indicates that increasing the number of scenes and modyfing the weights of losses is not enough to get better RGB images and that too much information is destroyed when passing through the autoencoder pipeline.

The very good results of the bidirectional mapping presented in the previous section indicates that the problem is not with the block $E2_\phi$ and $D2_\psi$, but with the blocks E and D that are directly interacting with RGB images. This intuition has been confirmed by encoding RGB images with E and directly decoding with D which gave visualy poor reconstruction results with important losses in details like the windows of the car (figure 19).

Considering this limitation of the encoder E and decoder D of Stable Diffusion, our results on RGB images are better than first thought. It is not reasonably possible to expect a good image quality for RGB images $\bar{x}_{s,p}$ that went through the whole autoencoder pipeline (in blue in 13) since they pass through E and D. This is encouraging since it means that we managed to reach the quality of the simple autoencoding (encoding with E and directly decoding with D), with

our more complex autoencoding pipeline (in blue in 13) that performs a bidirectional mapping between the latent of space of Stable Diffusion and a 3D-aware latent space as we can see in the equality of the PSNR values in the columns AE and Full AE of table 6.2.4.

Figure 19: **Comparison of RGB Reconstructions.** The first column shows a ground truth RGB image. The second column displays the RGB image reconstructed by the autoencoder pipeline (blue in Fig. 13). The third column shows the RGB images reconstructed by the 3D Regularisation pipeline (red in Fig. 13).

Table 1: PSNR values of the baseline and ablations.

Configuration	**Small AE**	**Full AE**	**RGB NeRF**	**Bridge**	**3D**
Baseline with 50 scenes	27.0	27.0	22.9	43.8	29.5
Baseline with 500 scenes + higher $\mathcal{L}_{\text{bridge}}$ weight	27.0	27.0	22.6	46.9	39.5
Removing $\mathcal{L}_{\text{rgb nerf}}$	27.0	27.0	21.5	44.6	33.9
Removing $\mathcal{L}_{\text{bridge nerf}}$	27.0	27.0	22.9	9.5	39.7

The PSNR values are averaged over test views. Small AE (autoencoder) evaluates the reconstruction quality of RGB images encoded by E and directly decoded by D. Full AE (autoencoder) corresponds to the quality of $\bar{x}_{s,p}$, i.e., RGB images processed through the entire autoencoder pipeline (highlighted in blue in Fig. 13). RGB NeRF measures the quality of $\hat{x}_{s,p}$, the RGB images obtained by decoding Tri-Plane latents using $D2_\psi$ followed by D. Bridge reports the reconstruction quality of $z'_{s,p}$, the latent produced by passing $z_{s,p}$ through E and $E2_\phi$. Finally, 3D quantifies the alignment between the outputs of $E2_\phi$ and the 3D-aware features extracted from the NeRFs.

6.3 Ablations

- **Removing $\mathcal{L}_{\text{rgb nerf}}$.** Removing the loss $\mathcal{L}_{\text{rgb nerf}}$ results in a decrease of 1.4 PSNR point on $\hat{x}_{s,p}$ (column RGB NeRF in table 6.2.4), while the PSNR for $\bar{x}_{s,p}$ remains constant at 27 (column Full AE in table 6.2.4). The ablation increases the PSNR of $z'_{s,p}$ from 43.8 to 44.6 (column Bridge in table 6.2.4), which can be attributed to the optimizer reallocating its capacity to other objectives. However, this gain is not problematic given that the reconstruction fidelity was very high. Consequently, we retain the $\mathcal{L}_{\text{rgb nerf}}$ term, as it

contributes to improving both the visual quality of the RGB images rendered by decoding the Tri-Planes and the alignment of the latents, $z_{s,p}$ and $\hat{z}_{s,p}$ while maintaining a good quality of alignment between $z_{s,p}$ and $z'_{s,p}$.

- **Removing $\mathcal{L}_{\text{bridge nerf}}$.** When removing the loss $\mathcal{L}_{\text{bridge nerf}}$, the PSNR of $\hat{x}_{s,p}$ and $\bar{x}_{s,p}$ remain unchanged, and the PSNR of $z'_{s,p}$ increases by 10.2 points. Nevertheless, this comes at the cost of a substantial 17-point drop in $\hat{z}_{s,p}$ PSNR. It is important to have a good PSNR for $\hat{z}_{s,p}$ because it is a proxy for assessing the quality of Tri-Planes produced latents, ensuring that they encode meaningful information rather than noise. Since NeRF latents reside in a 3D-aware space, they cannot be directly compared with Stable Diffusion latents; the D2 block provides a projection that enables such comparison. We remark that Stable Diffusion's decoder D still manages to reach the same quality of RGB even with those degraded latents as we can see in the column Full AE and RGB NeRF in table 6.2.4, but the problems of the latents outputed by the Tri-Planes become evident when looking at the unstructured and noisy latents in figure 20. Consequently, we retain the $\mathcal{L}_{\text{bridge nerf}}$ term to preserve the alignment between the latents $\hat{z}_{s,p}$ and $z_{s,p}$, and to ensure meaningful Tri-Planes outputs.

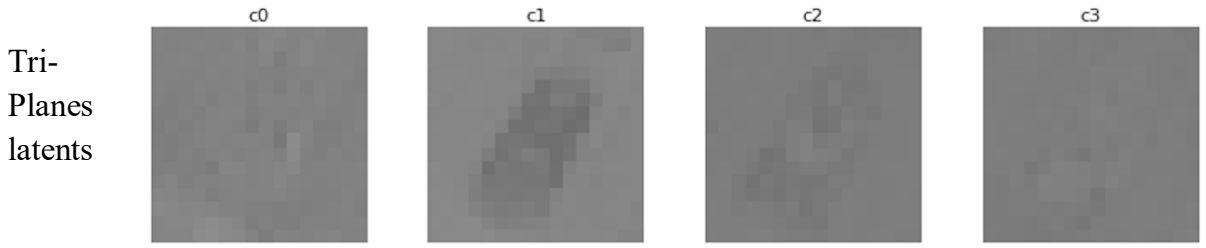

Figure 20: **Tri-Planes latents when removing $\mathcal{L}_{\text{bridge nerf}}$.** The latents outputted by the Tri-Planes become noisy and unstructured when we remove the loss $\mathcal{L}_{\text{bridge nerf}}$. The columns represent the four channels of the space.

- **Not pretraining Tri-Planes.** Similarly as Schnepf et al. [17], we observed that pretraining the Tri-Planes is absolutely necessary. Not pretraining the Tri-Planes distilled noise in the block $E2_\phi$ leading to catastrophic results where both Tri-Planes and $E2_\phi$ output uniformly gray latents. Consequently, we always pretrain the Tri-Planes.

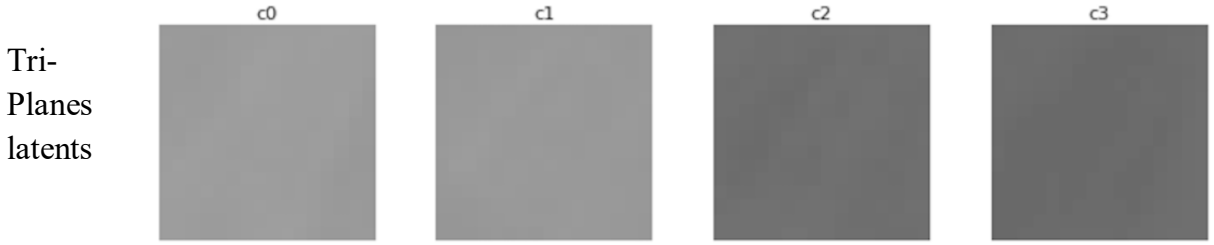

Figure 21: **Tri-Planes latents without pretraining and after 50 epochs of training.** Even after 50 epochs of training, the latents outputted by the Tri-Planes are uniformly gray when the Tri-Planes are not pretrained. The columns represent the four channels of the space.

7 Conclusion

This internship explored the alignment between 3D-aware latent spaces and Stable Diffusion's latent space, with the goal of making large pretrained 2D diffusion models usable for 3D computer vision tasks such as scene editing or generation. Building on the IG-AE [17] framework, the proposed E2D2 pipeline introduces a bidirectional mapping between a geometrically consistent 3D-aware latent space and the latent space of Stable Diffusion. This mapping also has the advantage of not requiring retraining for each scene. In addition, we proposed an adaptation of the latent NeRF pipeline presented in IG-AE [17] to train a NeRF with E2D2 on a new scene once E2D2 is trained, although this extension has not yet been implemented or experimentally validated.

The experiments demonstrated that E2D2 successfully learns the bidirectional mapping, producing smooth, 3D-aware latents while maintaining high-fidelity reconstructions in Stable Diffusion's latent space. The bidirectionality of the mapping was confirmed both qualitatively, through visual comparison, and quantitatively, with high PSNR values exceeding 40 dB, showing that a latent can be mapped from Stable Diffusion's space to a 3D-aware one and back. The ablation studies further validated the importance of each component of the model for achieving consistent and interpretable results.

Nevertheless, several limitations remain. The reconstruction quality of RGB images is limited, primarily due to the intrinsic limitations of Stable Diffusion's encoder-decoder when applied to our preprocessed synthetic data from ShapeNet, rather than the E2D2 architecture itself. The computational cost of training Tri-Planes and the limited diversity of the ShapeNet-Cars dataset also limit the generalisation of E2D2 to more complex scenes such as high-resolution real 3D scenes. Future work could address these limitations by experimenting with alternative autoencoders such as Ostris [1], expanding the dataset to more varied 3D scenes, integrating real images as in [17] to avoid overfitting on synthetic data, and trying more efficient 3D representations than Tri-Planes.

We hope that the proposed bidirectional mapping between 3D-aware and Stable Diffusion latent spaces will encourage the development of new pipelines leveraging the geometric coherence of 3D-aware latent spaces together with the generative power of Stable Diffusion to perform geometrically consistent 3D scene editing and generation.

8 Bibliography

References

[1] Jaret Burkett. *Ostris VAE – KL-f8-d16*. Online; available at https://huggingface.co/ostris/vae-kl-f8-d16. 2024.

[2] Eric R. Chan et al. "Efficient Geometry-aware 3D Generative Adversarial Networks". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 16102–16112. DOI: 10.1109/CVPR52688.2022.01565.

[3] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. 2015. arXiv: 1512.03012 [cs.GR]. URL: https://arxiv.org/abs/1512.03012.

[4] Anpei Chen et al. "TensoRF: Tensorial Radiance Fields". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 333–350. ISBN: 978-3-031-19824-3.

[5] Sara Fridovich-Keil et al. "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance". In: *CVPR*. 2023.

[6] Sara Fridovich-Keil et al. "Plenoxels: Radiance Fields without Neural Networks". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 5491–5500. DOI: 10.1109/CVPR52688.2022.00542.

[7] Gemma Frisius. *De radio astronomico et geometrico liber*. See fol. 39r for the first printed illustration of a camera obscura. Louvain: Apud G. Rescium, 1545. URL: https://archive.org/details/bub_gb_Ftk5AAAAcAAJ.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[9] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Trans. Graph.* 42.4 (July 2023). ISSN: 0730-0301. DOI: 10.1145/3592433. URL: https://doi.org/10.1145/3592433.

[10] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[11] Ricardo Martin-Brualla et al. "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 7206–7215. DOI: 10.1109/CVPR46437.2021.00713.

[12] Gal Metzer et al. “Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 12663–12673. DOI: 10.1109/CVPR52729.2023.01218.

[13] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *ECCV*. 2020.

[14] Thomas Müller et al. “Instant neural graphics primitives with a multiresolution hash encoding”. In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–15. ISSN: 1557-7368. DOI: 10.1145/3528223.3530127. URL: http://dx.doi.org/10.1145/3528223.3530127.

[15] Jangho Park, Gihyun Kwon, and Jong Chul Ye. “ED-NeRF: Efficient Text-Guided Editing of 3D Scene With Latent Space NeRF”. In: *International Conference on Learning Representations*. 2023. URL: https://api.semanticscholar.org/CorpusID:271746200.

[16] Robin Rombach et al. “ High-Resolution Image Synthesis with Latent Diffusion Models ”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2022, pp. 10674–10685. DOI: 10.1109/CVPR52688.2022.01042. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.01042.

[17] Antoine Schnepf et al. “Bringing NeRFs to the Latent Space: Inverse Graphics Autoencoder”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=LTDtjrv02Y.

[18] Alex Yu et al. “PlenOctrees for Real-time Rendering of Neural Radiance Fields”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 5732–5741. URL: https://api.semanticscholar.org/CorpusID:232352425.