NeuralField-LDM: Scene Generation with Hierarchical Latent Diffusion Models

Seung Wook Kim[1,2,3*] Bradley Brown[1,5*†] Kangxue Yin[1] Karsten Kreis[1] Katja Schwarz[6†]
Daiqing Li[1] Robin Rombach[7†] Antonio Torralba[4] Sanja Fidler[1,2,3]
[1]NVIDIA [2]University of Toronto [3]Vector Institute [4] CSAIL, MIT [5]University of Waterloo
[6]University of Tübingen, Tübingen AI Center [7]LMU Munich

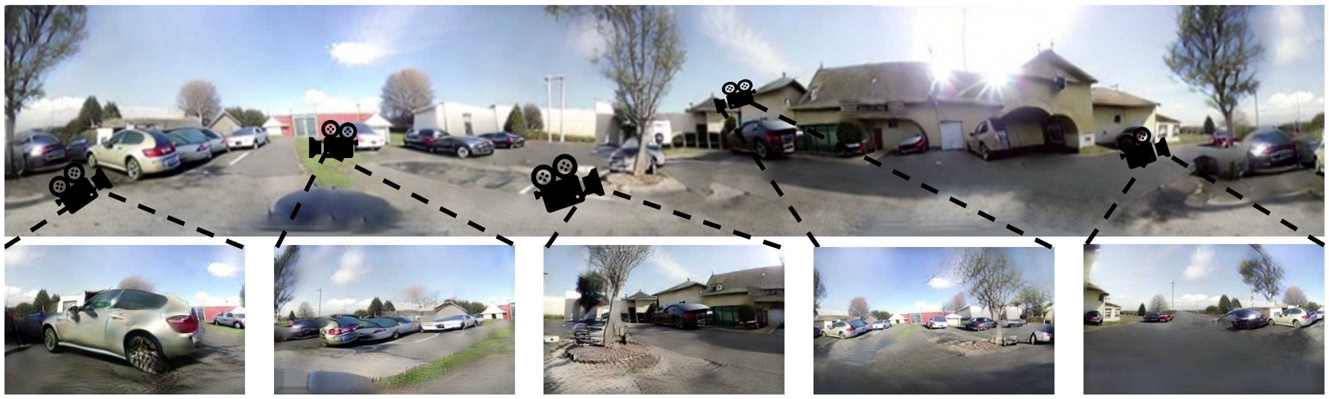

Figure 1. We introduce NeuralField-LDM, a generative model for complex open-world 3D scenes. This figure contains a panorama constructed from NeuralField-LDM's generated scene. We visualize different parts of the scene by placing cameras on them.

Abstract

Automatically generating high-quality real world 3D scenes is of enormous interest for applications such as virtual reality and robotics simulation. Towards this goal, we introduce NeuralField-LDM, a generative model capable of synthesizing complex 3D environments. We leverage Latent Diffusion Models that have been successfully utilized for efficient high-quality 2D content creation. We first train a scene auto-encoder to express a set of image and pose pairs as a neural field, represented as density and feature voxel grids that can be projected to produce novel views of the scene. To further compress this representation, we train a latent-autoencoder that maps the voxel grids to a set of latent representations. A hierarchical diffusion model is then fit to the latents to complete the scene generation pipeline. We achieve a substantial improvement over existing state-of-the-art scene generation models. Additionally, we show how NeuralField-LDM can be used for a variety of 3D content creation applications, including conditional scene generation, scene inpainting and scene style manipulation.

1. Introduction

There has been increasing interest in modelling 3D real-world scenes for use in virtual reality, game design, digital twin creation and more. However, designing 3D worlds by hand is a challenging and time-consuming process, requiring 3D modeling expertise and artistic talent. Recently, we have seen success in automating 3D content creation via 3D generative models that output individual object assets [17, 53, 85]. Although a great step forward, automating the generation of real-world scenes remains an important open problem and would unlock many applications ranging from scalably generating a diverse array of environments for training AI agents (*e.g.* autonomous vehicles) to the design of realistic open-world video games. In this work, we take a step towards this goal with NeuralField-LDM (NF-LDM), a generative model capable of synthesizing complex real-world 3D scenes. NF-LDM is trained on a collection of posed camera images and depth measurements which are easier to obtain than explicit ground-truth 3D data, offering a scalable way to synthesize 3D scenes.

Recent approaches [3, 7, 9] tackle the same problem of generating 3D scenes, albeit on less complex data. In [7, 9], a latent distribution is mapped to a set of scenes using adversarial training, and in GAUDI [3], a denoising diffusion model is fit to a set of scene latents learned using an auto-decoder. These models all have an inherent weakness of attempting to capture the entire scene into a single vector that conditions a neural radiance field. In practice, we find that this limits the ability to fit complex scene distributions.

Recently, diffusion models have emerged as a very powerful class of generative models, capable of generating high-

[*]Equal contribution.
[†]Work done during an internship at NVIDIA.

quality images, point clouds and videos [20, 27, 45, 55, 60, 85, 91]. Yet, due to the nature of our task, where image data must be mapped to a shared 3D scene without an explicit ground truth 3D representation, straightforward approaches fitting a diffusion model directly to data are infeasible.

In NeuralField-LDM, we learn to model scenes using a three-stage pipeline. First, we learn an auto-encoder that encodes scenes into a neural field, represented as density and feature voxel grids. Inspired by the success of latent diffusion models for images [60], we learn to model the distribution of our scene voxels in latent space to focus the generative capacity on core parts of the scene and not the extraneous details captured by our voxel auto-encoders. Specifically, a latent-autoencoder decomposes the scene voxels into a 3D coarse, 2D fine and 1D global latent. Hierarchichal diffusion models are then trained on the tri-latent representation to generate novel 3D scenes. We show how NF-LDM enables applications such as scene editing, birds-eye view conditional generation and style adaptation. Finally, we demonstrate how score distillation [53] can be used to optimize the quality of generated neural fields, allowing us to leverage the representations learned from state-of-the-art image diffusion models that have been exposed to orders of magnitude more data.

Our contributions are: 1) We introduce NF-LDM, a hierarchical diffusion model capable of generating complex open-world 3D scenes and achieving state of the art scene generation results on four challenging datasets. 2) We extend NF-LDM to semantic birds-eye view conditional scene generation, style modification and 3D scene editing.

2. Related Work

2D Generative Models In past years, generative adversarial networks (GANs) [4, 19, 31, 48, 65] and likelihood-based approaches [38, 56, 58, 78] enabled high-resolution photorealistic image synthesis. Due to their quality, GANs are used in a multitude of downstream applications ranging from steerable content creation [34, 39, 41, 42, 68, 89] to data driven simulation [30, 35, 36, 39]. Recently, autoregressive models and score-based models, e.g. diffusion models, demonstrate better distribution coverage while preserving high sample quality [11, 12, 15, 23, 25, 50, 55, 60, 61, 79]. Since evaluation and optimization of these approaches in pixel space is computationally expensive, [60, 79] apply them to latent space, achieving state-of-the-art image synthesis at megapixel resolution. As our approach operates on 3D scenes, computational efficiency is crucial. Hence, we build upon [60] and train our model in latent space.

Novel View Synthesis In their seminal work [49], Mildenhall et al. introduce Neural Radiance Fields (NeRF) as a powerful 3D representation. PixelNeRF [84] and IBRNet [82] propose to condition NeRF on aggregated features from multiple views to enable novel view synthesis from a sparse set of views. Another line of works scale NeRF to large-scale indoor and outdoor scenes [46, 57, 86, 88]. Recently, Nerfusion [88] predicts local radiance fields and fuses them into a scene representation using a recurrent neural network. Similarly, we construct a latent scene representation by aggregating features across multiple views. Different from the aforementioned methods, our approach is a generative model capable of synthesizing novel scenes.

3D Diffusion Models A few recent works propose to apply denoising diffusion models (DDM) [23, 25, 72] on point clouds for 3D shape generation [45, 85, 91]. While PVD [91] trains on point clouds directly, DPM [45] and LION [85] use a shape latent variable. Similar to LION, we design a hierarchical model by training separate conditional DDMs. However, our approach generates both texture and geometry of a scene without needing 3D ground truth as supervision.

3D-Aware Generative Models 3D-aware generative models synthesize images while providing explicit control over the camera pose and potentially other scene properties, like object shape and appearance. SGAM [69] generates a 3D scene by autoregressively generating sensor data and building a 3D map. Several previous approaches generate NeRFs of single objects with conditional coordinate-based MLPs [8, 51, 66]. GSN [9] conditions a coordinate-based MLP on a "floor plan", i.e. a 2D feature map, to model more complex indoor scenes. EG3D [7] and VoxGRAF [67] use convolutional backbones to generate 3D representations. All of these approaches rely on adversarial training. Instead, we train a DDM on voxels in latent space. The work closest to ours is GAUDI [3], which first trains an auto-decoder and subsequently trains a DDM on the learned latent codes. Instead of using a global latent code, we encode scenes onto voxel grids and train a hierarchical DDM to optimally combine global and local features.

3. NeuralField-LDM

Our objective is to train a generative model to synthesize 3D scenes that can be rendered to any viewpoint. We assume access to a dataset $\{(i, \kappa, \rho)\}_{1..N}$ which consists of N RGB images i and their camera poses κ, along with a depth measurement ρ that can be either sparse (*e.g.* Lidar points) or dense. The generative model must learn to model both the texture and geometry distributions of the dataset in 3D by learning solely from the sensor observations, which is a highly non-trivial problem.

Past work typically tackles this problem with a generative adversarial network (GAN) framework [7, 9, 66, 67]. They produce an intermediate 3D representation and render images for a given viewpoint with volume rendering [29, 49]. Discriminator losses then ensure that the 3D representation produces a valid image from any viewpoint. However, GANs come with notorious training instability

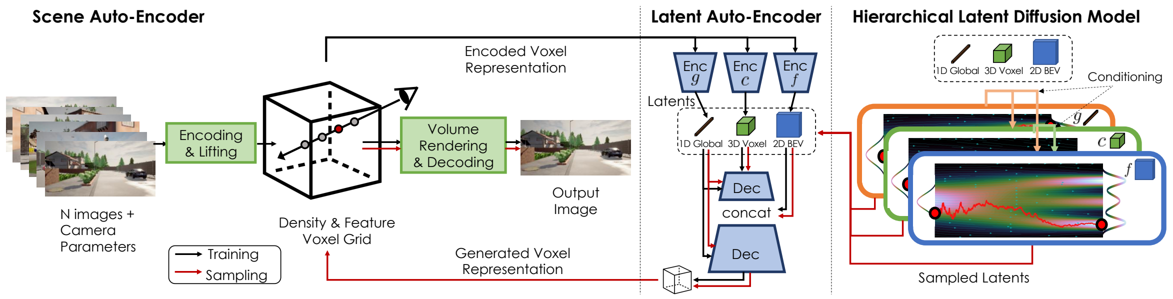
Scene Auto-Encoder
Latent Auto-Encoder
Hierarchical Latent Diffusion Model
Encoded Voxel Representation
Encoding & Lifting
Volume Rendering & Decoding
Output Image
N images + Camera Parameters
Density & Feature Voxel Grid
Training
Sampling
Generated Voxel Representation
Enc g
Enc c
Enc f
Latents
1D Global
3D Voxel
2D BEV
Dec
concat
Dec
Conditioning
Sampled Latents

Figure 2. **Overview of NeuralField-LDM**. We first encode RGB images with camera poses into a neural field represented by density and feature voxel grids. We compress the neural field into smaller latent spaces and fit a hierarchical latent diffusion model on the latent space. Sampled latents can then be decoded into a neural field that can be rendered into a given viewpoint.

and mode dropping behaviors [1, 18, 40]. Denoising Diffusion models [23] (DDMs) have recently emerged as an alternative to GANs that avoid the aforementioned disadvantages [60, 63, 64]. However, DDMs model the data likelihood explicitly and are trained to reconstruct the training data. Thus, they have been used in limited scenarios [85,90] since ground-truth 3D data is not readily available at scale.

To tackle the challenging problem of generating an entire scene with texture and geometry, we take inspiration from latent diffusion models (LDM) [60], which first construct an intermediate latent distribution of the training data then fit a diffusion model on the latent distribution. In Sec. A.1, we introduce a scene auto-encoder that encodes the set of RGB images into a neural field representation consisting of density and feature voxel grids. To accurately capture a scene, the voxel grids' spatial dimension needs to be much larger than what current state-of-the-art LDMs can model. In Sec. 3.2, we show how we can further compress and decompose the explicit voxel grids into compressed latent representations to facilitate learning the data distribution. Finally, Sec. 3.3 introduces a latent diffusion model that models the latent distributions in a hierarchical manner. Fig. 12 shows an overview of our method, which we name NeuralField-LDM (NF-LDM). We provide training and additional architecture details in the supplementary.

3.1. Scene Auto-Encoder

The goal of the scene auto-encoder is to obtain a 3D representation of the scene from input images by learning to reconstruct them. Fig. 3 depicts the auto-encoding process. The scene encoder is a 2D CNN and processes each RGB image $i_{1..N}$ separately, producing a $\mathbb{R}^{H\times W\times(D+C)}$ dimensional 2D tensor for each image, where H and W are smaller than i's size. We follow a similar procedure to Lift-Splat-Shoot (LSS) [52] to lift each 2D image feature map and combine them in the common voxel-based 3D neural field. We build a discrete frustum of size $H \times W \times D$ with the camera poses κ for each image. This frustum contains image features and density values for each pixel, along a pre-defined discrete set of D depths. Unlike LSS, we take the first D channels of the 2D CNN's output and use them as density values. That is, the d'th channel of the CNN's output at pixel (h, w) becomes the density value of the frustum entry at (h, w, d). Motivated by the volume rendering equation [49], we get the occupancy weight O of each element (h, w, d) in the frustum using the density values $\sigma \geq 0$:

$$O(h,w,d) = \exp(-\sum_{j=0}^{d-1} \sigma_{(h,w,j)}\delta_j)(1-\exp(-\sigma_{(h,w,d)}\delta_d)) \tag{1}$$

where h, w denotes the pixel coordinate of the frustum and δ_j is the distance between each depth in the frustum. Using the occupancy weights, we put the last C channels of the CNN's output into the frustum F:

$$F(h,w,d) = [O(h,w,d)\phi(h,w), \sigma(h,w,d)] \tag{2}$$

where $\phi(h, w)$ denotes the C-channeled feature vector at pixel (h, w) which is scaled by $O(h, w, d)$ for F at depth d.

After constructing the frustum for each view, we transform the frustums to world coordinates and fuse them into a shared 3D neural field, represented as density and feature voxel grids. Let $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$ denote the density and feature grid, respectively. This formulation of representing a scene with density and feature grids has been explored before [74] for optimization-based scene reconstruction and we utilize it as an intermediate representation for our scene auto-encoder. $V_{\texttt{Density,Feat}}$ have the same spatial size, and each voxel in V represents a region in the world coordinate system. For each voxel indexed by (x, y, z), we pool all densities and features of the corresponding frustum entries. In this paper, we simply take the mean of the pooled features. More sophisticated pooling functions (*e.g.* attention) can be used, which we leave as future work.

Finally, we perform volume rendering using the camera poses κ to project V onto a 2D feature map. We trilinearly interpolate the values on each voxel to get the feature and density for each sampling point along the camera rays. 2D features are then fed into a CNN decoder that produces the output image $\hat{i}$. We denote rendering of voxels to output images as $\hat{i} = r(V, \kappa)$. From the volume rendering process, we also get the expected depth $\hat{\rho}$ along each ray [57]. The scene

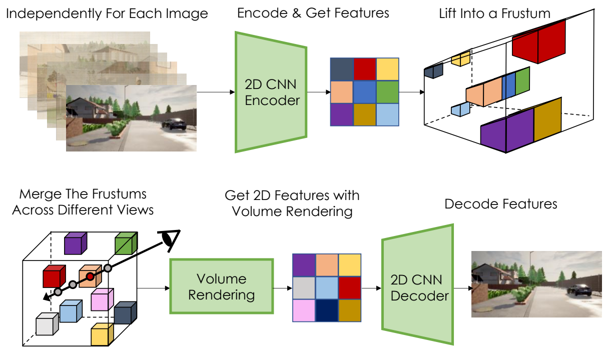
Independently For Each Image
Encode & Get Features
Lift Into a Frustum
2D CNN Encoder
Merge The Frustums Across Different Views
Get 2D Features with Volume Rendering
Decode Features
Volume Rendering
2D CNN Decoder

Figure 3. **Scene Auto-Encoder:** Each input image is processed with a 2D CNN then lifted up to 3D and merged into the shared voxel grids. Density prediction is not shown here for brevity.

auto-encoding pipeline is trained with an image reconstruction loss $||i - \hat{i}||$ and a depth supervision loss $||\rho - \hat{\rho}||$. In the case of sparse depth measurements, we only supervise the pixels with recorded depth. We can further improve the quality of the auto-encoder with adversarial loss as in VQ-GAN [15] or by doing a few optimization steps at inference time, which we discuss in the supplementary.

3.2. Latent Voxel Auto-Encoder

It is possible to fit a generative model on voxel grids obtained from Sec. A.1. However, to capture real-world scenes, the dimensionality of the representation needs to be much larger than what SOTA diffusion models can be trained on. For example, Imagen [63] trains DDMs on 256×256 RGB images, and we use voxels of size $128 \times 128 \times 32$ with 32 channels. We thus introduce a latent auto-encoder (LAE) that compresses voxels into a 128-dimensional global latent as well as coarse (3D) and fine (2D) quantized latents with channel dimensions of four and spatial dimensions $32 \times 32 \times 16$ and 128×128 respectively.

We concatenate $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$ along the channel dimension and use separate CNN encoders to encode the voxel grid V into a hierarchy of three latents: 1D global latent g, 3D coarse latent c, and 2D fine latent f, as shown in Fig. 12. The intuition for this design is that g is responsible for representing the global properties of the scene, such as the time of the day, c represents coarse 3D scene structure, and f is a 2D tensor with the same horizontal size $X \times Y$ as V, which gives further details for each location (x, y) in bird's eye view perspective. We empirically found that 2D CNNs perform similarly to 3D CNNs while being more efficient, thus we use 2D CNNs throughout. To use 2D CNNs for the 3D input V, we concatenate V's vertical axis along the channel dimension and feed it to the encoders. We also add latent regularizations to avoid high variance latent spaces [60]. For the 1D vector g, we use a small KL-penalty via the reparameterization trick [38], and for c and f, we impose a vector-quantization [15, 80] layer to regularize them.

The CNN decoder is similarly a 2D CNN, and takes c, concatenated along vertical axis, as the initial input. The decoder uses conditional group normalization layers [83] with g as the conditioning variable. Lastly, we concatenate f to an intermediate tensor in the decoder. The latent decoder outputs $\hat{V}$ which is the reconstructed voxel. LAE is trained with the voxel reconstruction loss $||V - \hat{V}||$ along with the image reconstruction loss $||i - \hat{i}||$ where $\hat{i} = r(\hat{V}, \kappa)$. Note that the image reconstruction loss only helps with the learning of LAE, and the scene auto-encoder is kept fixed.

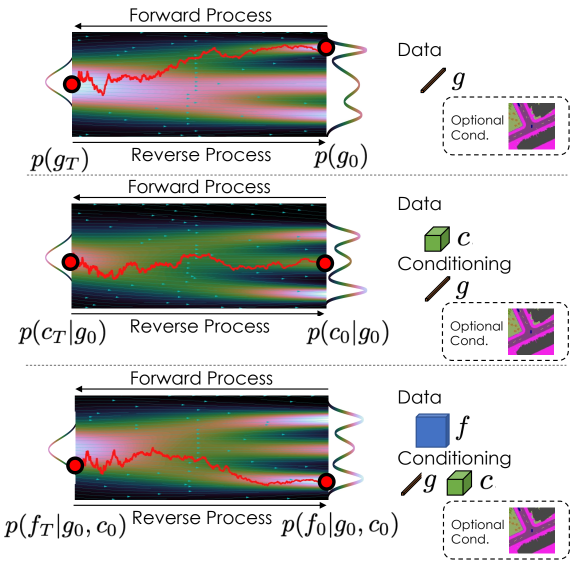

Figure 4. **Hierarchical LDM. Top:** LDM ψ_g for KL-regulairzed global latent g. **Middle:** LDM ψ_c for vector-quantized coarse latent c. **Bottom:** LDM ψ_f for vector-quantized fine latent f. All LDMs optionally take an additional conditioning variable as input, such as a Bird's Eye View segmentation map as depicted here.

3.3. Hierarchical Latent Diffusion Models

Given the latent variables g, c, f that represent a voxel-based scene representation V, we define our generative model as $p(V, g, c, f) = p(V|g, c, f)p(f|g, c)p(c|g)p(g)$ with Denoising Diffusion Models (DDMs) [24]. In general, DDMs with discrete time steps have a fixed Markovian forward process $q(x_t|x_{t-1})$ where $q(x_0)$ denotes the data distribution and $q(x_T)$ is defined to be close to the standard normal distribution, where we use the subscript to denote the time step. DDMs then learn to revert the forward process $p_\theta(x_{t-1}|x_t)$ with learnable parameters θ. It can be shown that learning the reverse process is equivalent to learning to denoise x_t to x_0 for all timesteps t [24, 27] by reducing the following loss:

$$\mathop{\mathbb{E}}_{t,\epsilon,x_0}\left[w(\lambda_t)||x_0 - \hat{x}_\theta(x_t, t)||_2^2\right] \quad (3)$$

where t is sampled from a uniform distribution for timesteps, ϵ is sampled from the standard normal to noise the data x_0, $w(\lambda_t)$ is a timestep dependent weighting constant, and $\hat{x}_\theta$ denotes the learned denoising model.

We train our hierarchical LDM with the following losses:

$$\mathbb{E}_{t,\epsilon,g_0}\Big[w(\lambda_t)||g_0 - \psi_g(g_t,t)||_2^2\Big] \quad (4)$$

$$\mathbb{E}_{t,\epsilon,g_0,c_0}\Big[w(\lambda_t)||c_0 - \psi_c(c_t,g_0,t)||_2^2\Big] \quad (5)$$

$$\mathbb{E}_{t,\epsilon,g_0,c_0,f_0}\Big[w(\lambda_t)||f_0 - \psi_f(f_t,g_0,c_0,t)||_2^2\Big] \quad (6)$$

where ψ denotes the learnable denoising networks for g, c, f. Fig. 4 visualizes the diffusion models. ψ_g is implemented with linear layers with skip connections and ψ_c and ψ_f adopt the U-net architecture [62]. g is fed into ψ_c and ψ_f with conditional group normalization layers. c is interpolated and concatenated to the input to ψ_f. The camera poses contain the trajectory the camera is travelling, and this information can be useful for modelling a 3D scene as it tells the model where to focus on generating. Therefore, we concatenate the camera trajectory information to g and also learn to sample it. For brevity, we still call the concatenated vector g. For conditional generation, each ψ takes the conditioning variable as input with cross-attention layers [60].

Each ψ can be trained in parallel and, once trained, can be sampled one after another following the hierarchy. In practice, we use the v-prediction parameterization [64] that has been shown to have better convergence and training stability [63, 64]. Once g, c, f are sampled, we can use the latent decoder from Sec. 3.2 to construct the voxel V which represents the neural field for the sampled scene.. Following the volume rendering and decoding step in Sec. A.1, the sampled scene can be visualized from desired viewpoints.

3.4. Post-Optimizing Generated Neural Fields

Samples generated from our model on real-world data contain reasonable texture and geometry (Fig. 10), but can be further optimized by leveraging recent advances in 2D image diffusion models trained on orders of magnitude more data. Specifically, we iteratively update initially generated voxels, V, by rendering viewpoints from the scene and applying Score Distillation Sampling (SDS) [53] loss on each image independently:

$$\nabla_V L_{SDS} = \mathbb{E}_{\epsilon,t,\kappa}\Big[w(\lambda_t)(\epsilon - \hat{\epsilon_\theta}(r(V,\kappa),t))\frac{\partial r(V,\kappa)}{\partial V}\Big] \quad (7)$$

where κ is sampled uniformly in a $6m^2$ region around the origin of the scene with random rotation about the vertical axis, $w(\lambda_t)$ is the weighting schedule used to train $\hat{\epsilon_\theta}$ and $t \sim U[0.02T, 0.2T]$ where T is the amount of noise steps used to train $\hat{\epsilon_\theta}$. Note that for latent diffusion models, the noise prediction step is applied after encoding $r(V,\kappa)$ to the LDM's latent space and the partial gradient term is updated appropriately. For $\hat{\epsilon_\theta}$, we use an off-the-shelf latent diffusion model [60], finetuned to condition on CLIP image embeddings [54][1]. We found that CLIP contains a representation of the quality of images that the LDM is able to interpret: denoising an image while conditioning on CLIP image embeddings of our model's samples produced images with similar geometry distortions and texture errors. We leverage this property by optimizing L_{SDS} with negative guidance. Letting y, y' be CLIP embeddings of clean image conditioning (*e.g.* dataset images) and artifact conditioning (*e.g.* samples) respectively, we perform classifier-free guidance [26] with conditioning vector y, but replace the unconditional embedding with y'. As shown in the supplementary, this is equivalent (up to scale) to sampling from $\frac{p(x|y)^\alpha}{p(x|y')}$ at each denoising step where α controls the trade-off between sampling towards dataset images and away from images with artifacts. We stress that this post-optimization is only successful due to the strong scene prior contained in our voxel samples, as shown by our comparison to running optimization on randomly initialized voxels in the supplementary.

[1]https://github.com/justinpinkney/stable-diffusion

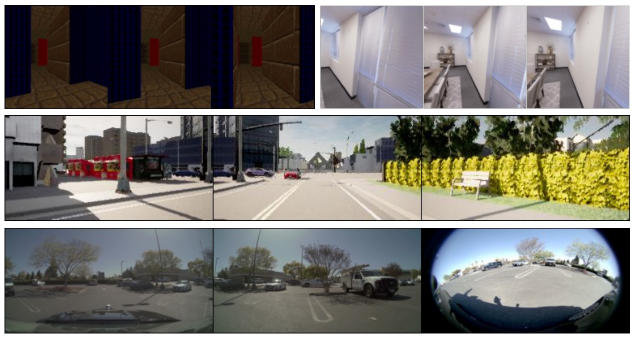
Figure 5. **Datasets:** Top-left: VizDoom [33]. Top-right: Replica [73]. Middle: Carla [13] . Bottom: AVD. For Carla and AVD, we visualize a subset of available cameras.

4. Experiments

We evaluate NeuralField-LDM on the following four datasets (Fig. 5). Each dataset contains RGB images and a depth measurement with their corresponding camera poses.

VizDoom [32] consists of front-view sensor observations obtained by navigating inside a synthetic game environment. We use the dataset provided by [10], which contains 34 trajectories with a length of 600.

Replica [73] is a dataset of high-quality reconstructions of 18 indoor scenes, containing 101 front-view trajectories with a length of 100. We use the dataset provided by [10].

Carla [13] is an open-source simulation platform for self-driving research. We mimic the camera settings for a self-driving car, by placing six cameras *(front-left, front, front-right, back-left, back, back-right)*, covering 360 degrees with some overlaps, and move the car in a randomly sampled direction and distance for 10 timesteps. We sample 43K datapoints, each containing 60 images.

AVD is an in-house dataset of human driving recordings in roads and parking lots. It has ten cameras with varying

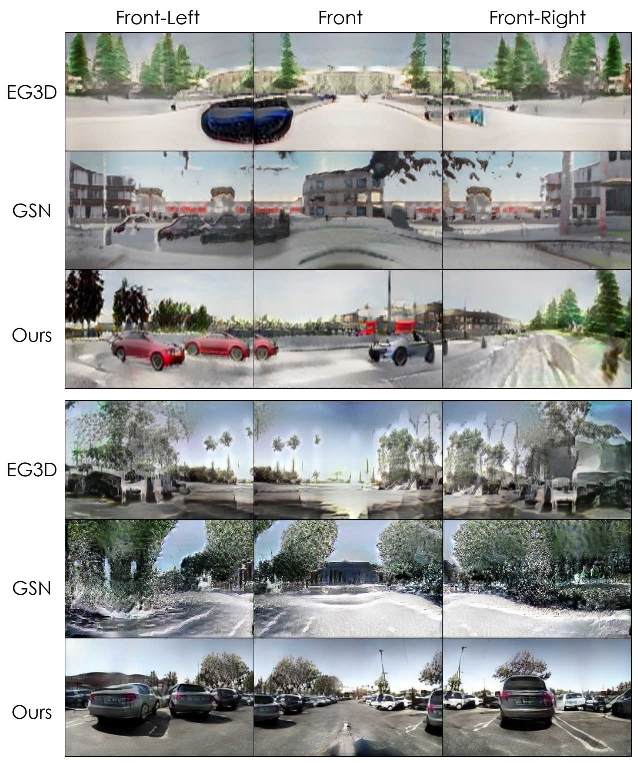

Figure 6. **Generated Scenes:** The top three rows are samples from Carla, and the bottom three rows are samples from AVD.

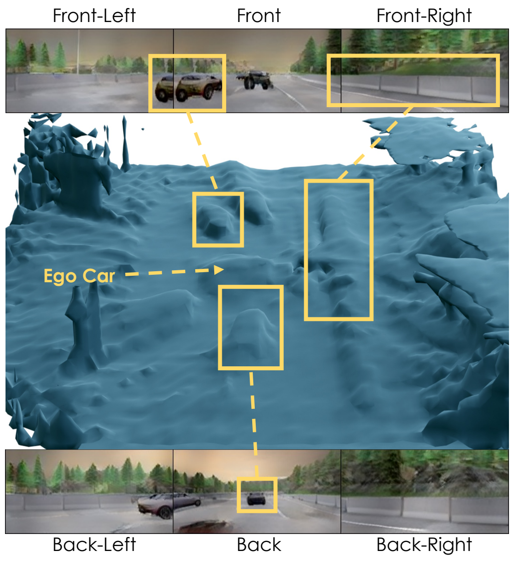

Figure 7. We run marching-cubes [43] on the density voxels to visualize the geometry of the samples generated by NF-LDM. The structure of the scene is reflected well in the mesh.

lens types along with Lidar for depth measurement. It has 73K sequences, each with 8 frames extracted at 10 fps.

4.1. Baseline Comparisons

Unconditional Generation We evaluate the unconditional generation performance of NF-LDM by comparing it with baseline models. All results are without the post-

Criterion	Method	Depth	VizDoom	Replica
FID (↓)	GRAF [66]	✗	47.50	65.37
	π-GAN [5]	✗	143.55	166.55
	GSN [9]	✓	37.21	41.75
	GAUDI [3]	✓	33.70	18.75
	NF-LDM	✓	**19.54***	**14.59**

Table 1. FID [22] scores on VizDoom and Replica. NF-LDM outperforms all baseline models. Baseline numbers are from [3].

Criterion	Method	Depth	Carla	AVD
FID (↓)	EG3D [6]	✗	76.89	194.34
	GSN [9]	✓	75.45	166.07
	NF-LDM	✓	**35.69**	**54.26**

Table 2. FID [22] scores on Carla and AVD datasets. Baseline models have trouble learning the distribution of complex outdoor datasets, in particular AVD, while NF-LDM models them well.

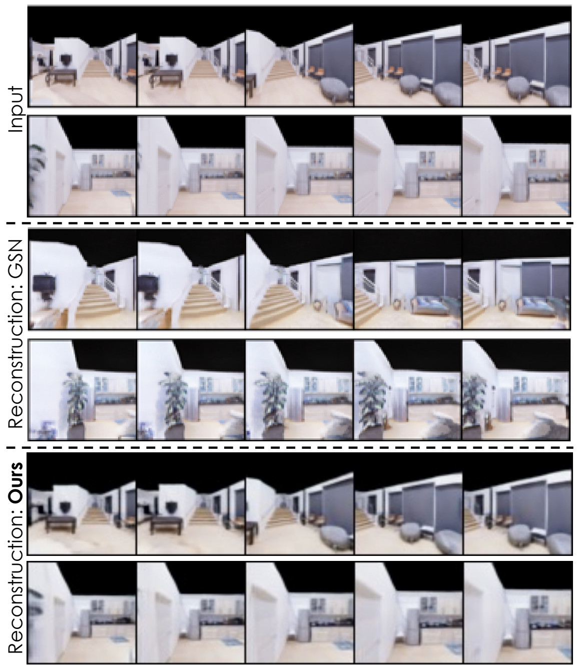

Figure 8. Reconstructing held-out scenes not seen during training.

optimization step (Sec. 3.4), unless specified. Tab. 1 shows the results on VizDoom and Replica. GRAF [66] and π-GAN [5], which do not utilize ground truth depth in training, have shown successes in modelling single objects, but they exhibit worse performance than others that leverages depth information for modelling scenes. GAUDI [3] is an auto-decoder-based diffusion model. Their auto-decoder optimizes a small per-scene latent to reconstruct its matching scene. GAUDI comes with the advantage that learning the generative model is simple as it only needs to model the small dimensional latent distribution that acts as the key to their corresponding scenes. On the contrary, NF-LDM is trained on the latents that are a decomposition of the explicit 3D neural field. Therefore, GAUDI puts more modelling

Criterion	Method	Depth	Carla	AVD
FVD (↓)	EG3D [6]	✗	134.94	1232.38
	GSN [9]	✓	184.30	1659.81
	NF-LDM	✓	**91.80**	**242.50**

Table 3. FVD [77] scores on Carla and AVD Datasets. As for FID, baseline models have trouble learning to model complex datasets.

capacity into the auto-decoder part, and NF-LDM puts it more into the generative model part. We attribute our improvement over GAUDI to our expressive hierarchical LDM that can model the details of the scenes better. In VizDoom, only one scene exists, and each sequence contains several hundred steps covering a large area in the scene, which our voxels were not large enough to encode. Therefore, we chunked each VizDoom trajectory to be 50 steps long.

Tab. 2 shows results on complex outdoor datasets: Carla and AVD. We compare with EG3D [6] and GSN [9]. Both are GAN-based 3D generative models, but GSN utilizes ground truth depth measurements. Note that we did not include GAUDI [3] as the code was not available. NF-LDM achieves the best performance, and both baseline models have difficulty modelling the real outdoor dataset (AVD). Fig. 6 compares the samples from different models.

Since the datasets are composed of frame sequences, we can treat them as videos and further evaluate with Fréchet Video Distance (FVD) [77] to compare the distributions of the dataset and sampled sequences. This can quantify samples' 3D structure by how natural the rendered sequence from a moving camera is. For EG3D and GSN, we randomly sample a trajectory from the datasets and for NF-LDM, we sample a trajectory from the global latent diffusion model. Tab. 3 shows that NF-LDM achieves the best results. We empirically observed GSN sometimes produced slightly inconsistent rendering, which could attribute to its lower FVD score than EG3D's. We also visualize the geometry of NF-LDM's samples by running marching-cubes [43] on the density voxels. Fig. 7 shows that our samples produce a coarse but realistic geometry.

Ablations We evaluate the hierarchical structure of NF-LDM. Tab. 4 shows an ablation study on Carla. The model with the full hierarchy achieves the best performance. The global latent makes it easier for the other LDMs to sample as conditioning on the global properties of the scene (*e.g.* time of day) narrows down the distribution they need to model. The 2D fine latent helps retain the residual information missing in the 3D coarse latent, thus improving the latent auto-encoder and, consequently, the LDMs.

Scene Reconstruction Unlike previous approaches, NF-LDM has an explicit scene auto-encoder that can be used for scene reconstruction. GAUDI [3] is auto-decoder based, so it is not trivial to infer a latent for a new scene. GSN [9] can invert a new scene using a GAN inversion method [59, 92], but as Fig. 8 shows, it fails to get the details of the scene correct. Our scene auto-encoder generalizes well and is scalable as the number of scenes grow.

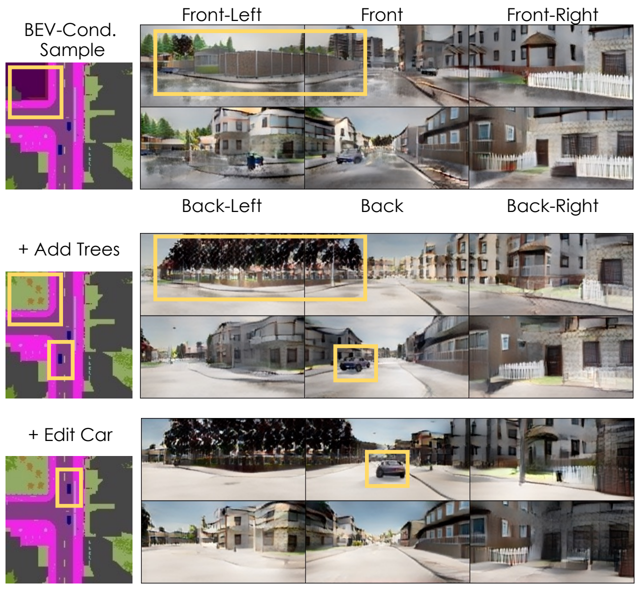

Figure 9. **BEV-Conditioned Synthesis**: NF-LDM allows controllable generation by editing the BEV segmentation map. From the initial sample, we add trees (green) and then edit the location of the car (blue). Note the ego car is at the center and thus not rendered.

	Coarse lat. c	+ Fine lat. f	+ Global lat. g
FID (↓)	46.43	43.52	**35.69**

Table 4. FID [22] on ablating the choice of hierarchy on the Carla dataset. The first column is for training both LAE and LDM only with the coarse latent. The last column is our full model.

4.2. Applications and Limitations

Conditional Synthesis NF-LDM can utilize additional conditioning signals for controllable generation. In this paper, we consider Bird's Eye View (BEV) segmentation maps, but our model can be extended to other conditioning variables. We use cross attention layers [60], which have been shown to be effective for conditional synthesis. Fig. 28 shows that NF-LDM follows the given BEV map faithfully and how the map can be edited for controllable synthesis.

Scene Editing Image diffusion models can be used for image inpainting without explicit training on the task [60, 72]. We leverage this property to edit scenes in 3D by resampling a region in the 3D coarse latent c. Specifically, at each denoising step, we noise the region to be kept and concatenate with the region being sampled, and pass it through the diffusion model. We use reconstruction guidance [27] to better harmonize the sampled and kept regions. After we get a new c, the fine latent is also re-sampled conditioned on c. Fig. 11 shows results on scene editing with NF-LDM.

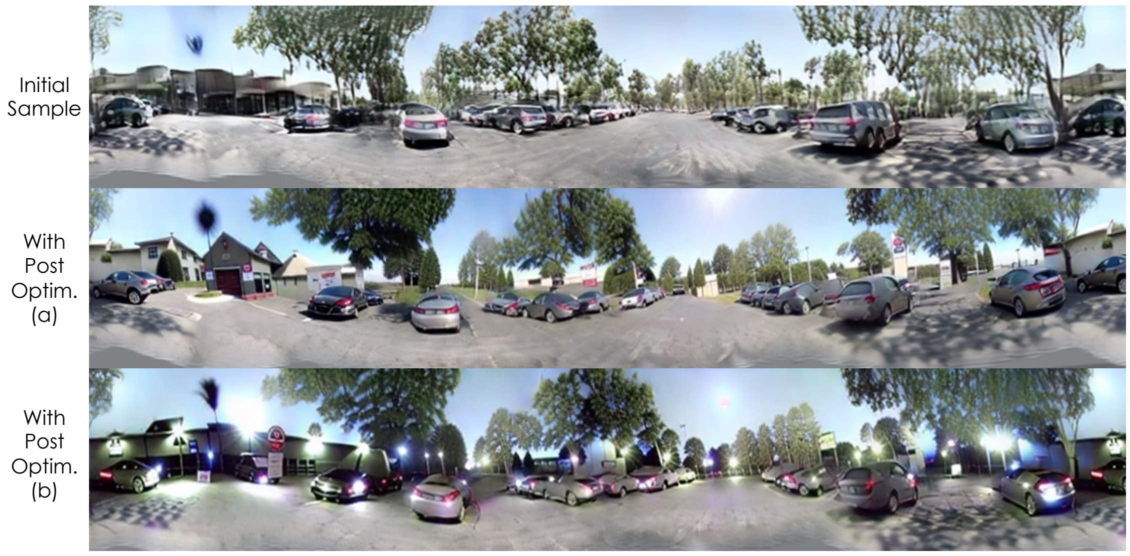

Figure 10. **Panoramas from NF-LDM's samples:** From the initial sample at the top, we apply post-optimization with Score Distillation Sampling [53] (Sec. 3.4). (a) demonstrates improved sample quality. (b) showcases style modification by conditioning on evening scenes.

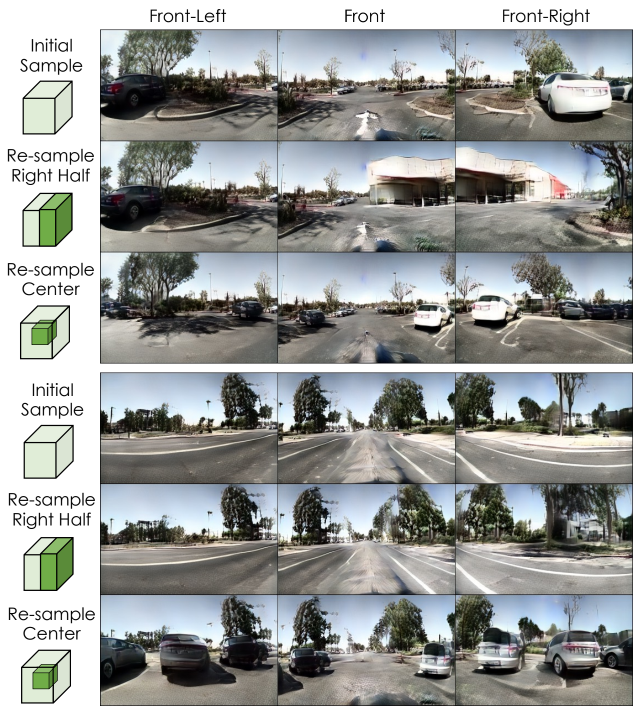

Figure 11. **Scene Editing**: We use the 3D coarse latent c for scene editing. From the initial sample indicated by light green, we re-sample a part of the latent, indicated by dark green.

Post-Optimization Fig. 10 shows how post-optimization (Sec. 3.4) can improve the quality of NF-LDM's initial sample while retaining the 3D structure. In addition to improving quality, we can also modify scene properties, such as time of day and weather, by conditioning the LDM on images with the desired properties. SDS-based style modification is effective for changes where a set of clean image data is available with the desired property and is reasonably close to our dataset's domain (*e.g*. street images for AVD). In the supplementary, we also provide results experimenting with directional CLIP loss [16] to quickly finetune our scene decoder for a given text prompt.

Limitations NF-LDM's hierarchical structure and three stage pipeline allows us to achieve high-quality generations and reconstructions, but it comes with a degradation in training time and sampling speed. In this work, the neural field representation is based on dense voxel grids, and it becomes expensive to volume render and learn the diffusion models as they get larger. Therefore, exploring alternative sparse representations is a promising future direction. Lastly, our method requires multi-view images which limits data availability and therefore risks universal problems in generative modelling of overfitting. For example, we found that output samples in AVD had limited diversity because the dataset itself was recorded in a limited number of scenes.

5. Conclusion

We introduced NeuralField-LDM (NF-LDM), a generative model for complex 3D environments. NF-LDM first constructs an expressive latent distribution by encoding input images into a 3D neural field representation which is further compressed into more abstract latent spaces. Then, our proposed hierarchical LDM is fit onto the latent spaces, achieving state-of-the-art performance on 3D scene generation. NF-LDM enables a diverse set of applications, including controllable scene generation and scene editing. Future directions include exploring more efficient sparse voxel representations, training on larger-scale real-world data and learning to continously expand generated scenes.

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*, 2017. 3

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 13

[3] Miguel Ángel Bautista, Pengsheng Guo, Samira Abnar, Walter Talbott, Alexander Toshev, Zhuoyuan Chen, Laurent Dinh, Shuangfei Zhai, Hanlin Goh, Daniel Ulbricht, Afshin Dehghan, and Josh M. Susskind. GAUDI: A neural architect for immersive 3d scene generation. 2022. 1, 2, 6, 7

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 2

[5] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. *arXiv*, 2012.00926, 2020. 6

[6] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021. 6, 7

[7] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*, 2022. 1, 2

[8] Eric R. Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. Pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *CVPR*, 2021. 2

[9] Terrance DeVries, Miguel Ángel Bautista, Nitish Srivastava, Graham W. Taylor, and Joshua M. Susskind. Unconstrained scene generation with locally conditioned radiance fields. In *ICCV*, 2021. 1, 2, 6, 7, 14

[10] Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W. Taylor, and Joshua M. Susskind. Unconstrained scene generation with locally conditioned radiance fields. *arXiv*, 2021. 5

[11] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *NeurIPS*, 2021. 2

[12] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped langevin diffusion. In *ICLR*, 2022. 2

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. 2017. 5

[14] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. 14

[15] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 2, 4, 14, 15

[16] Rinon Gal, Or Patashnik, Haggai Maron, Gal Chechik, and Daniel Cohen-Or. Stylegan-nada: Clip-guided domain adaptation of image generators, 2021. 8, 37

[17] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022. 1

[18] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016. 3

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 2

[20] William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Weilbach, and Frank Wood. Flexible diffusion modeling of long videos, 2022. 2

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 14

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017. 6, 7

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. 2, 3

[24] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. 4, 15, 16

[25] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *J. Mach. Learn. Res.*, 2022. 2

[26] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 5

[27] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv:2204.03458*, 2022. 2, 4, 7

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of the International Conf. on Machine learning (ICML)*, 2015. 13

[29] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984. 2

[30] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *ICCV*, 2019. 2

[31] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *CVPR*, 2020. 2, 13, 14

[32] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2016. 5

[33] Margret Keuper, Siyu Tang, Bjoern Andres, Thomas Brox, and Bernt Schiele. Motion segmentation & multiple object tracking by correlation co-clustering. *IEEE TPAMI*, 2018. 5

[34] Seung Wook Kim, Karsten Kreis, Daiqing Li, Antonio Torralba, and Sanja Fidler. Polymorphic-gan: Generating aligned samples across multiple domains with learned morph maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022. 2

[35] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. In *CVPR*, 2021. 2

[36] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020. 2

[37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Machine learning (ICML)*, 2015. 15

[38] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014. 2, 4, 14

[39] Daiqing Li, Huan Ling, Seung Wook Kim, Karsten Kreis, Adela Barriuso, Sanja Fidler, and Antonio Torralba. Bigdatasetgan: Synthesizing imagenet with pixel-wise annotations. *arXiv*, 2201.04684, 2022. 2

[40] Ke Li and Jitendra Malik. On the implicit assumptions of gans. *arXiv preprint arXiv:1811.12402*, 2018. 3

[41] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3d controllable image synthesis. In *CVPR*, 2020. 2

[42] Huan Ling, Karsten Kreis, Daiqing Li, Seung Wook Kim, Antonio Torralba, and Sanja Fidler. Editgan: High-precision semantic image editing. In *NeurIPS*, 2021. 2

[43] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Trans. on Graphics*, 1987. 6, 7, 19, 25

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 17

[45] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, 2021. 2

[46] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021. 2

[47] Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021. 26

[48] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *Proc. of the International Conf. on Machine learning (ICML)*, 2018. 2, 14

[49] 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. 2, 3

[50] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *Proc. of the International Conf. on Machine learning (ICML)*, 2022. 2

[51] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 2

[52] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *ECCV*, 2020. 3, 19

[53] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022. 1, 2, 5, 8

[54] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv*, 2103.00020, 2021. 5, 17

[55] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv*, abs/2204.06125, 2022. 2

[56] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *NeurIPS*, 2019. 2

[57] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Thomas A. Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *CVPR*, 2022. 2, 3

[58] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proc. of the International Conf. on Machine learning (ICML)*, 2014. 2

[59] Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. Encoding in style: a stylegan encoder for image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2287–2296, 2021. 7

[60] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 2, 3, 4, 5, 7, 14, 16, 17, 26

[61] Robin Rombach, Patrick Esser, and Björn Ommer. Geometry-free view synthesis: Transformers and no 3d priors. In *ICCV*, 2021. 2

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 5, 14, 17

[63] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022. 3, 4, 5

[64] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 3, 5, 16

[65] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. *ACM Trans. on Graphics*, 2022. 2

[66] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. 2020. 2, 6

[67] Katja Schwarz, Axel Sauer, Michael Niemeyer, Yiyi Liao, and Andreas Geiger. Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids. In *NeurIPS*, 2022. 2

[68] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, 2020. 2

[69] Yuan Shen, Wei-Chiu Ma, and Shenlong Wang. SGAM: Building a virtual 3d world through simultaneous generation and mapping. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. 2

[70] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. 15

[71] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. 17

[72] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. 2, 7, 15, 16

[73] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv*, 1906.05797, 2019. 5

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

[75] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. of the International Conf. on Machine learning (ICML)*, 2019. 13

[76] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv*, 1607.08022, 2016. 13

[77] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018. 7

[78] Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. 2

[79] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. In *NeurIPS*, 2021. 2

[80] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 4, 14, 15

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 14

[82] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas A. Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021. 2

[83] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 4, 14

[84] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2

[85] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. LION: latent point diffusion models for 3d shape generation. In *NeurIPS*, 2022. 1, 2, 3

[86] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv*, 2010.07492, 2020. 2

[87] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 13

[88] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. In *CVPR*, 2022. 2

[89] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. In *ICLR*, 2021. 2

[90] Yan Zheng, Lemeng Wu, Xingchao Liu, Zhen Chen, Qiang Liu, and Qixing Huang. Neural volumetric mesh generator. *arXiv preprint arXiv:2210.03158*, 2022. 3

[91] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *ICCV*, 2021. 2

[92] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain gan inversion for real image editing. In *European conference on computer vision*, pages 592–608. Springer, 2020. 7

Supplementary Materials for
NeuralField-LDM: Scene Generation with Hierarchical Latent Diffusion Models

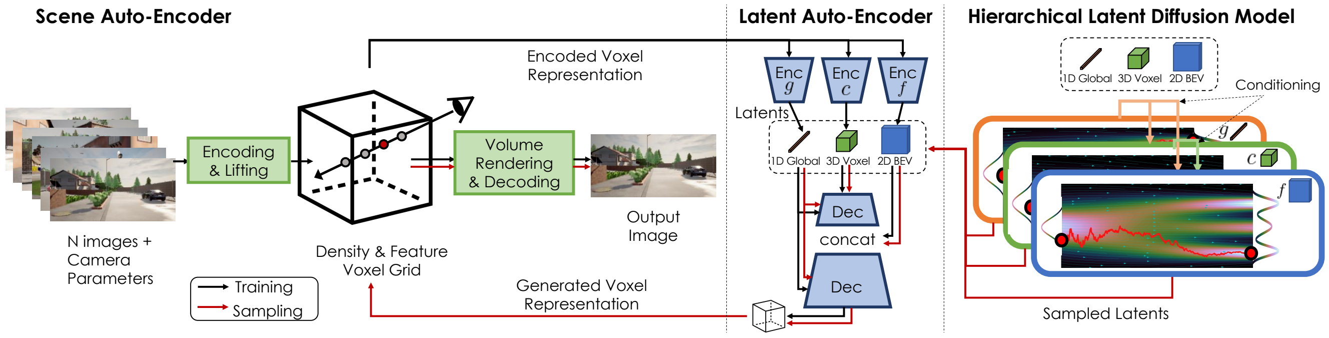
Scene Auto-Encoder
Latent Auto-Encoder
Hierarchical Latent Diffusion Model
Encoded Voxel Representation
Encoding & Lifting
Volume Rendering & Decoding
Output Image
N images + Camera Parameters
Density & Feature Voxel Grid
Training
Sampling
Generated Voxel Representation
Enc g
Enc c
Enc f
Latents
1D Global
3D Voxel
2D BEV
Dec
concat
Dec
Conditioning
Sampled Latents

Figure 12. **Overview of NeuralField-LDM**. We put the model overview from the main text for reference. We first encode RGB images with camera poses into a neural field represented by density and feature voxel grids. We compress the neural field into smaller latent spaces and fit a hierarchical latent diffusion model on the latent space. Sampled latents can then be decoded into a neural field that can be rendered into a given viewpoint.

This supplementary document is organized as follows:

- Section A includes additional model architecture and training details.
- Section B includes additional qualitative results.
- We also include a supplementary video which has better visualizations for evaluating the 3D aspect of our model.

A. Model Architecture and Training Details

We use the convention $C \times Z \times X \times Y$ to denote a 3D tensor with C channels, where the Z-axis points in the upward direction in 3D. Similarly, $C \times H \times W$ denotes a 2D tensor with C channels, with height H and width W. In practice, we found that perceptual loss [87] works better than L1 or L2 loss as an image reconstruction loss, and we use it throughout the paper as the loss function for image reconstruction.

A.1. Scene Auto-Encoder

At every iteration, the scene auto-encoder takes as input $\{(i, \kappa, \rho)\}_{1..(N+M)}$ consisting of $N + M$ RGB images i from the same scene along with their known camera posses κ and depth measurements ρ, which can either be sparse (*e.g.* Lidar points) or dense. We use all cameras as supervision, but only use the first N cameras as input to the model.

Encoder Architecture. We encode each of these N images independently through an EfficientNet-B1 [75] encoder, replacing Batchnorm layers [28] with Instance normalization [76]. Additionally, the second and eigth blocks' padding and stride are modified to preserve spatial resolutions so that each image is downsampled by a factor of eight. The EfficientNet-B1 head is replaced with a bilinear upsampling layer followed by a concatenation with the features before the last downsampling layers along the channel dimension and then two consecutive Conv2d, ReLU, BatchNorm2d layers, where the first Conv2D layer increases the channel dimension from 432 to 512. These features are then processed by two seperate two-layer Conv networks that reduce the channel dimensions to C and D, producing the feature and density values respectively for each pixel. These feature and density values are used to define the frustum as explained in the Section 3.1 of the main text. We clamp the density to lie in $[-10, 10]$ and apply the softplus activation function. Each voxel in the density and feature voxel grids $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$ represents a region in the world coordinate system. For all datasets considered in this paper, we define the dimension of voxels to be $32 \times 128 \times 128$ $(Z \times X \times Y)$. For VizDoom, each voxel represents a region of (4 game unit)^3. For Replica, each voxel represents a region of $(0.125m)^3$. For Carla, each voxel represents a region of $(0.75m)^3$. For AVD, we use non-uniform voxel sizes. The voxels at the center have $0.2m$ side length, and the furthest voxels from the center have $1.6m$ horizontal side length and $2.4m$ vertical side length.

Decoder Architecture. We perform volumetric rendering, using the Mip-NeRF [2] implementation on $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$ using $\{(\kappa)\}_{1..(N+M)}$ to get target features. These features are then fed through a decoder, using the blocks in StyleGAN2 [31] to produce output image predictions $\{\hat{i}\}_{1..(N+M)}$. The decoder consists of ten StyledConv blocks, where the convolution operation of the fourth layer is replaced with a transposed convolution to upsample the features by a factor of 2. A StyledConv block contains a style modulation layer [31], but we effectively skip the modulation process by feeding in a constant vector of 1s.

Training. The parameters of the encoder and decoder are trained with an image construction loss, $||i-\hat{i}||$ across all $N+M$ inputs with a coefficient of 1. We also supervise the expected depth obtained from volumetric rendering with an MSE loss on pixels that contain a ground truth depth measurement weighted with a coefficient of 5. Finally, we also add a regularization term on the sum over the entropy of all sampled opacity values from volumetric rendering to encourage very high or low values in the density voxels, weighted with a coefficient of 0.01. For all models, we use the Adam optimizer with a learning rate of 0.0002 and betas of $(0., 0.99)$. After training, we are able to further improve image quality by adding adversarial loss. We use StyleGAN2's [31] discriminator along with an R1 gradient regularization [48]. Furthermore, to capture the missing details from the encoding step while ensuring the distribution of training voxels does not diverge, we optionally perform a small number of additional per-scene optimization steps on the encoded voxels V. Specifically, for VizDoom, Replica and AVD, we perform 60 optimization steps by randomly sampling input views and reducing the image reconstruction loss, per encoded scene voxel.

Camera Settings. For VizDoom and Replica, we directly use the camera settings used in GSN [9]. For Replica, we use all 100 consecutive frames per training sequence, and for VizDoom, as the area each sequence covers was too large for our voxel size, we chunk each training sequence into 50 consequtive frames. For Carla, at every iteration we sample a scene and a camera. We sample $N+M=9$ consecutive frames from that scene and camera as our scene-encoder input, and randomly sample $N=6$ of those frames to input into the encoder. We do this so we can obtain information across multiple timesteps, without incurring the memory cost of using all camera views at every iteration. At inference time, for a given scene, we encode frames for all viewpoints at every timestep. For AVD, we create a set of 5 groups, each comprised of overlapping cameras. We sample $N+M=8$ consecutive frames from a sampled scene and camera group as our scene-encoder input. We use $N=5$ fish-eye cameras as input to the encoder as they have the largest field-of-view and so the encoder does not have to learn to process different types of cameras. We use all cameras for the losses. We use histogram equalization on the input images. At inference time for AVD, we encode only the fish-eye cameras.

A.2. Latent Voxel Auto-Encoder

We concatenate $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$ along the channel dimension and use separate CNN encoders to encode the voxel grid V into a hierarchy of three latents: 1D global latent g, 3D coarse latent c, and 2D fine latent f, as shown in Fig. 12. The intuition for this design is that g is responsible for representing the global properties of the scene, such as the time of the day, c represents coarse 3D scene structure, and f is a 2D tensor with the same horizontal size $X \times Y$ as V, which gives further details for each location (x, y) in bird's eye view perspective. We empirically found that 2D CNNs perform similarly to 3D CNNs while being more efficient, thus we use 2D CNNs throughout. To use 2D CNNs for the 3D input V, we concatenate V's vertical axis along the channel dimension and feed it to the encoders.

Encoder Architecture. We use the building blocks of the encoder architecture from VQGAN [15]. Tables 1-3 contain the descriptions of the encoder architectures. Resblocks [21] contain two convolution layers and each conv layer has a group normalization [83] and a SiLU activation [14] prior to it. AttnBlocks are implemented as self-attention modules [81] and MidBlocks represent a block of {ResBlock, AttnBlock, ResBlock}. We add latent regularizations to avoid high variance latent spaces [60]. For the 1D vector g, we use a small KL-penalty via the reparameterization trick [38], and for c and f, we impose a vector-quantization [15, 80] layer. c is quantized with a codebook containing 1024 entries, and f is quantized with a codebook containing 128 entries. Blocks that end with "-CGN" have group normalization layers replaced with conditional group normalization and they take in the global latent g as the conditioning input. Blocks that start with "Unet-" have a unet connection [62] from their counterpart downsampling blocks that have the same feature dimension. For example, in the encoder for f, the Unet-ResBlocks take in the features of the first few ResBlocks and concatenate them to their input.

Decoder Architecture. The latent decoder architecture is presented in Table 4. It is similarly a 2D CNN, and takes c, concatenated along the vertical axis, as the initial input. It also uses conditional group normalization layers with g as the conditioning variable. The fine latent f is combined with an intermediate tensor in the decoder. This process is represented as "Combine f" in the table. Specifically, we expand the channel dimension of f to 128 with a 3×3 Conv2D layer, and concatenate with the output tensor of the previous block. Then, it goes through three ResBlock-CGN layers to output a $512 \times 128 \times 128$ tensor. Finally, the tensor goes through a Conv2D layer and then is reshaped to the reconstructed voxel $\hat{V}$.

Training. The LAE is trained with the voxel reconstruction loss $||V-\hat{V}||$ along with the image reconstruction loss $||i-\hat{i}||$ where $\hat{i}=r(\hat{V}, \kappa)$. Note that the image reconstruction loss only helps with learning the LAE, and the scene auto-encoder

Layer	Output dimension
Input V (3D)	$32 \times 32 \times 128 \times 128$
Concat Z-axis	$(32 \times 32) \times 128 \times 128$
Conv2D 3×3	$128 \times 128 \times 128$
$6 \times \{$ResBlock	
ResBlock	$128 \times 2 \times 2$
Conv2D 3×3 stride 2$\}$	
ResBlock	$128 \times 2 \times 2$
ResBlock	$128 \times 2 \times 2$
AttnBlock	$128 \times 2 \times 2$
MidBlock	$128 \times 2 \times 2$
Conv2D 2×2	$256 \times 1 \times 1$
Reparameterization (1D)	128

Table 5. Encoder for the global latent g

Layer	Output dimension
Input V (3D)	$32 \times 32 \times 128 \times 128$
Concat Z-axis	$(32 \times 32) \times 128 \times 128$
Conv2D 3×3	$512 \times 128 \times 128$
$2 \times \{$ResBlock	
ResBlock	$512 \times 32 \times 32$
Conv2D 3×3 stride 2$\}$	
ResBlock	$512 \times 32 \times 32$
ResBlock	$512 \times 32 \times 32$
AttnBlock	$512 \times 32 \times 32$
MidBlock	$512 \times 32 \times 32$
Conv2D 3×3	$32 \times 32 \times 32$
Split Z-axis	$4 \times 8 \times 32 \times 32$
Quantization (3D)	$4 \times 8 \times 32 \times 32$

Table 6. Encoder for the coarse latent c

Layer	Output dimension
Input V (3D)	$32 \times 32 \times 128 \times 128$
Concat Z-axis	$(32 \times 32) \times 128 \times 128$
Conv2D 3×3	$256 \times 128 \times 128$
$2 \times \{$ResBlock	
ResBlock	$256 \times 32 \times 32$
Conv2D 3×3 stride 2$\}$	
MidBlock	$256 \times 32 \times 32$
Conv2D 3×3	$32 \times 32 \times 32$
Unet-MidBlock	$256 \times 32 \times 32$
$2 \times \{$Unet-ResBlock-CGN	
Unet-ResBlock-CGN	
ResBlock-CGN	$256 \times 128 \times 128$
Upsample2×$\}$	
Conv2D 3×3	$4 \times 128 \times 128$
Quantization (2D)	$4 \times 128 \times 128$

Table 7. Encoder for the fine latent f

Layer	Output dimension
Input c (3D)	$4 \times 8 \times 32 \times 32$
Concat Z-axis	$(4 \times 8) \times 32 \times 32$
Conv2D 3×3	$512 \times 32 \times 32$
MidBlock-CGN	$512 \times 32 \times 32$
ResBlock-CGN	$512 \times 32 \times 32$
ResBlock-CGN	$512 \times 32 \times 32$
ResBlock-CGN	$512 \times 32 \times 32$
$2 \times \{$ResBlock-CGN	
ResBlock-CGN	
ResBlock-CGN	$512 \times 128 \times 128$
Upsample2×$\}$	
Combine f	$512 \times 128 \times 128$
Conv2D 3×3	$1024 \times 128 \times 128$
Split Z-axis (3D)	$32 \times 32 \times 128 \times 128$

Table 8. Decoder of the latent auto-encoder

is kept fixed. For the voxel reconstruction loss, we divide V into two groups. One group contains empty voxels that does not encode any information, and the other group have voxels filled in from the scene-autoencoding step in Section A.1. The reconstruction loss is equally weighted between the two groups (*i.e.*, we take the mean of the losses for the two groups separately and add them up). We use different weightings for $V_{\texttt{Density}}$ and $V_{\texttt{Feat}}$. The reconstruction loss for $V_{\texttt{Density}}$ is weighted 2.5× higher to encourage the model to reconstruct the geometry of the scene well. We use a small KL coeffcient 2e-05 for g which is multiplied to the KL loss. We use a coefficient of 1.0 for the vector-quantization losses [15,80] for c and f. The image reconstruction loss is multiplied by 10. We train the LAE with the Adam optimizer [37] with a learning rate of 0.0002.

A.3. Hierarchical Latent Diffusion Models

Background on Denoising Diffusion Models Denoising Diffusion Models [24,70,72] (DDMs) are trained with denoising score matching to model a given data distribution $q(x_0)$. DDMs sample a diffused input $x_t = \alpha_t x + \sigma_t \epsilon,\ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ from a data point $x \sim q(x_0)$ where α_t and σ_t define a time t-dependent noise schedule. The schedule is pre-defined such that the logarithmic signal-to-noise ratio $\log(\alpha_t^2/\sigma_t^2)$ decreases monotonically. Now, a neural network model ψ is trained to denoise

	VizDoom	Replica	Carla	AVD
Global Latent Dimension	128	128	128	128
Trajectory Dimension	200	400	18	24
Number of Linear Blocks	10	10	10	6
Channel dimension	2048	2048	512	2048
Learning Rate	5e-05	5e-05	5e-05	5e-05

Table 9. Hyperparameters for ψ_g. Each training sequence in VizDoom consists of 50 timesteps, each with three-dimensional (x, y, z) location information and one-dimensional yaw information totalling 200 dimensions per trajectory. Similarly, Replica has 100 timesteps, totalling 400 dimensions per trajectory. Carla has the same z location for the Z-axis across different timesteps, so we only model the (x, y) trajectory information from nine consecutive timesteps. For AVD, we model all three (x, y, z) translation parameters across eight timesteps, totalling 24 dimensions per trajectory.

the diffused input by reducing the following loss

$$\mathbb{E}_{x\sim q(x_0),t\sim p_t,\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|y-\psi(x_t;t)\|_2^2\right], \tag{8}$$

where the target y is either the sampled noise ϵ or $v = \alpha_t\epsilon - \sigma_t x$. We use the latter target v following [64] which empirically demonstrates faster convergence. p_t denotes the distribution over time t and we use a uniform discrete time distribution $p_t \sim \mathcal{U}\{0, 1000\}$, following [24] . We use the *variance-preserving* noise schedule [72], for which $\sigma_t^2 = 1 - \alpha_t^2$.

Global Latent Diffusion Model. The global LDM ψ_g is implemented with linear blocks where each block is a residual block with skip connections:

$$\begin{aligned}
&h = linear(x)\\
&h_{emb} = linear(t_{emb})\\
&h = h + h_{emb}\\
&h = linear(h)\\
&return\ linear(x) + h
\end{aligned} \tag{9}$$

Here, x is the input to the block and t_{emb} is the timestep embedding for the diffusion time step t. We follow [60] to get the embedding. We have N such linear blocks. The inputs to the second half of the linear blocks are the concatenation of the previous block's output and the output of the corresponding first half of the linear block in a U-net fashion as depicted in Figure 13.

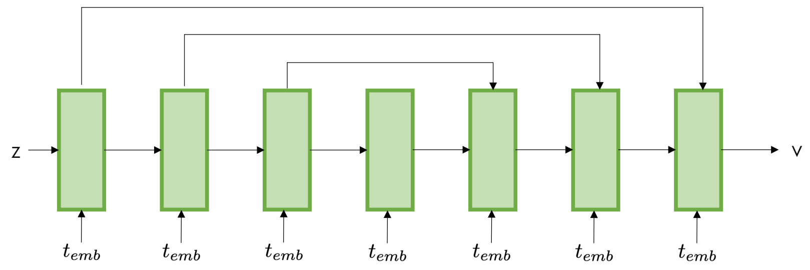

Figure 13. Architecture diagram of ψ_g. z is the input to the network and v is the output. The green blocks are the linear blocks with skip connections (Eq. 9). The model is a 1D analogous version of the 2D Unet commonly used in 2D diffusion models.

As mentioned in the main text, the input to ψ_g is both g and the camera trajectory information which is flattened to 1D. For Carla and AVD, we implement ψ_g as two separate networks that have the same architecture for the global latent and the camera trajectory. For Replica and VizDoom, we use a single network to model both the global latent and the camera trajectory as they are highly correlated (*e.g.* we found that each global latent in Replica represents a scene in the training dataset and trajectories should be sampled within the given scene, as otherwise, it could go out of the bound of the scene). Table 9 contains the hyperparameter choices for ψ_g.

	VizDoom	Replica	Carla	AVD
Input Shape	$4 \times 8 \times 32 \times 32$	$4 \times 8 \times 32 \times 32$	$4 \times 8 \times 32 \times 32$	$4 \times 8 \times 32 \times 32$
Channels	224	128	288	256
Channel Multiplier	1,2,3,4	1,2,3,4	1,2,3,4	1,2,3,4
Attention Resolutions	4,8,16	4,8,16	4,8,16	4,8,16
Learning Rate	6.4e-05	6.4e-05	6.4e-05	6.4e-05

Table 10. Hyperparameters for ψ_c. Channels denote the base number of channels. Each group of layers (four groups in our case as indicated by the number of channel multipliers) in the Unet (see [60] for further details) have the number of channels equal to the base channels multiplied by the corresponding channel multiplier. Attention layers are applied at the specified 2D spatial resolutions. The tensor with the smallest spatial resolution in the Unet has 4×4 spatial resolution.

	VizDoom	Replica	Carla	AVD
Input Shape	$4 \times 128 \times 128$	$4 \times 128 \times 128$	$4 \times 128 \times 128$	$4 \times 128 \times 128$
Channels	128	128	288	512
Channel Multiplier	1,2,2,2,4,4	1,2,2,2,4,4	1,2,2,2,4,4	1,1,1,1,1,1
Attention Resolutions	16,32,64	16,32,64	16,32,64	8,16,32
Learning Rate	6.4e-05	6.4e-05	6.4e-05	6.4e-05

Table 11. Hyperparameters for ψ_f. Channels denote the base number of channels. Each group of layers (six groups in our case as indicated by the number of channel multipliers) in the Unet (see [60] for further details) has the number of channels equal to the base channels multiplied by the corresponding channel multiplier. Attention layers are applied at the specified 2D spatial resolutions. The tensor with the smallest spatial resolution in the Unet has a 4×4 spatial resolution.

Coarse and Fine Latent Diffusion Model. ψ_c and ψ_f adopt the U-net architecture [62] and closely follow the 2D Unet architecture used in [60]. The input to ψ_c is 3D but we concatenate it along the Z-axis and use the 2D Unet architecture without introducing 3D components. The output is split along the channel dimension to recover the 3D output shape. ψ_f also takes in c as the conditioning input. We first concatenate c along the Z-axis, making its shape $32 \times 32 \times 32$, and then interpolate it to match the spatial dimension of f to be a tensor with shape $32 \times 128 \times 128$. Finally, the interpolated c is concatenated to f (so the shape of the concatenated tensor is $36 \times 128 \times 128$) and fed into the Unet model whose output matches the shape of f, $4 \times 128 \times 128$. Similar to ψ_g, ψ_c and ψ_f also take the timestep embedding t_{emb} for the sampled diffusion time step t. Table 10 and Table 11 contain the hyperparameter settings for ψ_c and ψ_f, respectively.

The cross attention layers in [60] are equivalent to self-attention layers if no extra conditioning information is given. For Bird's eye view (BEV) segmentation conditioned models, we additionally train a 2D convolution encoder network that takes in the segmentation map with size $\mathbb{R}^{3\times128\times128}$ and produces a BEV embedding with size $\mathbb{R}^{256\times32\times32}$. This BEV embedding is fed into the cross attention layers for conditional synthesis for ψ_c and ψ_f. For ψ_g, we take the mean of the embedding across the spatial dimension, and concatenate with the timestep embedding that goes into the linear blocks.

Training. We follow [60] for the choice of diffusion steps (1000), noise schedule (linear), and optimizer (AdamW [44]) for all experiments. For sampling, we use the DDIM sampler [71] with 250 steps.

A.4. Post-Optimizing Generated Neural Fields

Given a set of voxels V, obtained either through sampling or by encoding a set of views, we are able to increase the quality of V through post-optimization using SDS loss as shown in Figures 16 and 17. For the entire optimization, we use a fixed set of camera parameters $\{\kappa\}_{1\ldots N}$ sampled from the training dataset scene as the base camera position where, for AVD, $N = 6$ and all intrinsic matrices are replaced with the camera intrinsic parameters from the non-fisheye left-facing camera. At every iteration, we uniformly sample a translation offset in both the forwards and sideways directions between -3 and 3 metres as well as a rotation offset about the Z-axis uniformly between -10 and 10 degrees. We apply these offsets to $\{\kappa\}_{1\ldots N}$ to obtain $\{\hat{\kappa}\}_{1\ldots N}$, and render out images $\hat{i} = r(V, \hat{\kappa})$ for each viewpoint. We then either use random cropping or left/right cropping to make the aspect ratio square and bilinearly resize $\hat{i}$ to 512×512 resolution, matching the required input dimensions for the diffusion model. We obtain the gradient for the voxels using Equation 7 in the main text, leaving the decoder parameters fixed.

We use an off-the-shelf latent diffusion model [60], finetuned to condition on CLIP image embeddings [54]. We train with negative guidance, as detailed in Section A.4.1. For the positive conditioning, we sample $23k$ images from the front, left and right facing non-fisheye cameras from our dataset and take the average of their CLIP image embeddings. For the negative conditioning, we sample 80 voxels from our model and use the average CLIP image embeddings from $23k$ images rendered from the voxels using the same camera jitter distribution we use for post-optimization. We attempted to use classifier-free guidance without the negative conditioning, but found the outputs to be blurry as seen in Figure 27. At every update, we uniformly sample the noising timestep, $t \in [20, 200]$, independently for each image in the batch.

We train with a batch-size of 3 and a gradient accumulation of 2 steps, fixing the cameras in the even updates and odd updates so every gradient step contains updates from every camera view exactly once. We use the Adam optimizer with a learning rate of 1e-3, betas of $(0.9, 0.99)$ and epsilon set to 1e-15. We optimize a single scene for $20k$ iterations, taking approximately 13 hours on a single $V100$ GPU, but also see drastic quality improvements after $2k$ iterations.

We note that as seen in Figure 27, having a voxel initialization sampled or encoded from our model is critical to the success of post-optimization.

A.4.1 Negative-guidance

Let y, y' be positive conditioning (*e.g.* dataset image) and negative conditioning (*e.g.* sampled images with artifacts), respectively, and x a diffusion-step sample. Intuitively, we want to sample the diffusion model so that $p(x|y)$ is high and $p(x|y')$ is low. Thus, we want to sample from $\frac{p(x|y)^\alpha}{p(x|y')}$ where α trades off the importance of sampling towards y and away from y'. We see then that:

$$\nabla_x \log \frac{p(x|y)^\alpha}{p(x|y')} = \alpha \nabla_x \log p(x|y) - \nabla_x \log p(x|y')$$

which is equal to classifier free guidance with $\gamma = 2$, $\alpha = \gamma = 2$ and the unconditional embedding replaced with y'.

For reference, classifier-free guidance is defined as:

$$\gamma \nabla_x \log p(x|y) + (1-\gamma) \nabla_x \log p(x)$$

Empirically, we implement classifier-free guidance and replace the uncondtional embedding with y' which, as shown below, is equivalent to setting $\alpha = \frac{\gamma}{\gamma-1}$ and multiplying the gradient by $(\gamma - 1)$:

$$\begin{aligned}\gamma \nabla_x \log p(x|y) + (1-\gamma) \nabla_x \log p(x|y') &= \gamma \nabla_x \log p(x|y) - (\gamma - 1) \nabla_x \log p(x|y')) \\ &= (\gamma - 1)(\frac{\gamma}{\gamma - 1} \nabla_x \log p(x|y) - \nabla_x \log p(x|y')) \\ &= (\gamma - 1) \nabla_x \log \frac{p(x|y)^{\frac{\gamma}{\gamma-1}}}{p(x|y')}\end{aligned}$$

	$32 \times 32 \times 8$	$64 \times 64 \times 16$	$128 \times 128 \times 32$
Percept. Loss (↓)	0.3508	0.2688	**0.2237**

Table 12. Ablation of the voxel dimensions of the scene autoencoder. We report the validation perceptual loss.

	$ds = 16$	$ds = 8$	$ds = 4$
Vox. Recon Loss (↓)	0.6076	0.5949	**0.4915**

Table 13. Ablation of the downsampling factors (ds) of the latent autoencoder. We report the validation voxel reconstruction loss.

B. Additional Results

B.1. More Ablations

(1) Scene Encoder: the voxel size used by the scene encoder is crucial in capturing details of the scene. If we use larger voxel size and encoder frustum size, the voxel would be able to contain more pixel-level detail and consequently output better quality images. However, this comes with the disadvantage that modelling such high-dimensional voxel space with a generative model becomes challenging. In Fig. 14, we show samples from a diffusion model fit to our first-stage voxels for Carla. We hypothesize that current DMs cannot perform well on very high dimensional data, highlighting the importance of our hierarchical latent space. Tab. 12 reports perceptual loss on reconstructed output viewpoints. We concluded that $128 \times 128 \times 32$ provides a satisfactory output quality while still being small enough for the consequent stages to model and to not consume excessive GPU memory.

(2) Latent Encoder: as mentioned, having larger voxels gives better reconstruction, but fitting a generative model becomes more challenging. Therefore, our latent auto-encoder compresses voxels into smaller latents, and in Tab. 13, we report how downsampling factors (for the coarse 3D latent) in the encoder affect the voxel reconstruction quality. We found that $ds = 4$ gives a good compromise between having a low reconstruction loss and a latent size small enough to fit a diffusion model.

(3) Explicit Density: in Fig. 15, we show that having explicit feature and density grids outperforms implicitly inferring density from the voxel features with an MLP. Our encoder explicitly predicts the occupancy of each frustum entry before merging frustums across multiple views and thus prevents incorrect feature mixing due to occlusions that can happen if frustums are merged with naive mean-pooling without accounting for occupancy. Implicit depth prediction similar to Lift-Splat [52] can also account for occlusion but this requires an additional density prediction step for volume rendering which we avoid by predicting densities directly from each view.

(4) Sampling Steps: sampling with a larger number of steps only marginally improved FID - 50/37.18, 125/36.74, 250/35.69 (# steps/FID with DDIM sampler $\eta = 1.0$).

B.2. Generated Scenes

We provide additional generated samples on AVD in Figures 16 and 17. For Carla, we provide samples in Figure 18 and 19. Figure 20 contains visualizations of 3D meshes obtained by running marching-cubes [43] on samples.

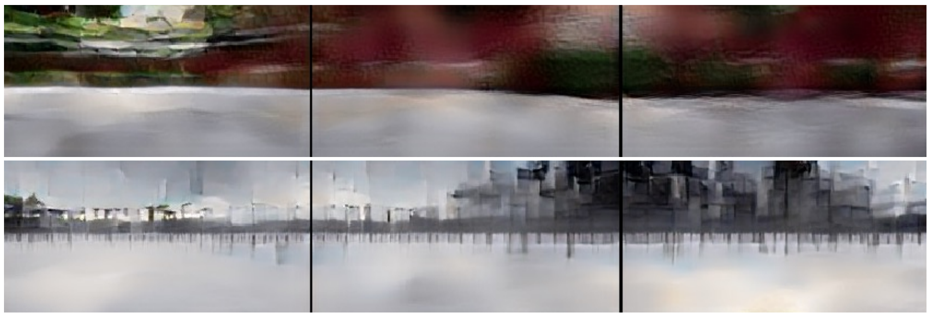

Figure 14. **Directly fitting a diffusion model without compression with latent auto-encoder is challenging.** Each row is a sample from a diffusion model trained directly on the $128 \times 128 \times 32$ grids from the first stage autoencoder.

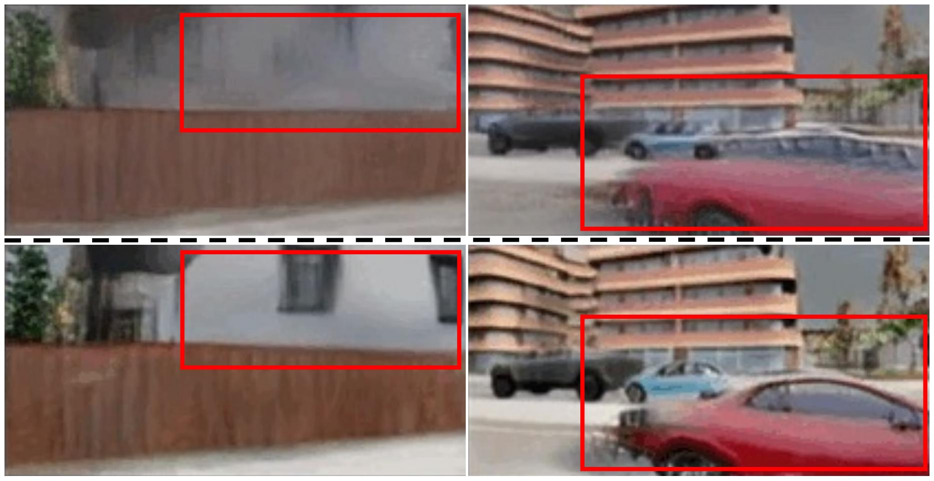

Figure 15. Renderings from the scene autoencoder. *Top row*: without explicit density & feature grids, *Bottom row*: the full model.

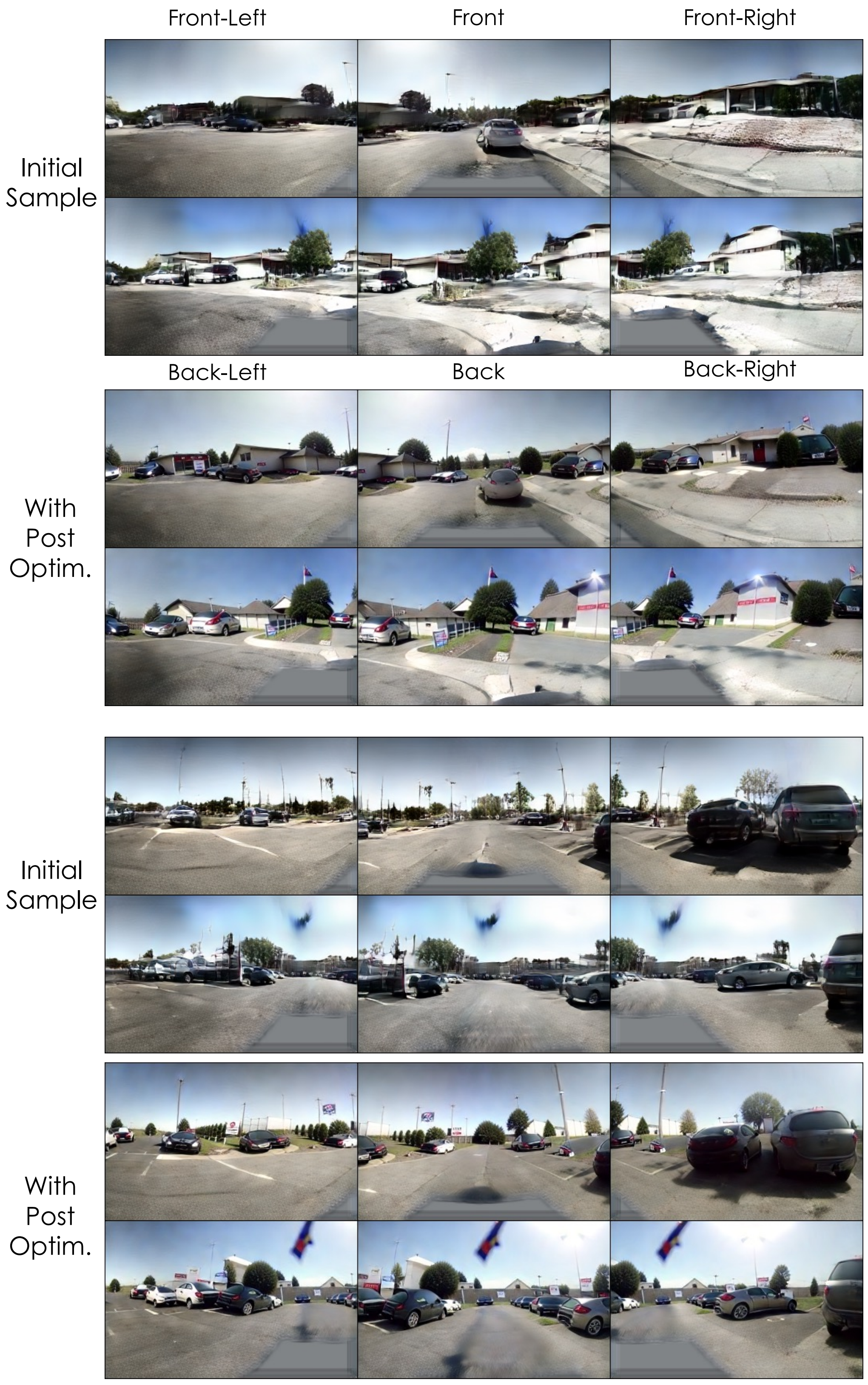

Figure 16. Additional generated samples on AVD. Each initial sample is further improved with post-optimization (Section A.4).

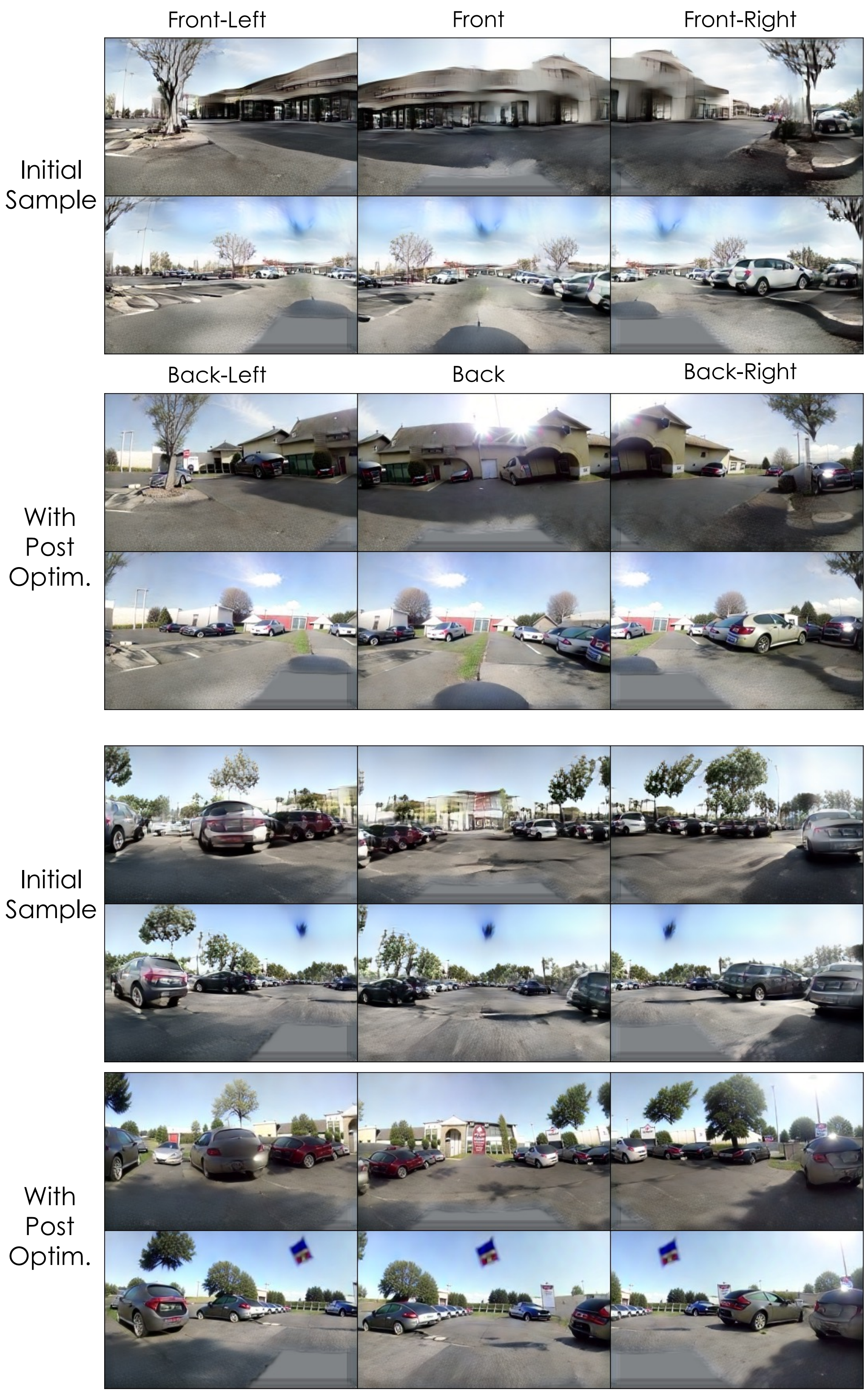

Figure 17. Additional generated samples on AVD. Each initial sample is further improved with post-optimization (Section A.4).

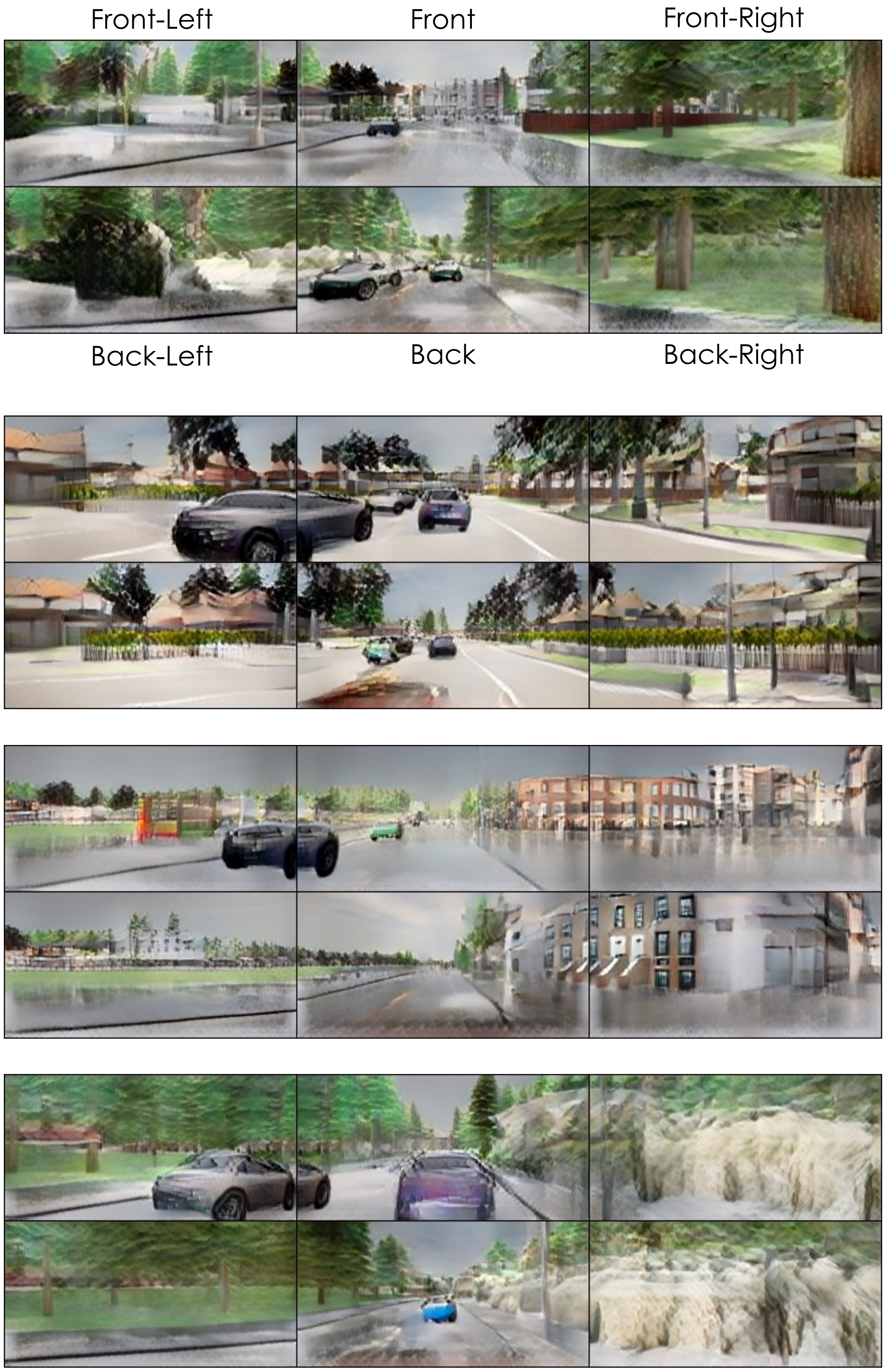

Figure 18. Additional generated samples on Carla.

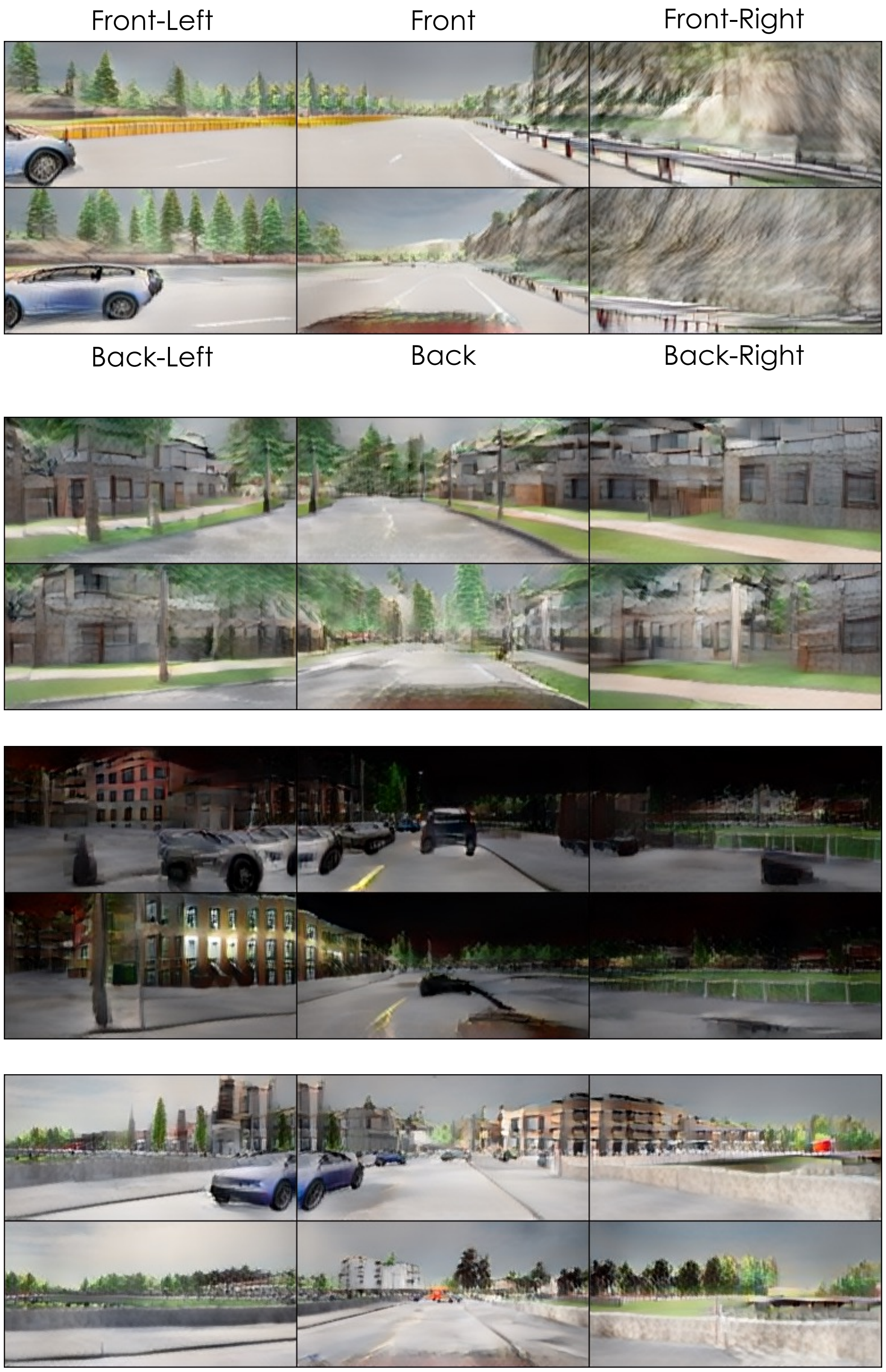

Figure 19. Additional generated samples on Carla.

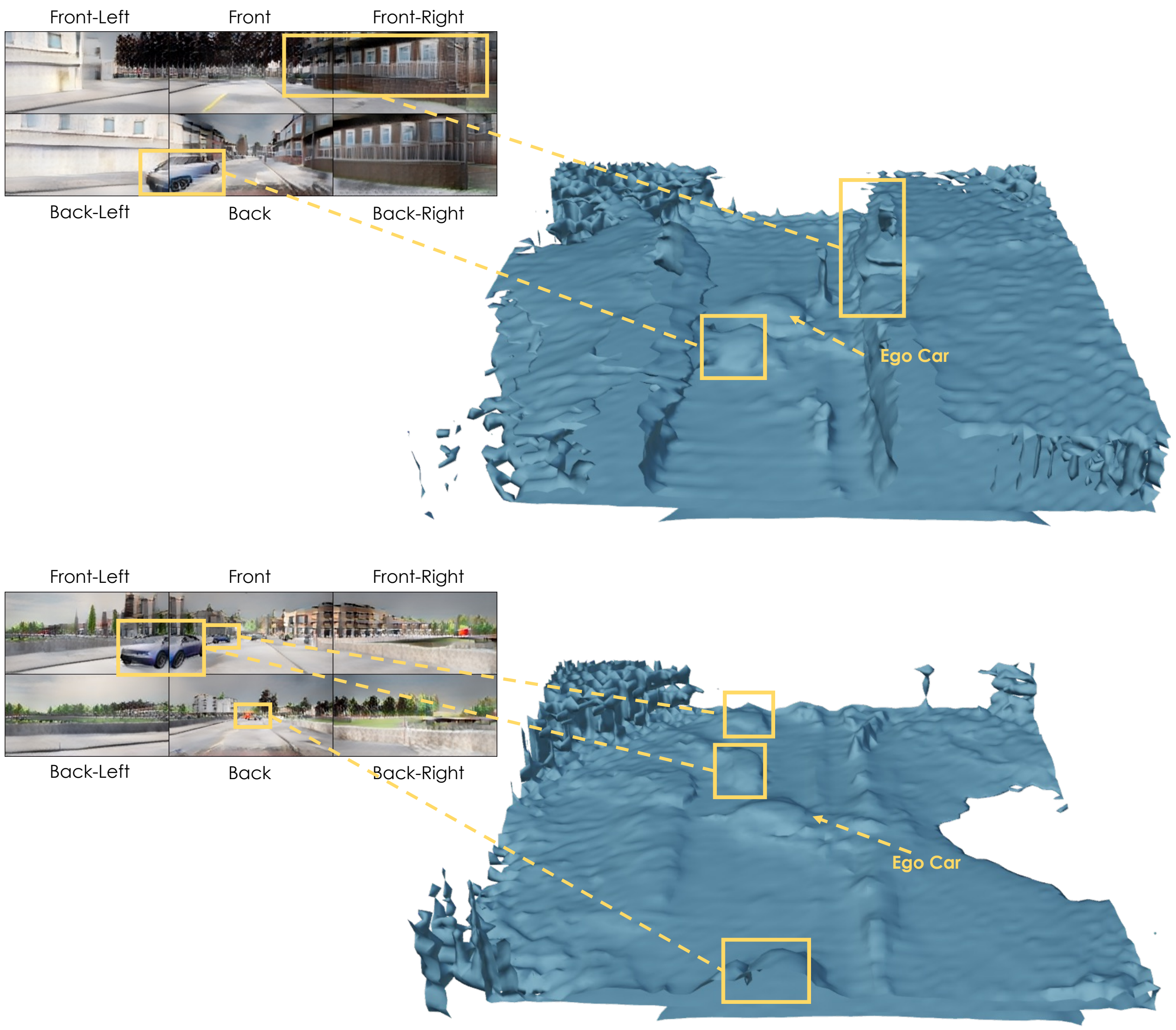

Figure 20. We run marching-cubes [43] on the density voxels to visualize the geometry of the samples generated by NF-LDM.

B.3. Stylization using Score Distillation Sampling (SDS) loss

In addition to using SDS loss to post-optimize our voxels for quality, we can also use it to modify the style of a given scene. Given a desired target style (*e.g.* a medieval castle), we first generate a dataset of target (positive) and source (negative) images using one of two methods:

- **Image translation**: We use stable diffusion [60] for text-guided image-to-image translation as introduced by SDEdit [47]. Specifically, we autoencode scenes from our dataset to contruct a set of reconstructed images which we use as the source images. We then run the image to image translation on the source's matching dataset images, using a strength of 0.4 and guidance scale of 10, using the text of the target style to get target images. We repeat this for 500 images and take the average of the source images' CLIP embeddings and target images' CLIP embeddings as y' and y used in negative guidance respectively.

- **Scraping**: We use the same negative conditioning y' as we do for quality post-optimization. For, y, we search and download 100 images from the internet with the target query, manually filter these images for relevance and take the average CLIP embedding.

We run SDS optimization with these modified conditioning vectors using the same procedure outlined in Section A.4. The stylization results can be seen in Figure 21-23. Moreover, as our neural fields are represented as voxel grids, we can easily combine different neural fields. In Figure 24-26, we combine two sampled voxels by replacing the center region $(32 \times 80 \times 80)$ of one voxel with the center region of the other one. We qualitatively show the importance of having our initial voxel samples and the effect of negative guidance in Figure 27.

We note that the stylized scenes match the target style well, but do not perfectly preserve the content of the original scene (*e.g.* the cars). For the scraping method, images for conditioning are randomly chosen and do not necessarily contain street scenes which could result in these semantic changes. For the image translation method, we empirically found that parts of the translated scene with worse content preservation appeared differently when doing stylization with SDEdit multiple times on a single rendered image (*e.g.* for lego stylization, cars contain different brick details and colors in each translation). We hypothesize that doing SDS loss with this conditioning for thousands of iterations encourages the optimization to satisfy these multiple possible translations which results in blurring and a lack of content preservation in these regions. Performing the post-optimization jointly with a reconstruction loss on images that preserves content and have the desired style (e.g. obtained through the same img2img translation) could improve content preservation.

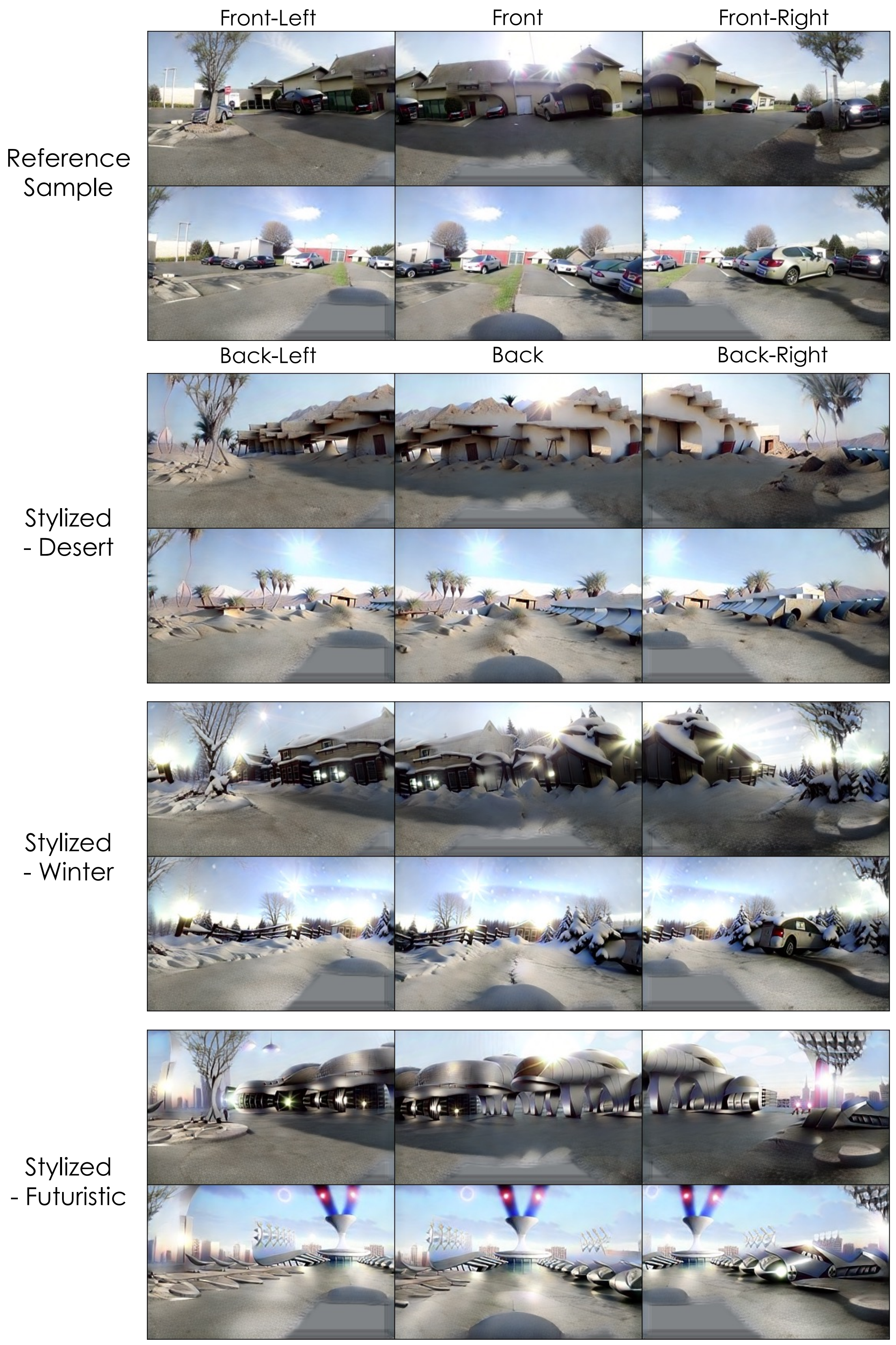

Figure 21. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

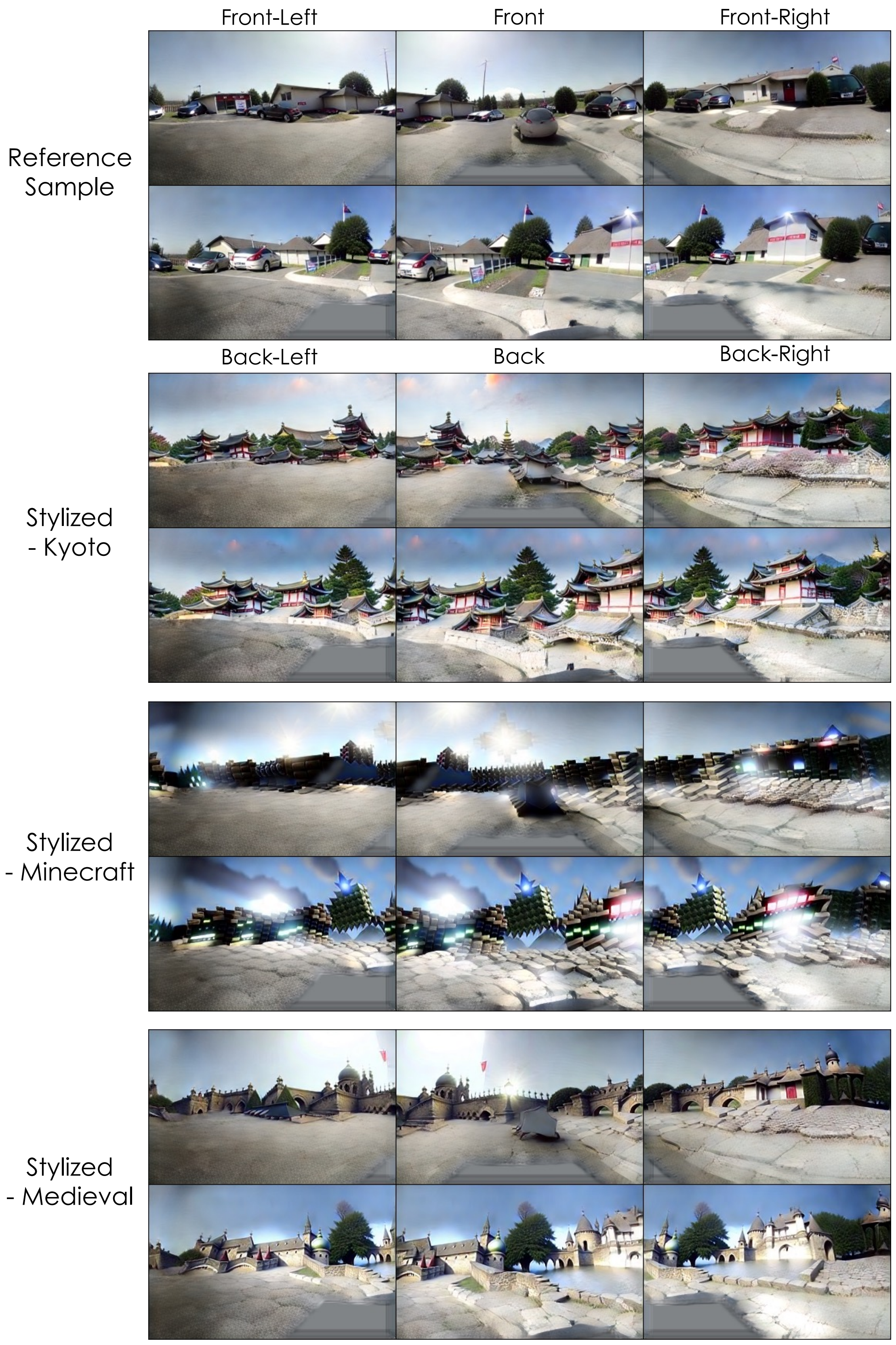

Figure 22. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

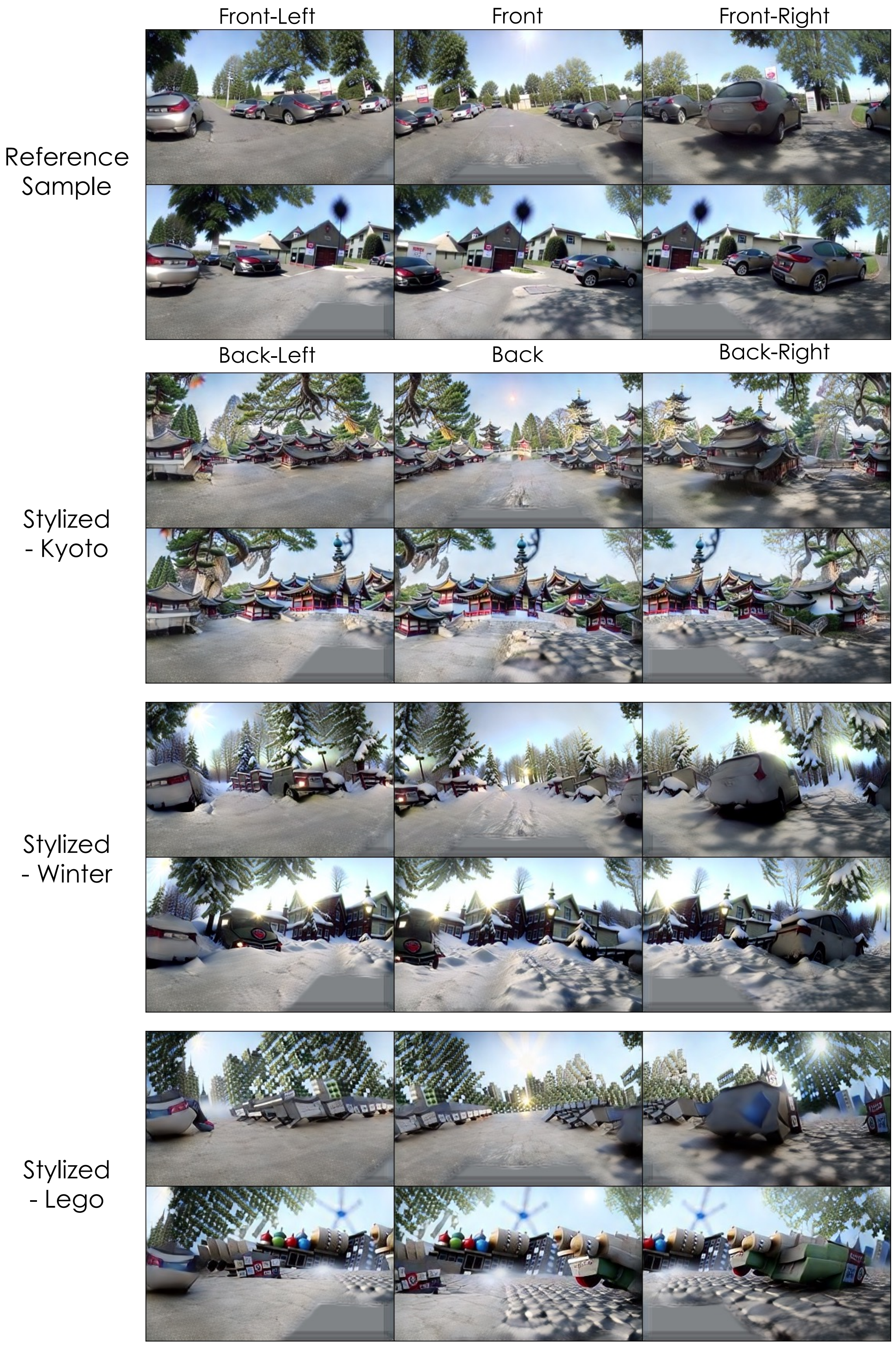

Figure 23. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

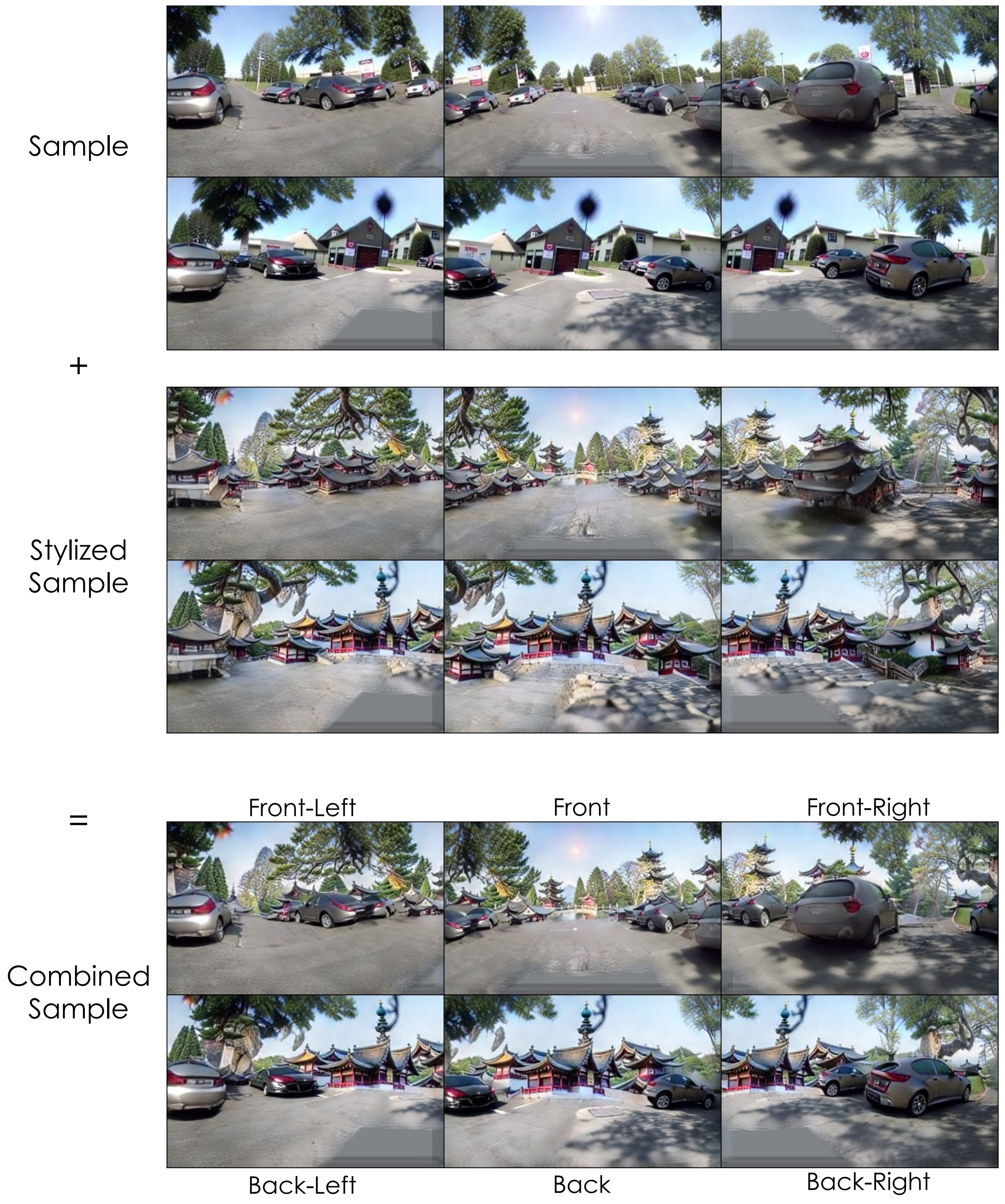

Figure 24. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

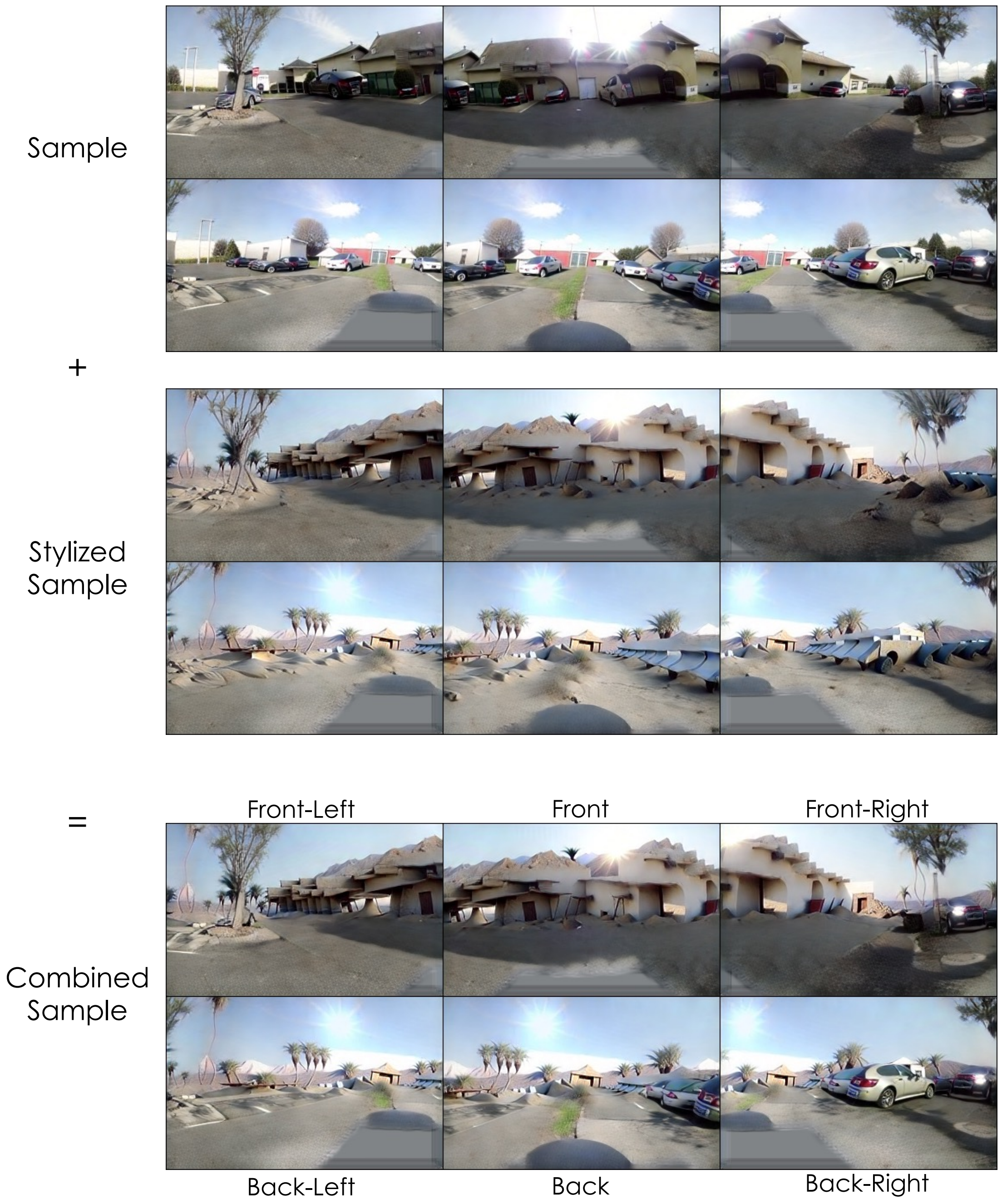

Figure 25. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

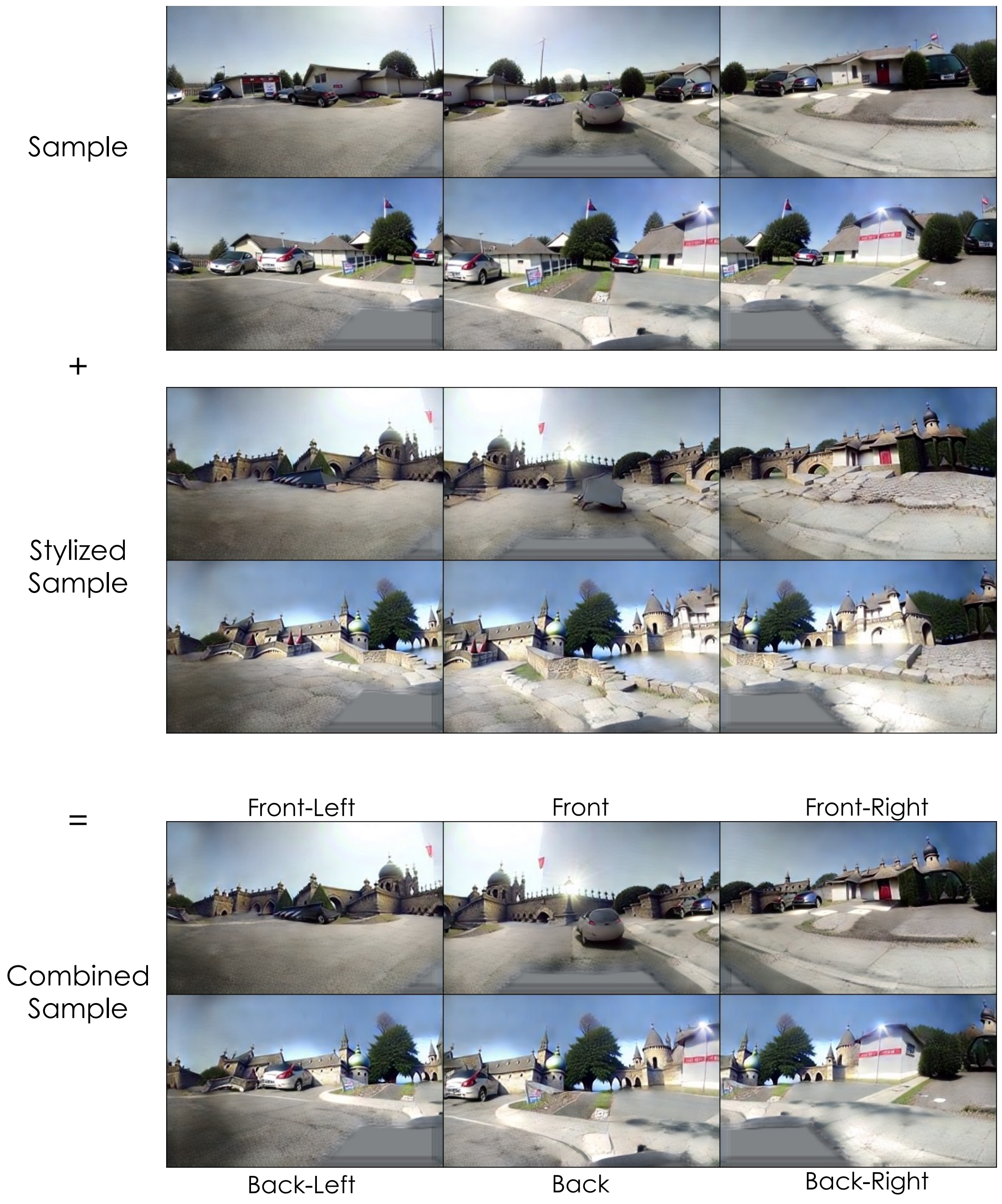

Figure 26. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

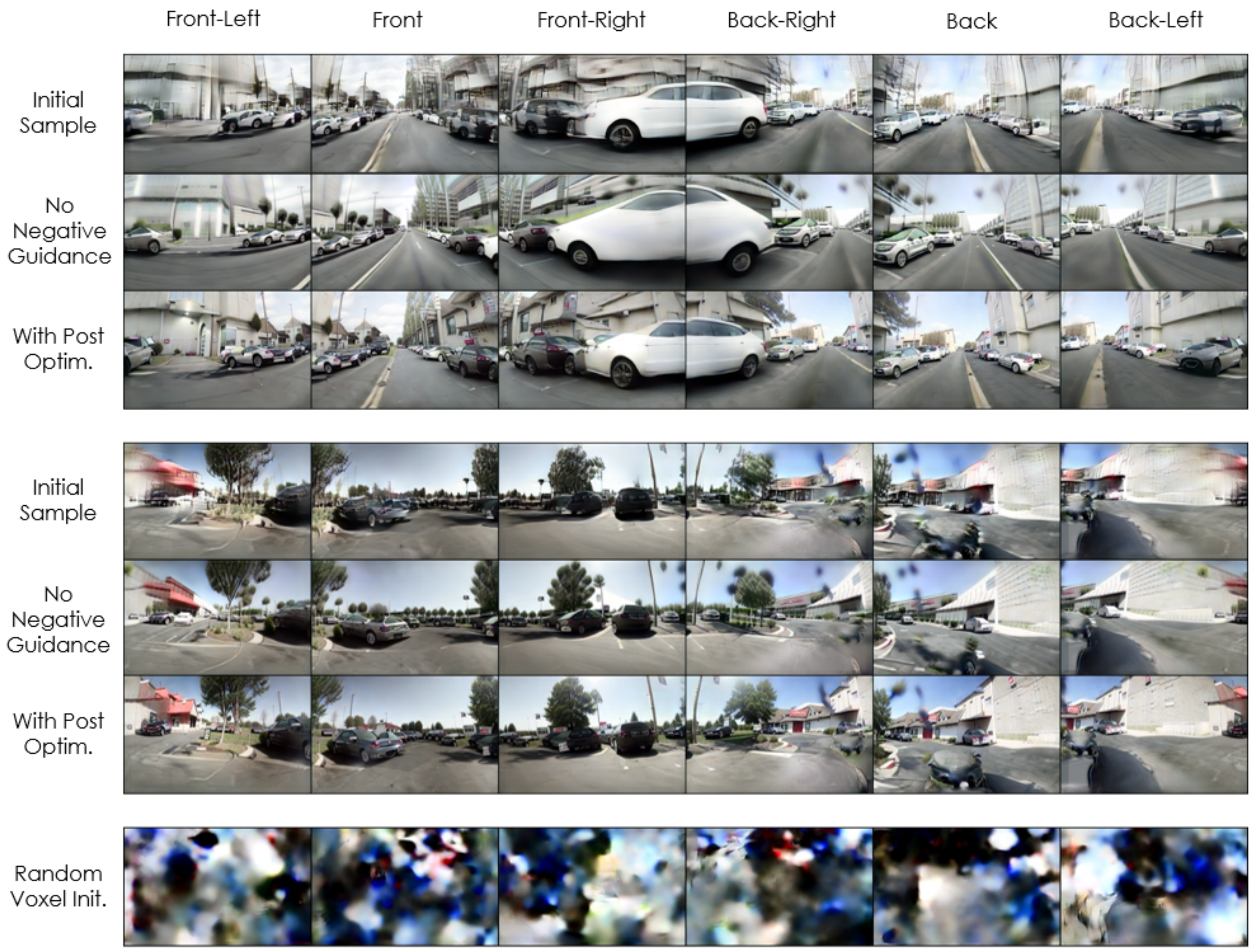

Figure 27. Ablating the post-optimization method. We show initial samples, samples optimized with classifier-free guidance and samples optimized with negative guidance for two scenes. Additionally, we show the result of post-optimization using a random gaussian intialization for the voxels.

B.4. Bird's Eye View Conditioned Synthesis

We provide additional Bird's Eye View conditioned synthesis results in Figure 28.

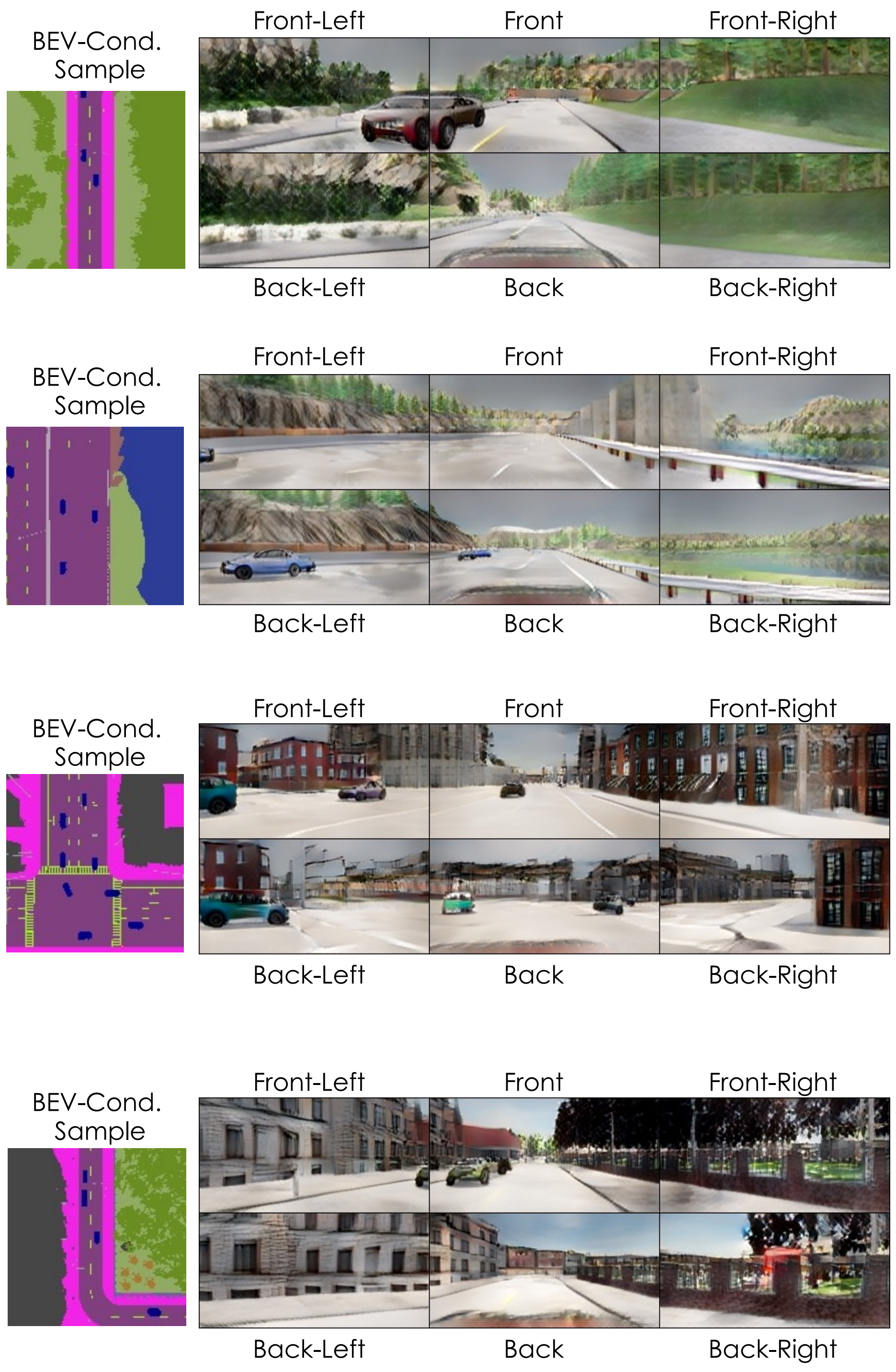

Figure 28. Additional results on Birds' Eye View Conditioned generation. In the BEV segmentation map, colors denote different region types: green - trees and vegetations, blue - water, grey - buildings, purple - road, pink - sidewalk, dark blue - vehicles (note that the ego car is at the center and thus not visualized).

B.5. Scene Editing

We provide additional scene editing results in Figure 29 and 30.

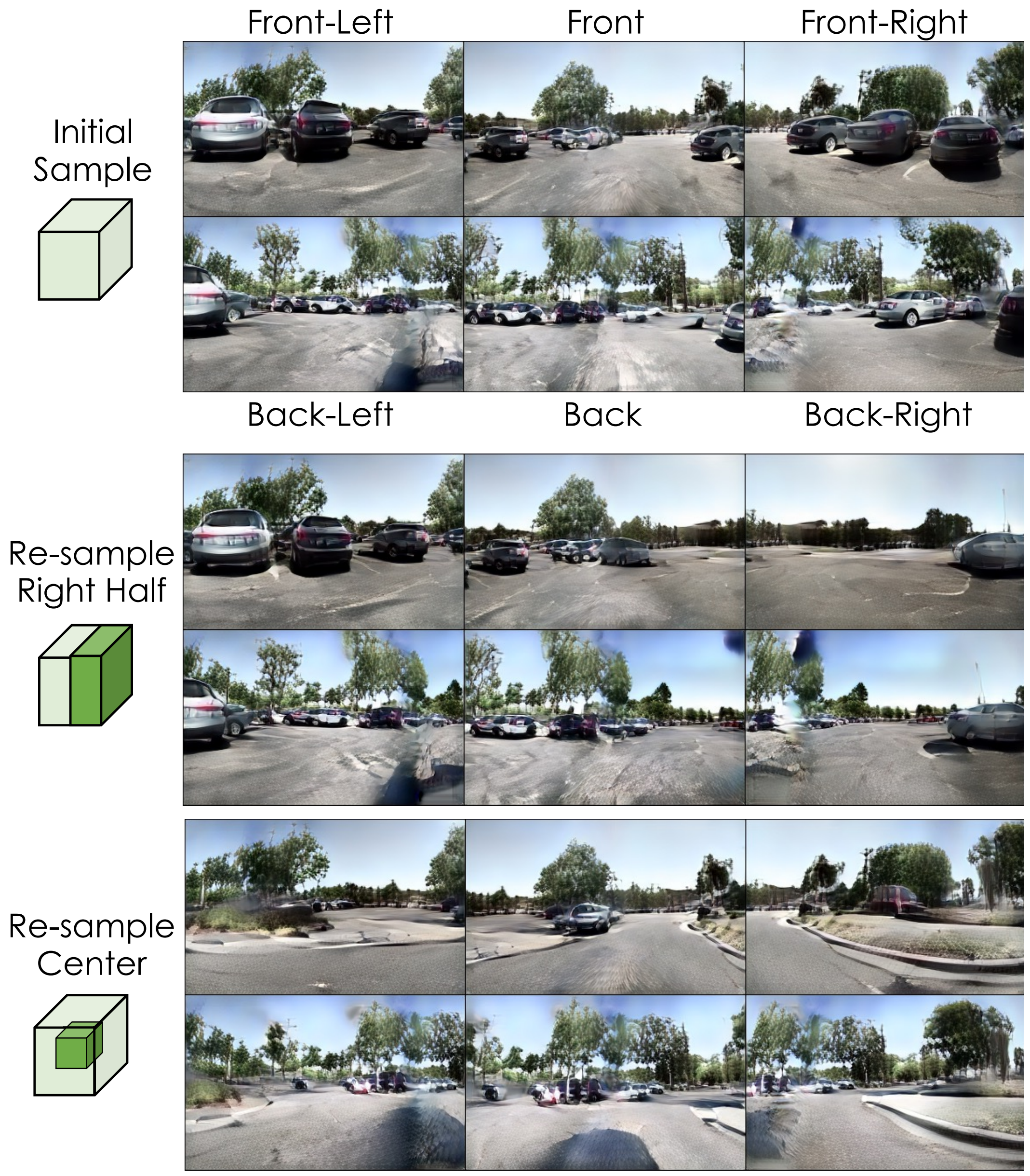

Figure 29. Additional results on scene editing by re-sampling. Given an initial sample, we edit the specified regions by re-sampling them with our model.

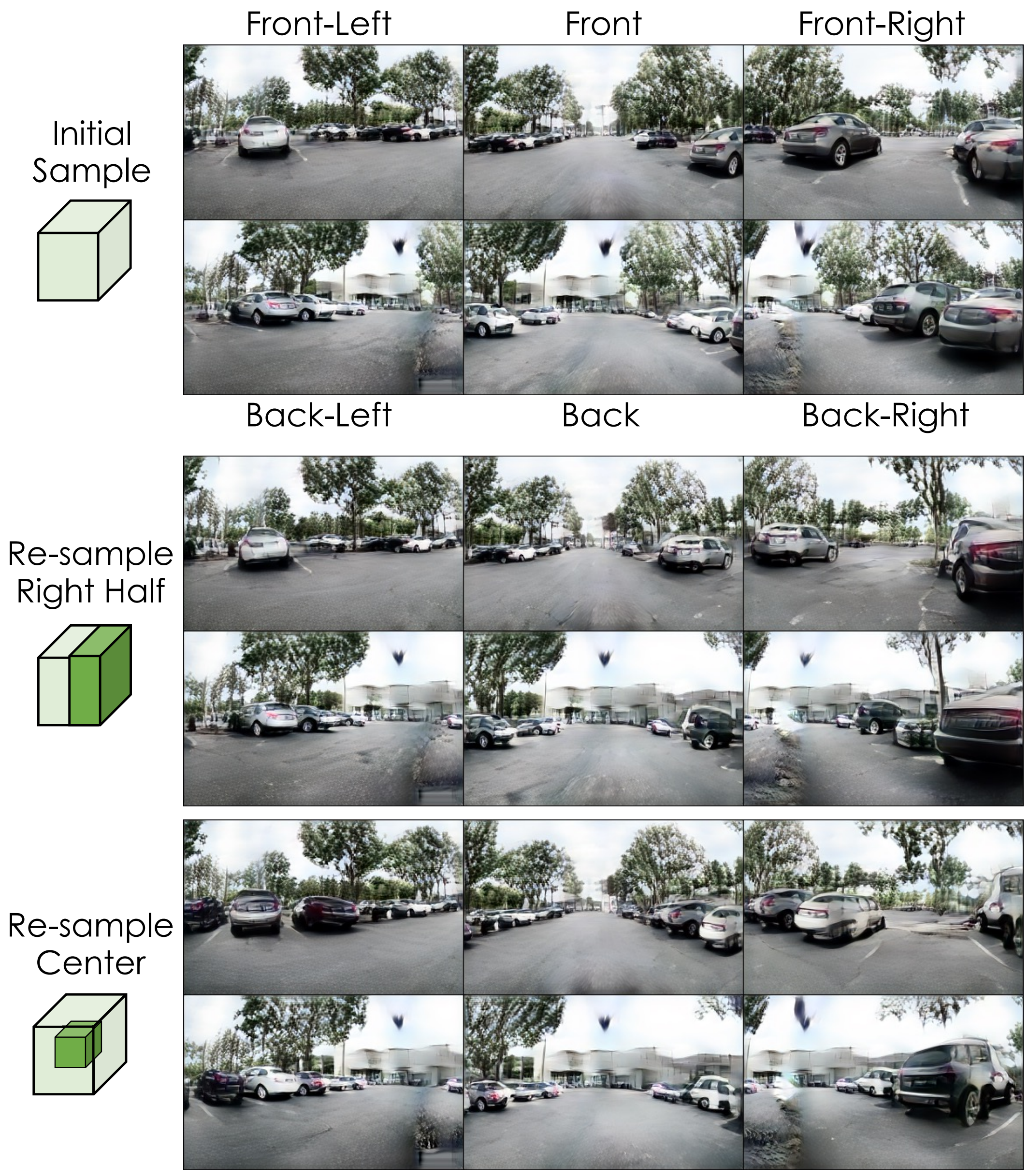

Figure 30. Additional results on scene editing by re-sampling. Given an initial sample, we edit the specified regions by re-sampling them with our model.

B.6. Text-Guided Style Transfer

In addition to stylization using SDS loss, which is effective for high-quality large structural modifications, in Figure 31, we show results of applying CLIP directional loss [16] to finetune our decoder for quick global style changes that generalize across scenes. The target domain is expressed in natural language (*e.g.* sketch of a city) and the source domain is either "photo" or "photo of a city". We first obtain the update direction in CLIP space, $u_t = e_{target} - e_{source}$, where e_{target} and e_{source} are the CLIP text embeddings of the source and domain respectively. Then, we sample an encoded or sampled voxel, V, and initialize a frozen and trainable copy of our scene-autoencoder's decoder D_f and D_t respectively. Additionally, we sample a set of camera paramaters $\{\kappa\}_{1...N}$ from our dataset as our base poses.

At every iteration, we uniformly sample a translation offset in both the forwards and sideways directions between -1 and 1 metres which we apply to the base poses to obtain jittered camera poses $\{\hat{\kappa}\}_{1...N}$. We render out images with the jittered poses using both the frozen and trainable decoders, obtaining $\hat{i}_f$ and $\hat{i}_t$ respectively. We then obtain the current decoder's image update direction as $u_i = \hat{e}_{target} - \hat{e}_{source}$ where e_{target} and e_{source} are the CLIP image embeddings of $\hat{i}_f$ and $\hat{i}_t$ respectively. The loss is then 1 minus the cosine similarity of u_i and u_t, which is used to update only D_t.

We train using the Adam optimizer with learning rate set to 0.002 and betas of $(0.9, 0.999)$ between $20 - 100$ iterations, taking around a minute on a single V100 GPU. We empirically found that while finetuning with CLIP directional loss is fast and training a domain-adapted model only requires optimizing on a single scene, SDS based stylization (Sections B.3) produces much higher quality results.

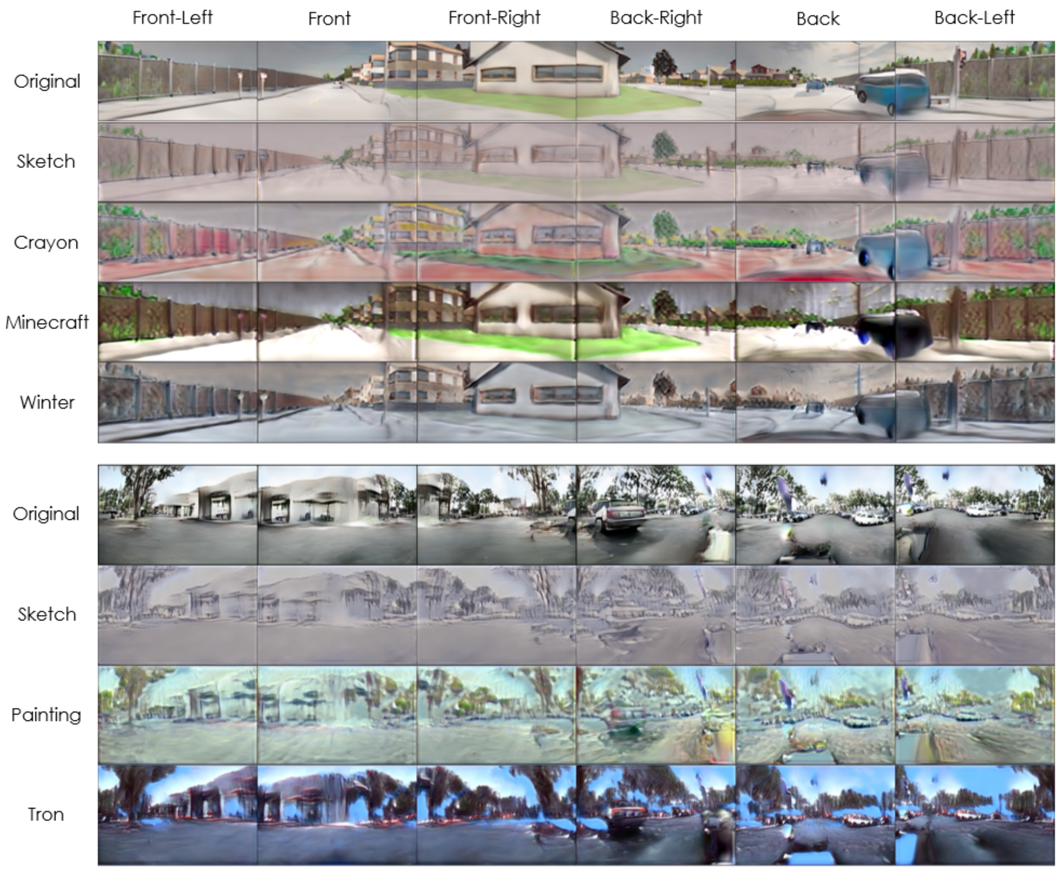

Figure 31. Text-guided style transfer results on an encoded Carla scene and a sampled AVD scene. Each result was obtained using a decoder finetuned by running CLIP directional loss with the specified style on a separate scene.