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

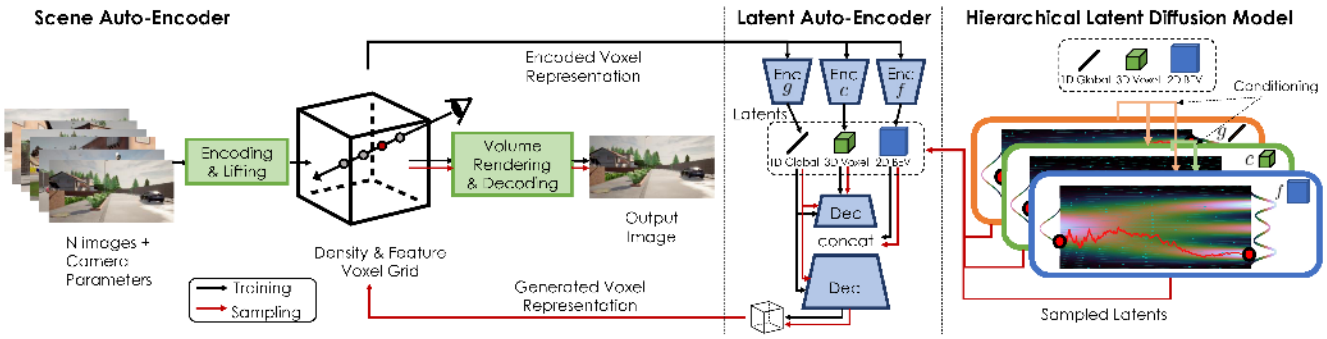

Figure 1. **Overview of NeuralField-LDM.** We put the model overview from the main text for reference. We first encode RGB images with camera poses into a neural field represented by density and feature voxel grids. We compress the neural field into smaller latent spaces and fit a hierarchical latent 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 1 includes additional model architecture and training details.
- Section 2 includes additional qualitative results.
- We also include a supplementary video which has better visualizations for evaluating the 3D aspect of our model.

1. 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 [29] 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.

1.1. 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 [24] encoder, replacing Batchnorm layers [8] with Instance normalization [25]. 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 [1] 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 [9] 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 [9], 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 [9] discriminator along with an R1 gradient regularization [15]. 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 [2]. 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.

1.2. 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. 1. 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 [4]. Tables 1-3 contain the descriptions of the encoder architectures. Resblocks [6] contain two convolution layers and each conv layer has a group normalization [28] and a SiLU activation [3] prior to it. AttnBlocks are implemented as self-attention modules [27] and MidBlocks represent a block of {ResBlock, AttnBlock, ResBlock}. We add latent regularizations to avoid high variance latent spaces [18]. For the 1D vector g, we use a small KL-penalty via the reparameterization trick [11], and for c and f, we impose a vector-quantization [4, 26] 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 [19] 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.

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 1. 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 2. 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 3. 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 4. Decoder of the latent auto-encoder

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 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 1.1. 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 [4, 26] for c and f. The image reconstruction loss is multiplied by 10. We train the LAE with the Adam optimizer [10] with a learning rate of 0.0002.

1.3. Hierarchical Latent Diffusion Models

Background on Denoising Diffusion Models Denoising Diffusion Models [7, 21, 23] (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 the diffused input by reducing the following loss

$$\mathbb{E}_{x\sim q(x_0), t\sim p_t, \epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|y - \psi(x_t; t)\|_2^2\right], \tag{1}$$

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 [20] 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 [7] . We use the *variance-preserving* noise schedule [23], 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:

$$\begin{aligned} h &= linear(x) \\ h_{emb} &= linear(t_{emb}) \\ h &= h + h_{emb} \\ h &= linear(h) \\ return\ &linear(x) + h \end{aligned} \tag{2}$$

Here, x is the input to the block and t_{emb} is the timestep embedding for the diffusion time step t. We follow [18] 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 2.

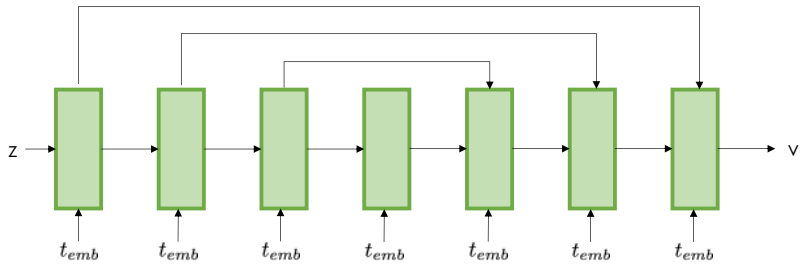

Figure 2. Architecture diagram of ψ_g. z is the input to the network and v is the output. The green blocks are the linear blocks with skip connections (Eq. 2). The model is a 1D analogous version of the 2D Unet commonly used in 2D diffusion models.

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 5 contains the hyperparameter choices for ψ_g.

Coarse and Fine Latent Diffusion Model. ψ_c and ψ_f adopt the U-net architecture [19] and closely follow the 2D Unet architecture used in [18]. 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

	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 5. 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.

	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 6. 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 [18] 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 7. 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 [18] 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.

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 6 and Table 7 contain the hyperparameter settings for ψ_c and ψ_f, respectively.

The cross attention layers in [18] 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 [18] for the choice of diffusion steps (1000), noise schedule (linear), and optimizer (AdamW [13]) for all experiments. For sampling, we use the DDIM sampler [22] with 250 steps.

1.4. 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 5 and 6. For the entire optimization, we use a fixed set of camera parameters $\{\kappa\}_{1...N}$ 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 $\{\kappa\}_{1...N}$ to obtain $\{\hat{\kappa}\}_{1...N}$, 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 [18], finetuned to condition on CLIP image embeddings [17]. We train with negative guidance, as detailed in Section 1.4.1. 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 16. 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 16, having a voxel initialization sampled or encoded from our model is critical to the success of post-optimization.

1.4.1 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 8. 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 9. Ablation of the downsampling factors (ds) of the latent autoencoder. We report the validation voxel reconstruction loss.

2. Additional Results

2.1. 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. 3, 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. 8 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. 9, 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. 4, 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 [16] 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 $\eta = 1.0$).

2.2. Generated Scenes

We provide additional generated samples on AVD in Figures 5 and 6. For Carla, we provide samples in Figure 7 and 8. Figure 9 contains visualizations of 3D meshes obtained by running marching-cubes [12] on samples.

Figure 3. **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.

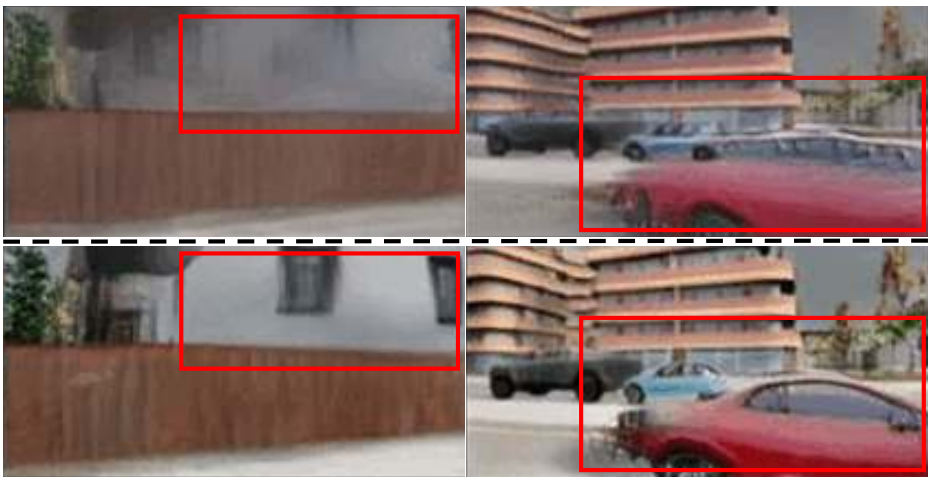

Figure 4. Renderings from the scene autoencoder. *Top row*: without explicit density & feature grids, *Bottom row*: the full model.

Figure 5. Additional generated samples on AVD. Each initial sample is further improved with post-optimization (Section 1.4).

Figure 6. Additional generated samples on AVD. Each initial sample is further improved with post-optimization (Section 1.4).

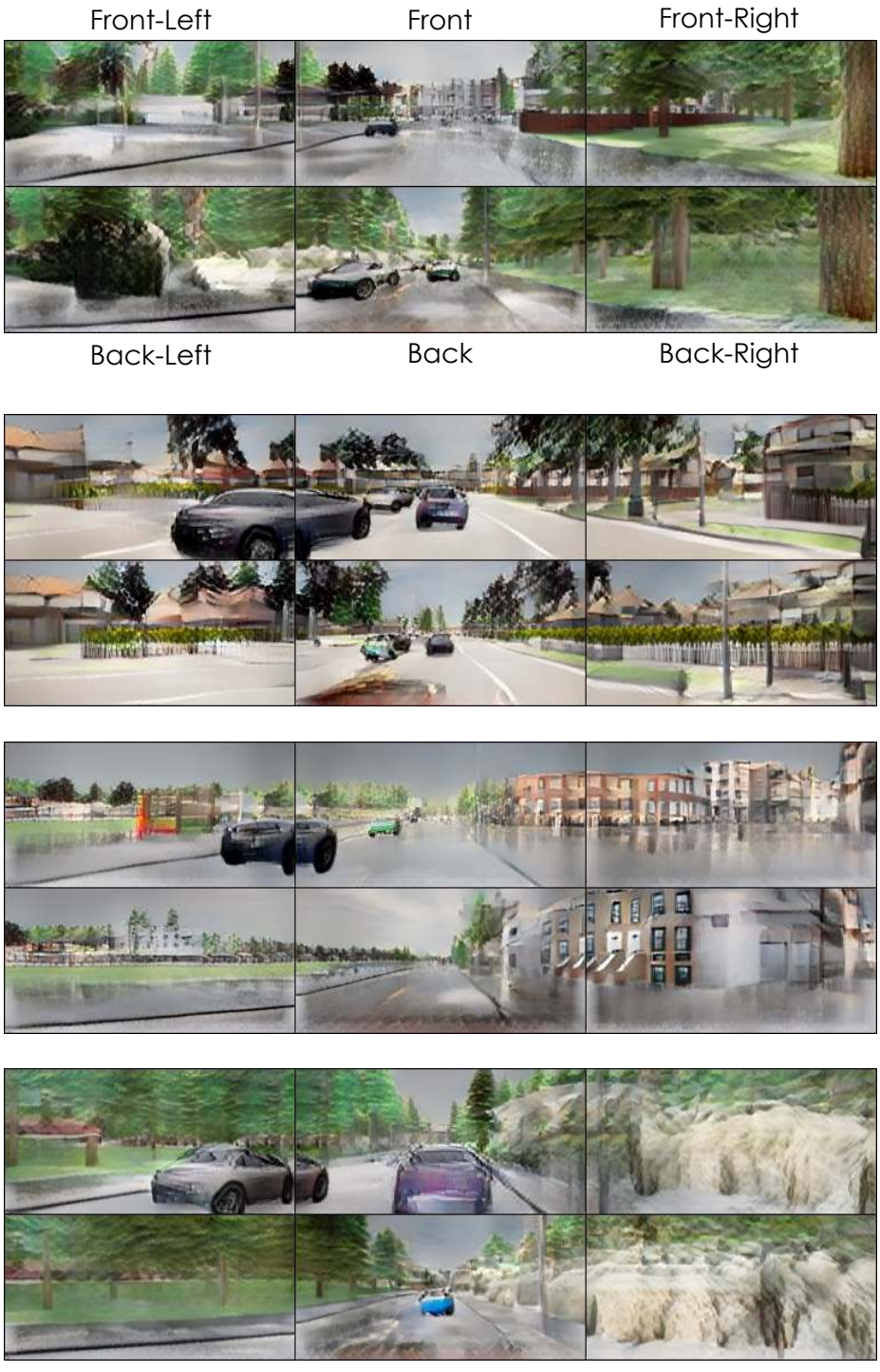

Figure 7. Additional generated samples on Carla.

Figure 8. Additional generated samples on Carla.

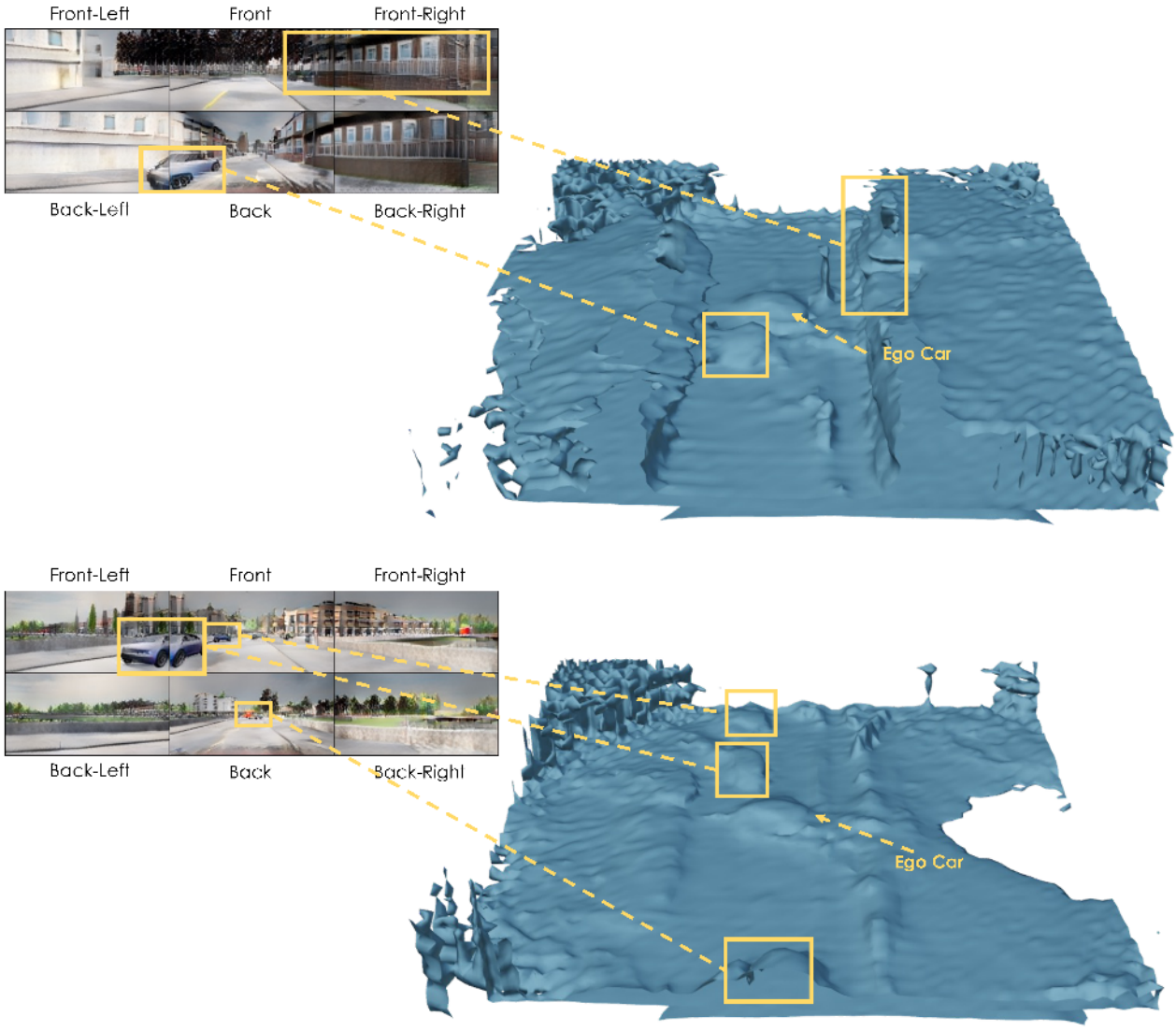

Figure 9. We run marching-cubes [12] on the density voxels to visualize the geometry of the samples generated by NF-LDM.

2.3. 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 [18] for text-guided image-to-image translation as introduced by SDEdit [14]. 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 1.4. The stylization results can be seen in Figure 10-12. Moreover, as our neural fields are represented as voxel grids, we can easily combine different neural fields. In Figure 13-15, 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 16.

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.

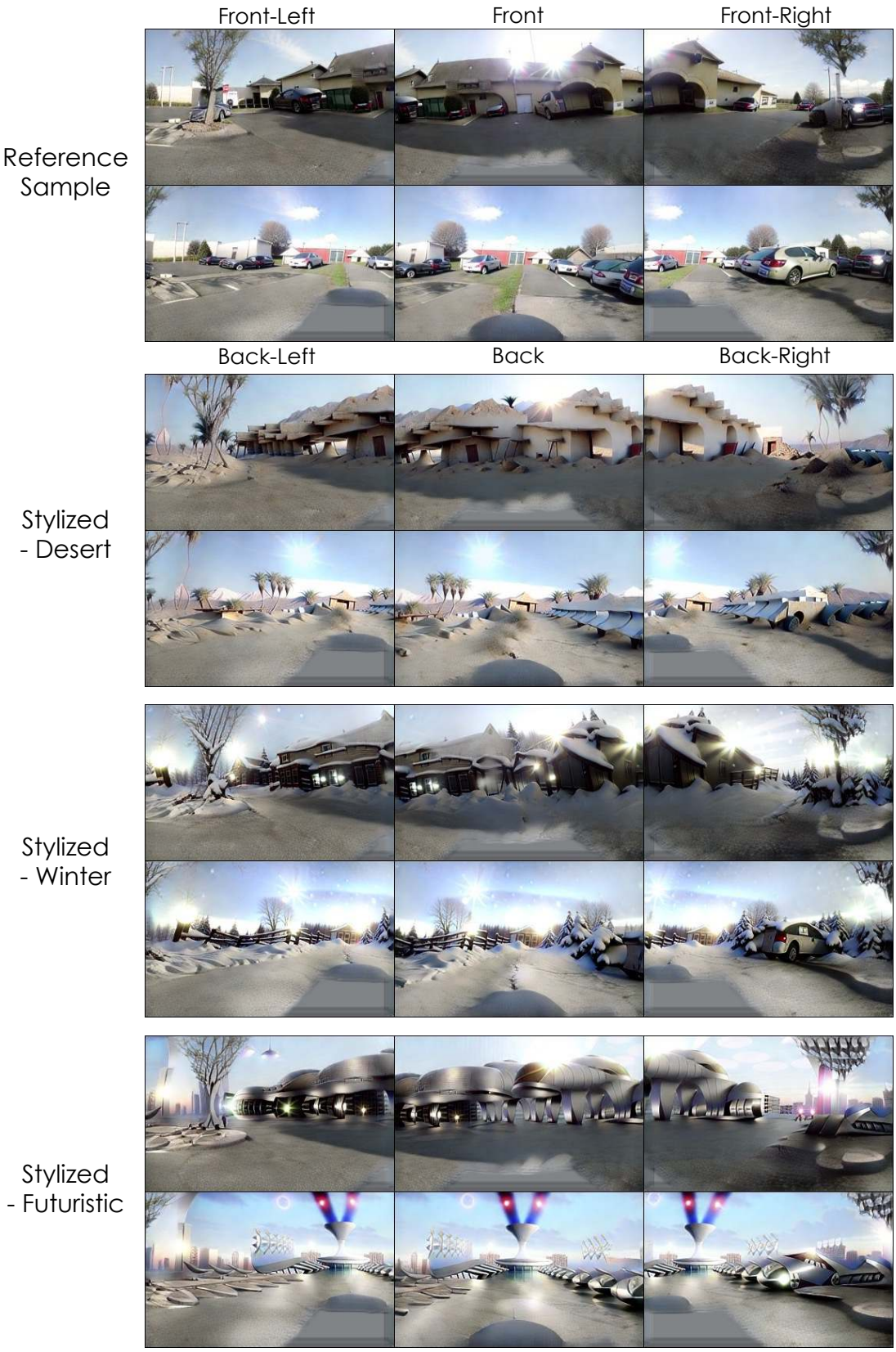

Figure 10. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

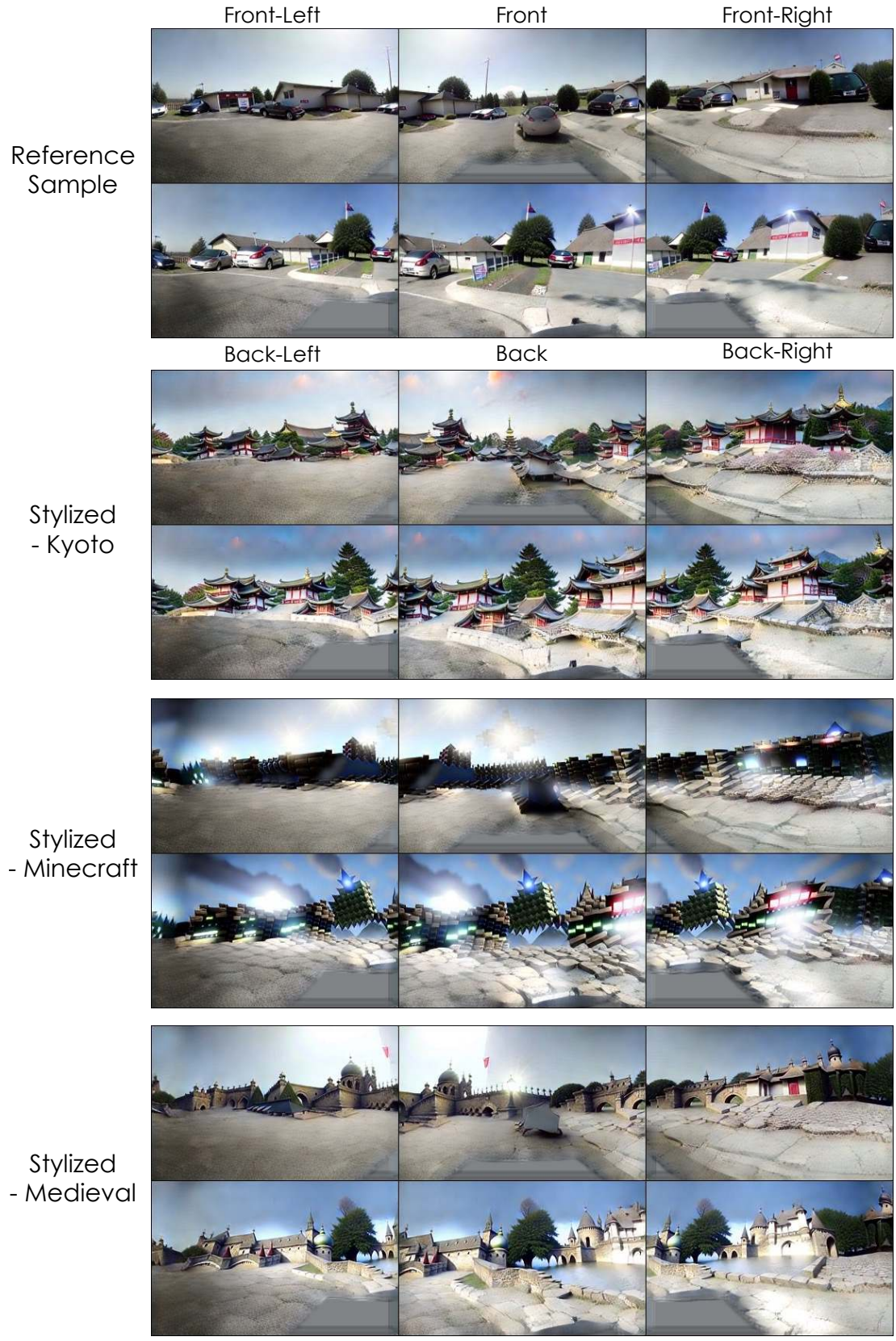

Figure 11. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

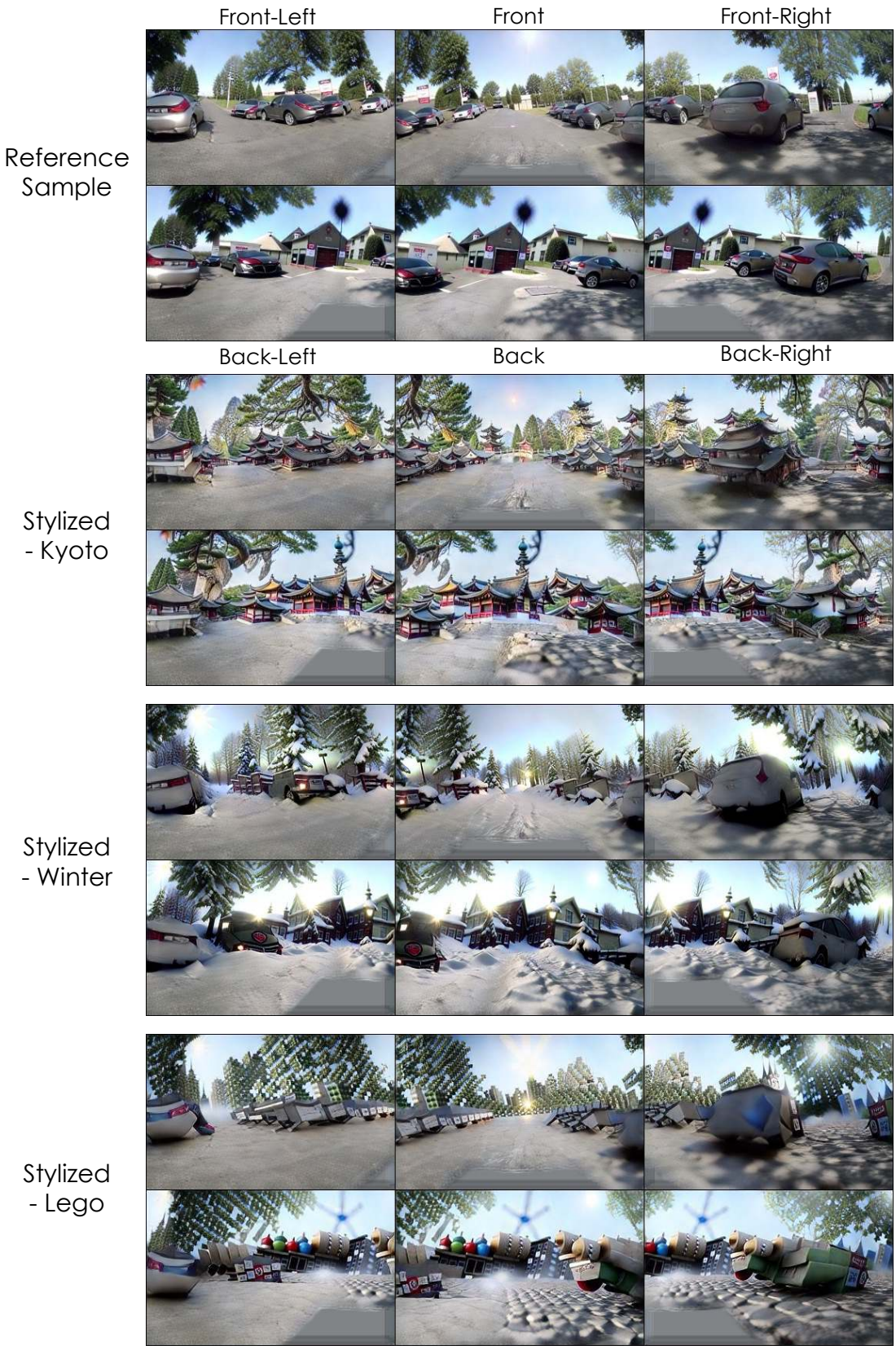

Figure 12. Additional stylized samples. All stylized samples start the post-optimization step from the same initial sample.

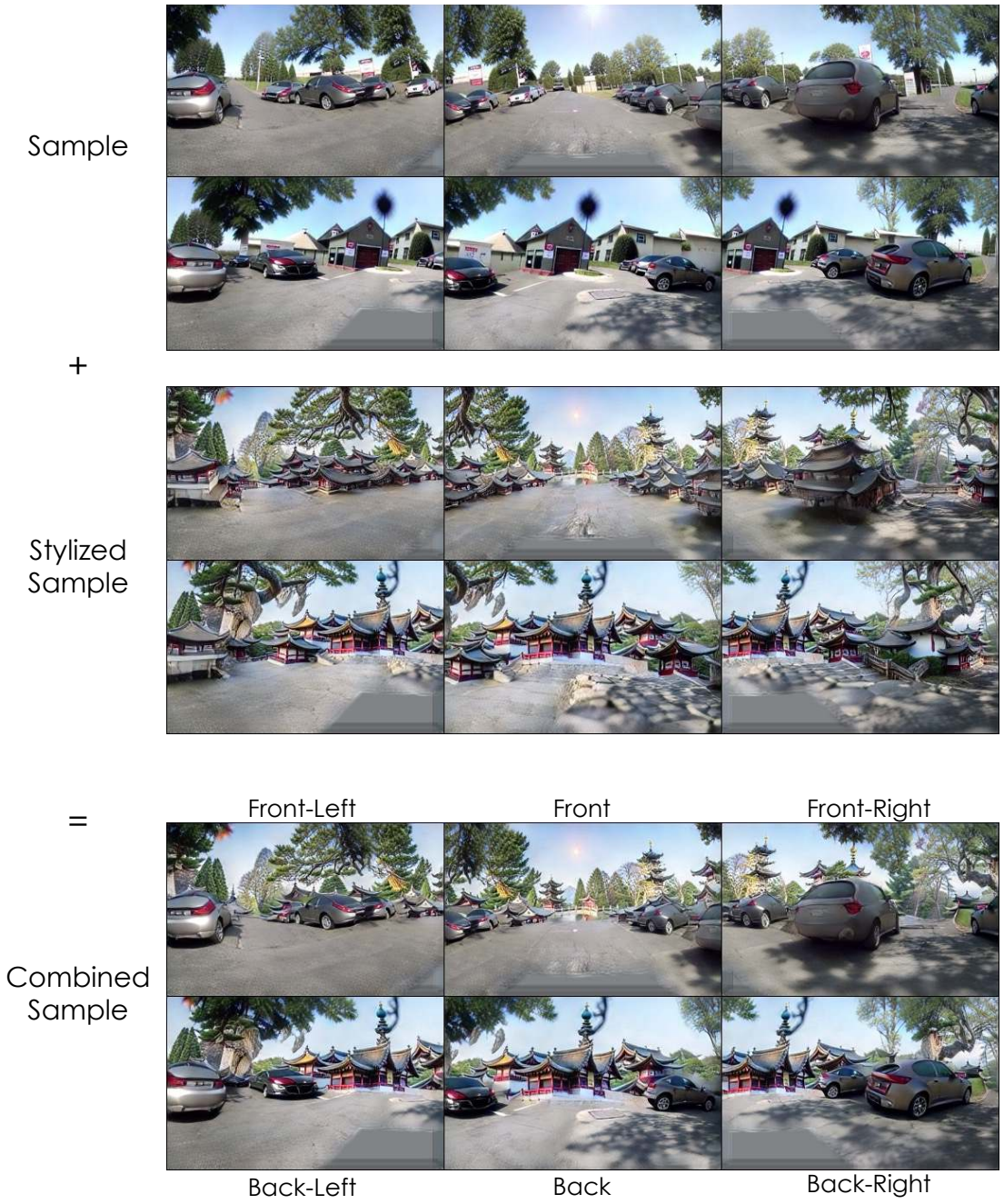

Figure 13. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

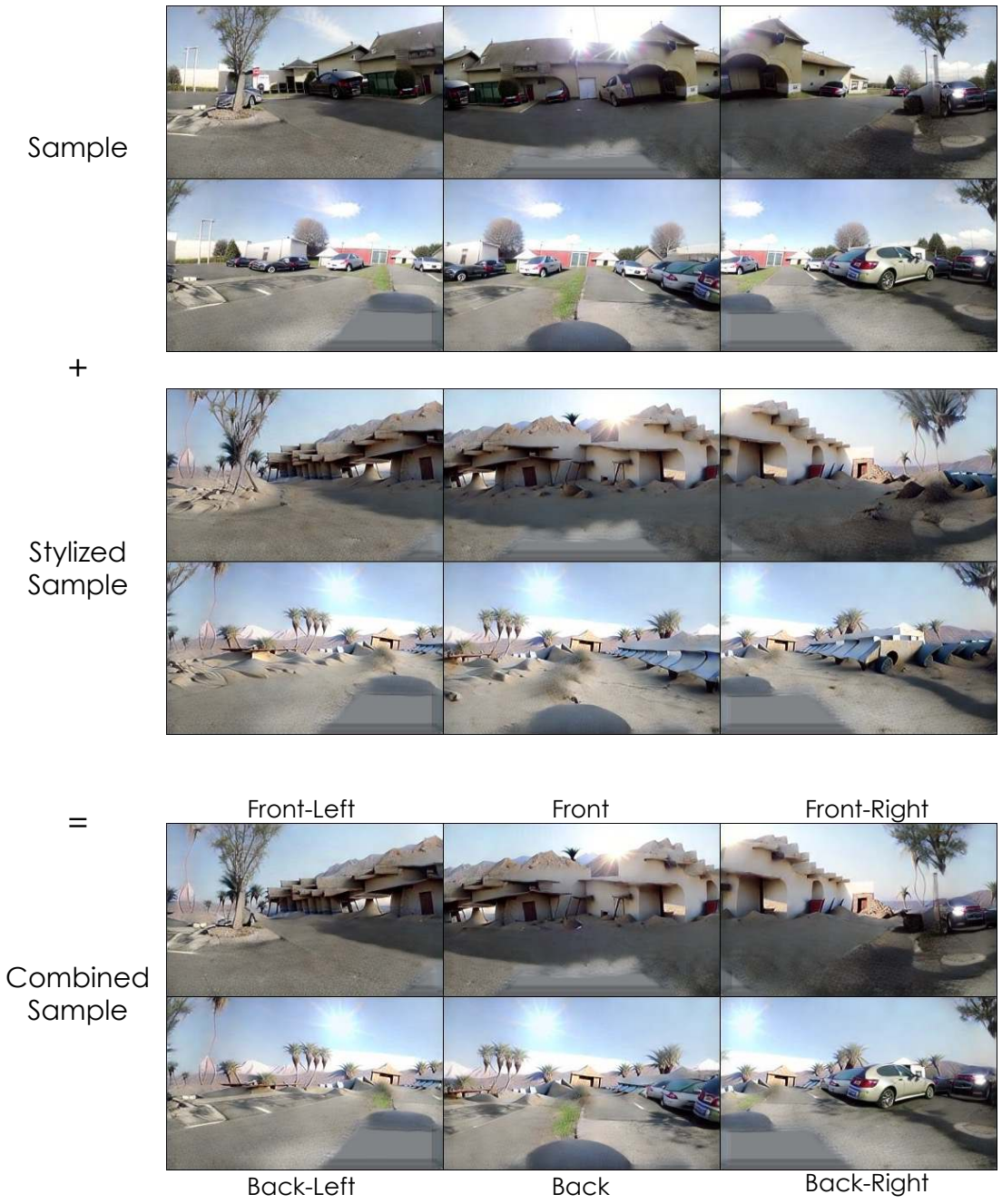

Figure 14. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

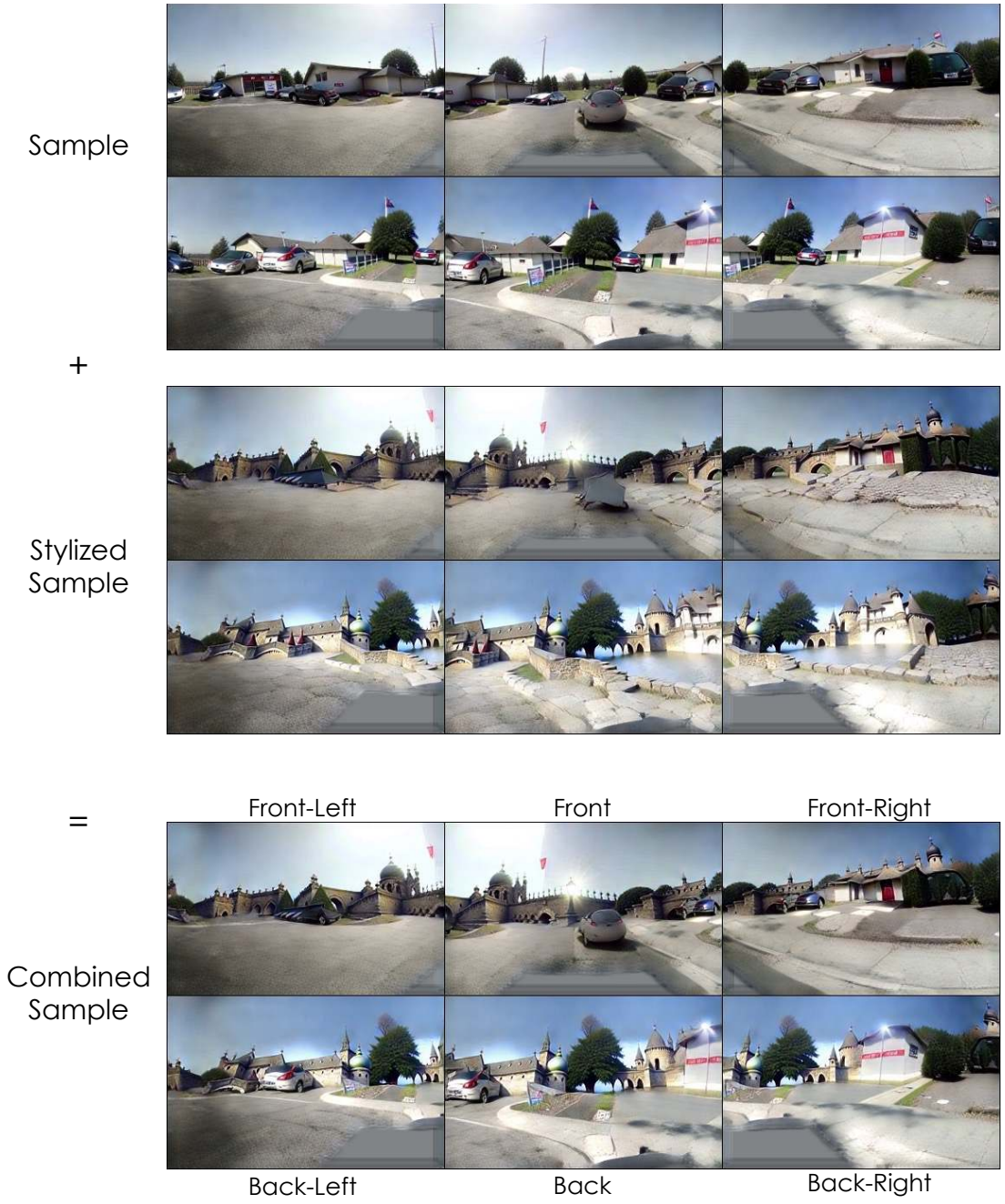

Figure 15. Combining voxels: we replace the center part of the stylized voxel with that of the sample at the top.

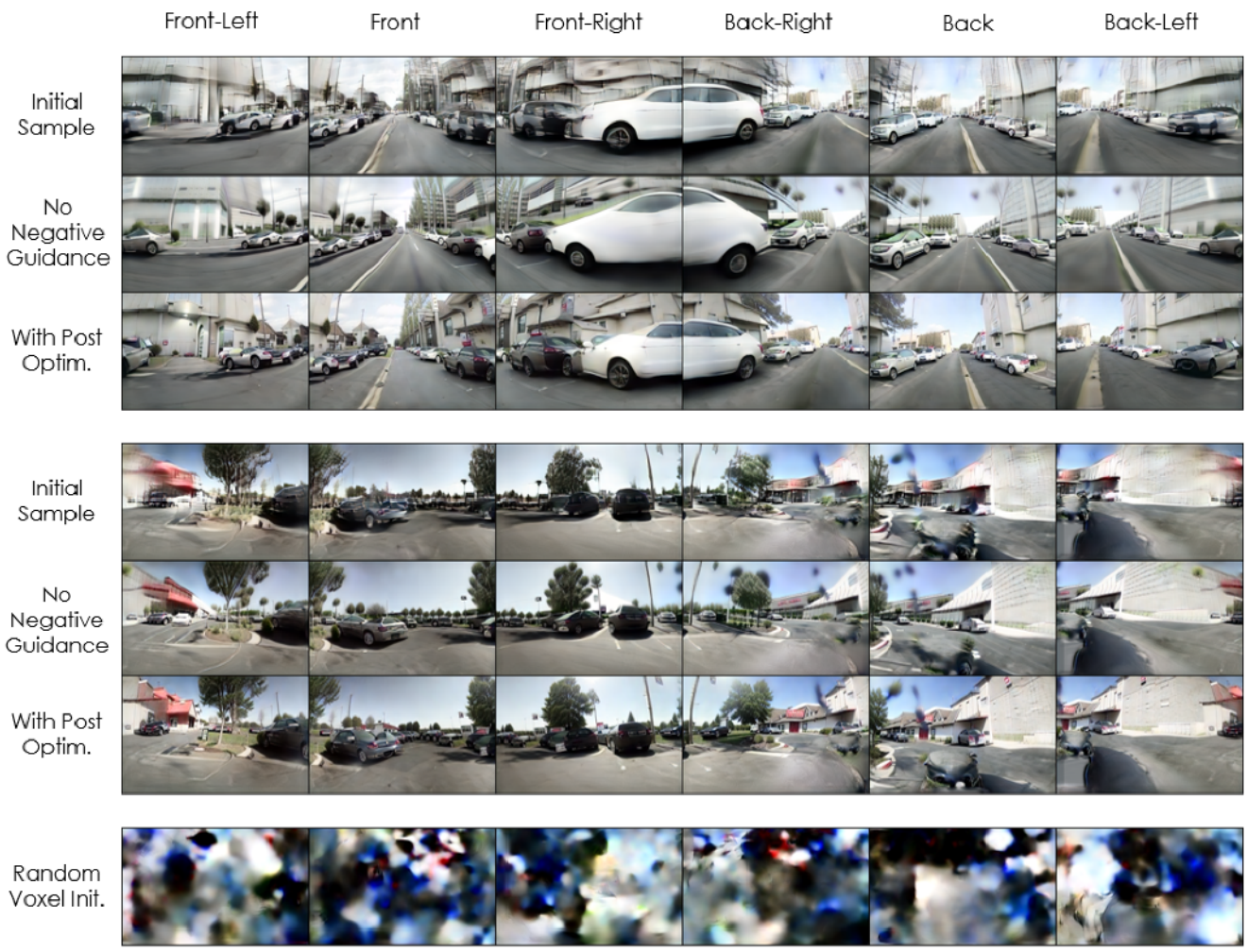

Figure 16. Ablating the post-optimization method. We show initial samples, samples optimized with classifier-free guidance and samples optimized with negative guidance for two scenes. Additionally, we show the result of post-optimization using a random gaussian intialization for the voxels.

2.4. Bird's Eye View Conditioned Synthesis

We provide additional Bird's Eye View conditioned synthesis results in Figure 17.

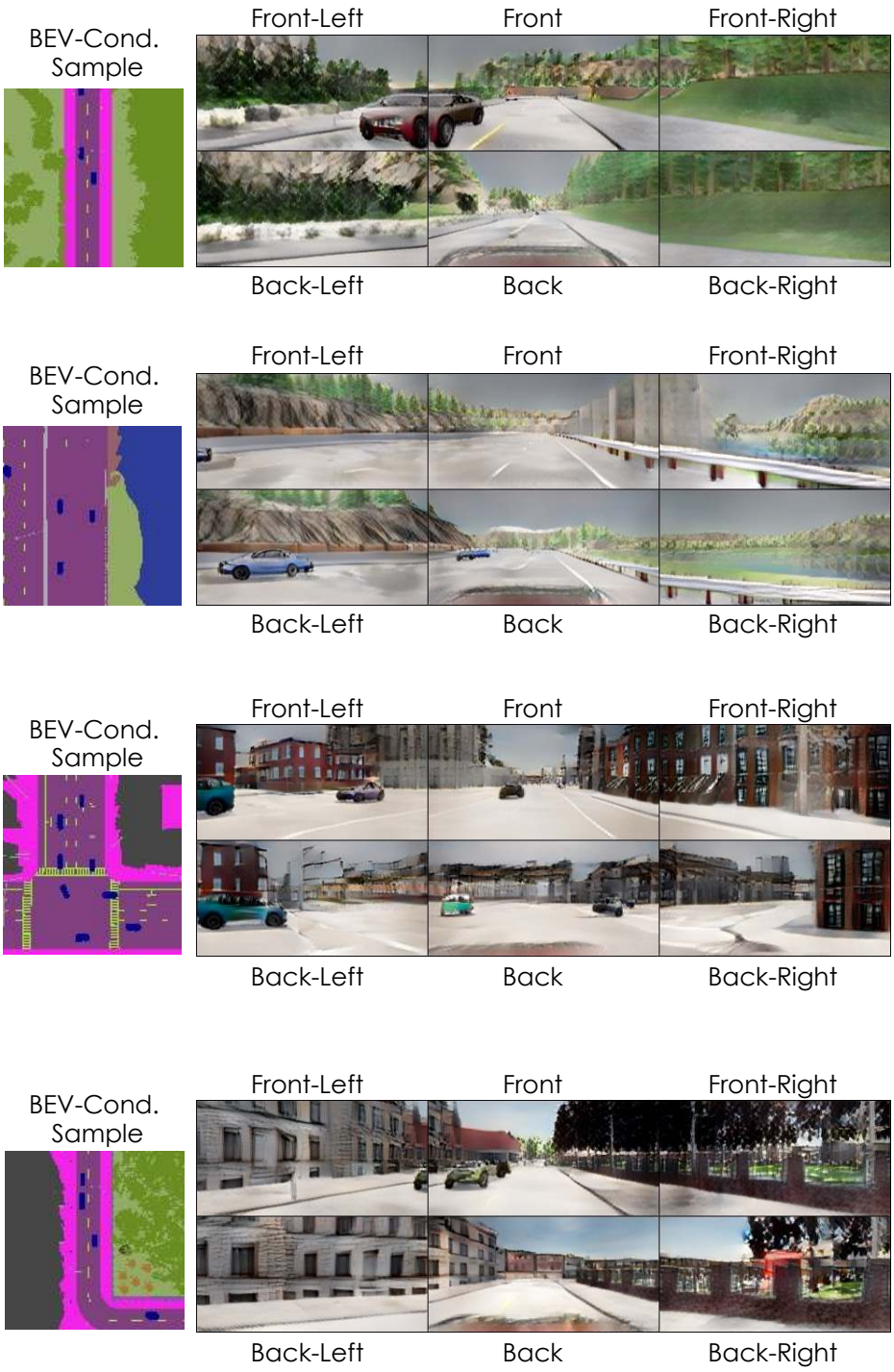

Figure 17. Additional results on Birds' Eye View Conditioned generation. In the BEV segmentation map, colors denote different region types: green - trees and vegetations, blue - water, grey - buildings, purple - road, pink - sidewalk, dark blue - vehicles (note that the ego car is at the center and thus not visualized).

2.5. Scene Editing

We provide additional scene editing results in Figure 18 and 19.

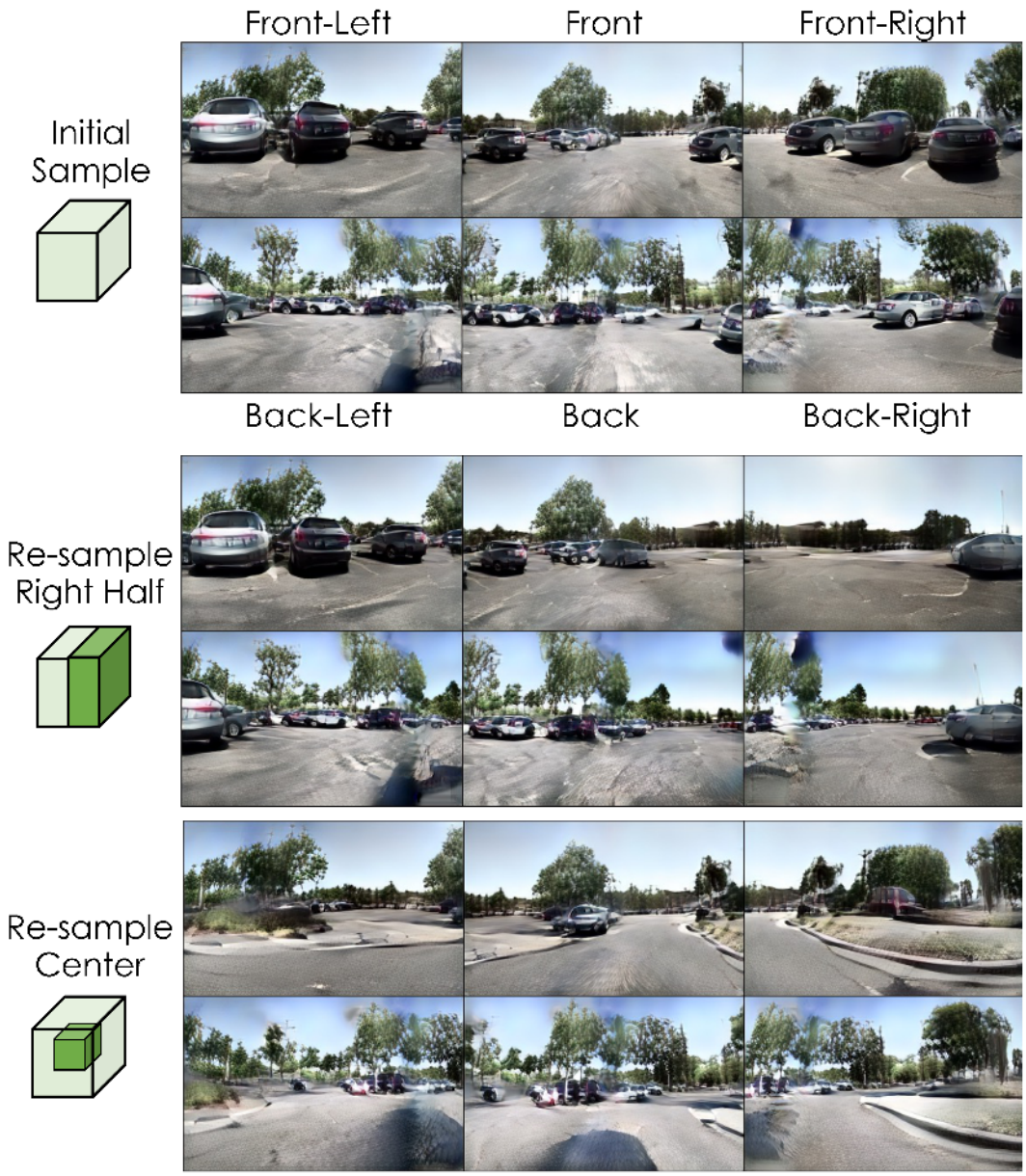

Figure 18. Additional results on scene editing by re-sampling. Given an initial sample, we edit the specified regions by re-sampling them with our model.

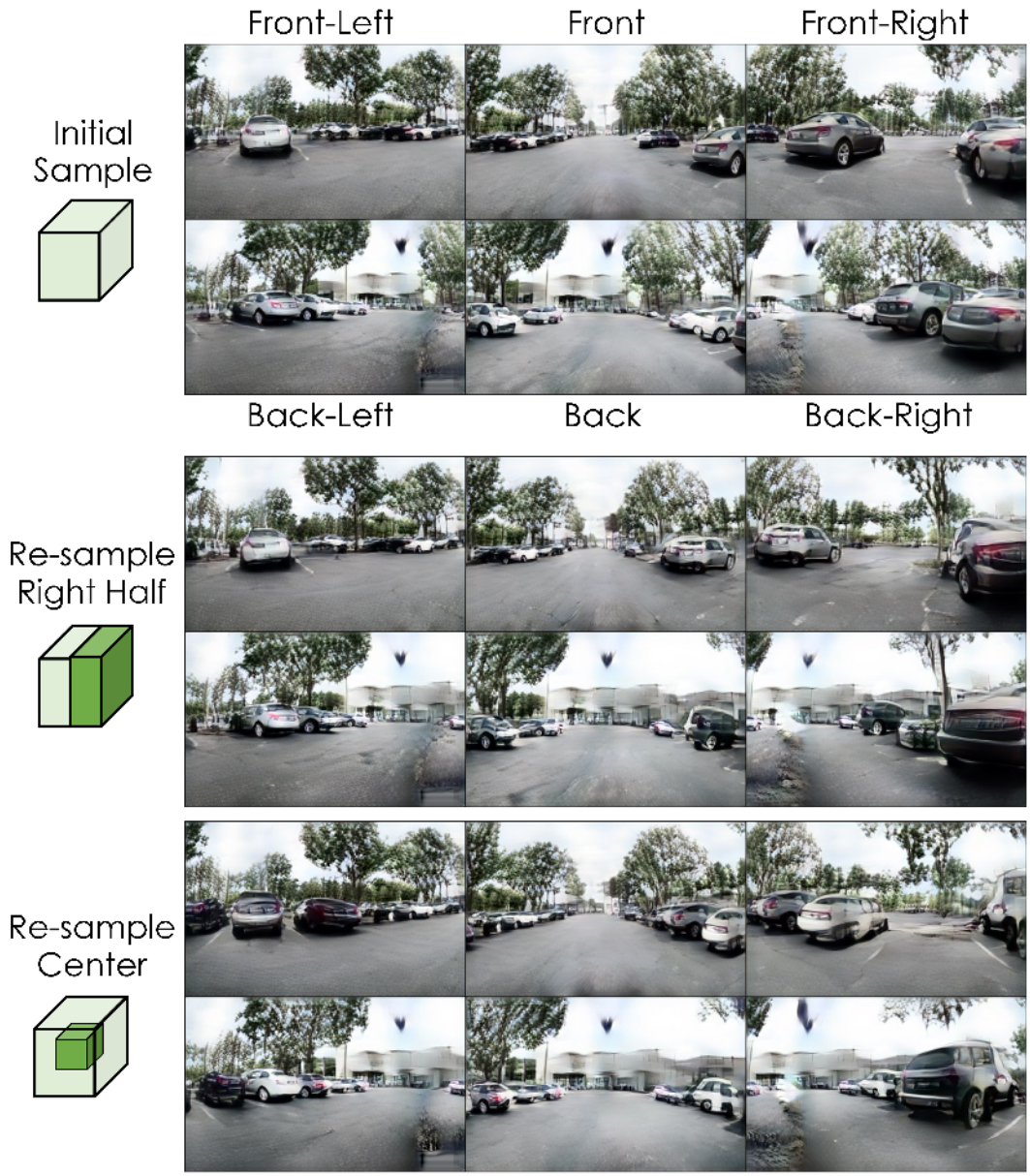

Figure 19. Additional results on scene editing by re-sampling. Given an initial sample, we edit the specified regions by re-sampling them with our model.

2.6. Text-Guided Style Transfer

In addition to stylization using SDS loss, which is effective for high-quality large structural modifications, in Figure 20, we show results of applying CLIP directional loss [5] 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 2.3) produces much higher quality results.

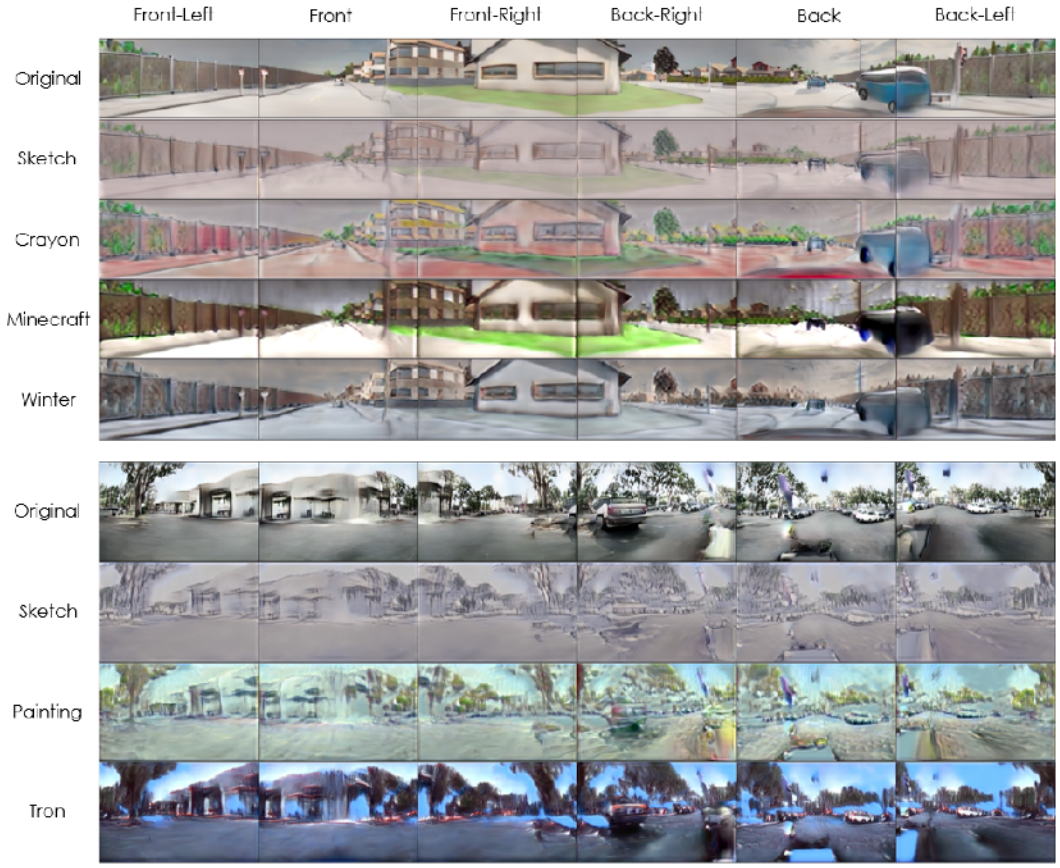

Figure 20. Text-guided style transfer results on an encoded Carla scene and a sampled AVD scene. Each result was obtained using a decoder finetuned by running CLIP directional loss with the specified style on a separate scene.

References

[1] 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. 2

[2] 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. 2

[3] 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. 2

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

[5] Rinon Gal, Or Patashnik, Haggai Maron, Gal Chechik, and Daniel Cohen-Or. Stylegan-nada: Clip-guided domain adaptation of image generators, 2021. 25

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[7] 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

[8] 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. 1

[9] 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

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Machine learning (ICML)*, 2015. 4

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

[12] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Trans. on Graphics*, 1987. 7, 13

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 5

[14] 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. 14

[15] 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

[16] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *ECCV*, 2020. 7

[17] 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. 6

[18] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 2, 4, 5, 6, 14

[19] 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. 2, 4

[20] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 4

[21] 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. 4

[22] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. 5

[23] 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. 4

[24] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. of the International Conf. on Machine learning (ICML)*, 2019. 1

[25] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv*, 1607.08022, 2016. 1

[26] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 2, 4

[27] 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. 2

[28] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 2

[29] 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. 1