Supplementary Material

Towards Generating Realistic Autonomous Driving 3D Semantic Training Data

This supplementary material provides further information regarding the method presented in the main article. Sec. 1 provides further details about the network architectures used as the variational auto-encoder (VAE) and as the denoising diffusion probabilistic model (DDPM). Sec. 2 provides further information regarding the binary cross-entropy (BCE) and dice losses used in the VAE training. Sec. 3 presents further assessments of our model regarding the pruning layers and efficiency compared to XCube [15], which employs the standard multi-model hierarchical scene generation strategy. Sec. 4 presents further experiments on Waymo dataset [17]. Sec. 5 shows the class IoU from the experiments done in Sec. 4.2 in the main article. Sec. 6 presents additional qualitative results from conditional and unconditional generation from our method. Finally, Sec. 7 discuss current limitations of our proposed method.

1 Architectures

This section presents the network architectures used in our approach. The VAE is trained to encode the scene $\mathcal{P}$ in the latent $\mathcal{Z}$, which is then padded to a dense grid $\boldsymbol{Z}$ such that the VAE decoder can reconstruct $\hat{\mathcal{P}} \approx \mathcal{P}$ from it. The VAE architecture layers employ sparse operations to enable the processing of large-scale data and avoid exponential memory growth. The DDPM is trained with the VAE learned latent $\boldsymbol{Z}$ by receiving as input the noisy latent $\boldsymbol{Z}^t$ at step t, predicting $\mathbf{v}^t$ following the $\mathbf{v}$-parameterization formulation [16], where $\mathbf{v}$ is parameterized in terms of the sampled noise $\boldsymbol{\epsilon}$ and the uncorrupted data $\boldsymbol{Z}^0$. Given the dense latent $\boldsymbol{Z}$, the DDPM uses dense convolutional operations. Both model architectures are depicted in Fig. 9.

The conditional DDPM model architecture is depicted in Fig. 10. The condition encoder computes a latent dense condition token $\mathcal{C}$ from the input point cloud $\mathcal{P}$. Then, at each l^{th} layer from the DDPM, the layer latent feature $\mathcal{F}_l$ is conditioned on the denoising step t and the latent condition token $\mathcal{C}$. The step t is first encoded by a positional encoder layer [13] and then projected to the current layer feature dimension with a linear layer, multiplying the projected feature by each element in the layer feature grid $\mathcal{F}_l$. The condition token $\mathcal{C}$ is projected to the current layer feature dimension and shape with a convolutional layer, merged to the layer latent grid $\mathcal{F}_l$ via an element-wise multiplication, and finally computing the conditioned feature grid $\mathcal{F}'_l$.

2 Pruning Losses

This section details the $\mathcal{L}_{\text{bce}}$ and $\mathcal{L}_{\text{dice}}$ losses used to learn the pruning mask at each VAE upsampling layer. As mentioned in Sec. 3.1 in the main article, the VAE is trained with a cross-entropy loss as the semantic loss and the KL divergence as the latent loss. To learn the pruning masks, we follow recent mask-based segmentation approaches [3], [4], [10], [11], supervising the mask prediction with the $\mathcal{L}_{\text{bce}}$ and $\mathcal{L}_{\text{dice}}$ losses. The $\mathcal{L}_{\text{bce}}$ aims at predicting whether an individual voxel is occupied or not at a given l^{th} upsampling layer and is computed as:

$$\mathcal{L}^l_{\text{bce}} = -(\boldsymbol{m}_l \log(\hat{\boldsymbol{m}}_l) + (1 - \boldsymbol{m}_l) \log(1 - \hat{\boldsymbol{m}}_l)). \tag{11}$$

where $\boldsymbol{m}_l$ is the target pruning mask and $\hat{\boldsymbol{m}}_l$ is the predicted mask. At the same time, the dice loss aims at predicting the whole scene layout by computing the dice coefficient, which approximates the IoU computation and maximizing this computation as:

$$\text{dice}^l = \frac{2\,|\hat{\boldsymbol{m}}_l \cap \boldsymbol{m}_l|}{|\hat{\boldsymbol{m}}_l| + |\boldsymbol{m}_l|}, \tag{12}$$

$$\mathcal{L}^l_{\text{dice}} = 1 - \text{dice}^l, \tag{13}$$

where as in Eq. (11), $\boldsymbol{m}_l$ is the target pruning mask and $\hat{\boldsymbol{m}}_l$ the predicted mask. Therefore, both losses complement each other, predicting both the individual voxel occupancy and the layout of the entire scene.

3 Ablation Studies

In this section, we evaluate our proposed approach to support our choice of a convolutional DDPM model instead of standard transformer-based DDPM [14], and also evaluate our model in terms of efficiency. In Sec. 3.1, we compare the quality of the scenes generated by our method with a convolution- and a transformer-based DDPM. Sec. 3.2 compares the memory usage of our model with and without the pruning layers to show the advantage of learning and removing unoccupied voxels. Finally, Sec. 3.3 evaluates the efficiency of modeling the coarse-to-fine scene generation within a single model in comparison to the standard hierarchical models strategy [2], [6], [12].

3.1 Convolution vs Transformer Layers

In this experiment, we compare the quality of scenes generated by our method using a convolutional- and a transformer-based DDPM. Tab. 11 presents the MMD and mIoU results between both DDPM architectures trained with the same setup. As shown, the convolution-based DDPM outperformed the transformer DDPM in both metrics, especially in the mIoU experiment. At first, this seems counterintuitive given that recent generative approaches commonly rely on transformer-based architectures. However, this can be explained by the fact that for generating solely semantic labels, local information is more important than global context for generating locally consistent objects and structures. The underperformance of transformer-based DDPM can also be explained by the fact that such architectures require longer training and larger

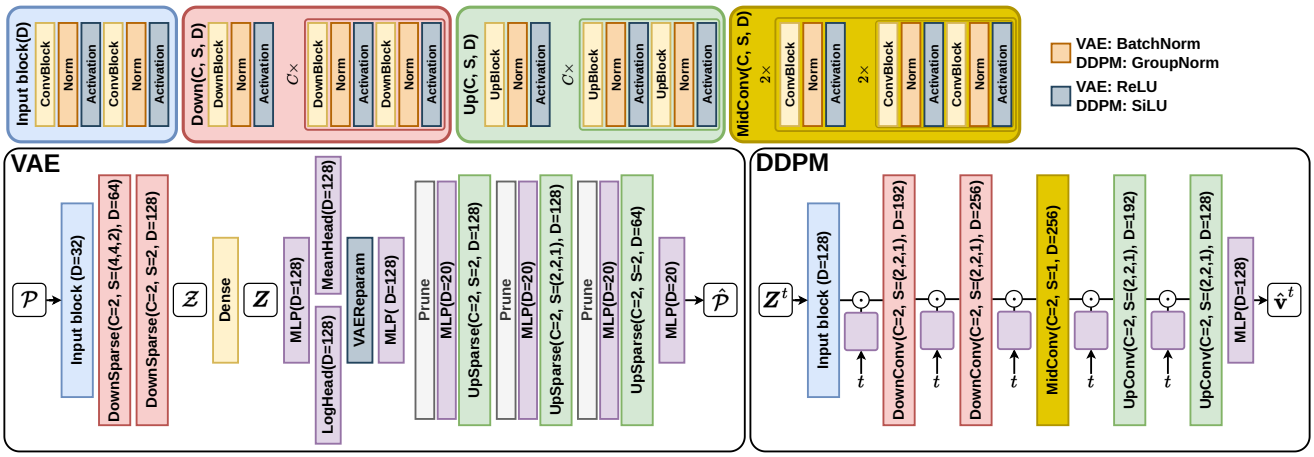

Fig. 9. VAE and DDPM model architectures. The VAE receives the voxelized point cloud $\mathcal{P}$, encode it to the latent $\mathcal{Z}$ which is densified to $\boldsymbol{Z}$ and decoded to $\hat{\mathcal{P}}$. The DDPM receives the noisy latent $\boldsymbol{Z}^t$ at step t and predicts $\hat{\mathbf{v}}^t$ following the $\mathbf{v}$-parameterization formulation [16].

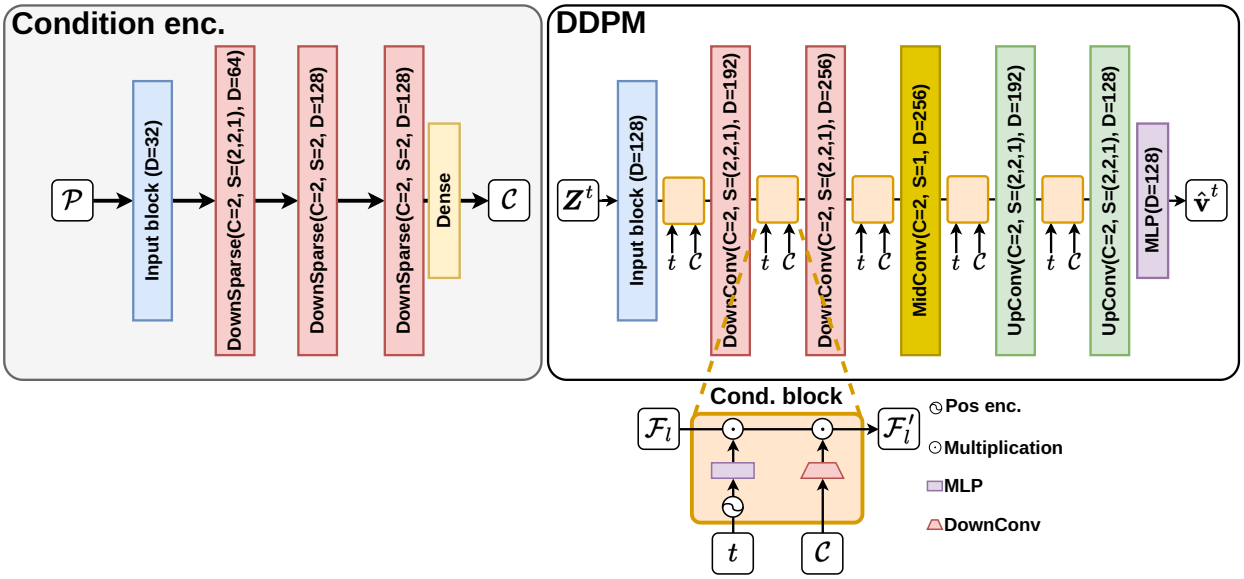

Fig. 10. Condition encoder and DDPM model architecture. The condition encoder receives the voxelized point cloud $\mathcal{P}$, encode it to the condition latent token $\mathcal{C}$. The DDPM receives the noisy latent $\boldsymbol{Z}^t$ at step t and the condition latent token predicts $\mathcal{C}$. The condition block projects the step t with an MLP and downsample the condition latent token $\mathcal{C}$ to the same latent resolution at the current DDPM layer, multiplying both into the layer feature $\mathcal{F}_l$ and computing the conditioned latent $\mathcal{F}'_l$ at each l^{th} layer in the DDPM.

	MMD ↓	mIoU [%] ↑
Transformer-based DDPM	0.0806	40.83
Convolution-based DDPM	**0.0733**	**53.09**

TABLE 11. MMD and mIoU between real and synthetic data comparing transformer- and convolution-based DDPMs.

datasets. Nevertheless, both DDPM architectures outperformed the baselines, highlighting once more the advantages of our proposed method in comparison to prior works. Given those results, we decided for using a convolution-based DDPM in the experiments reported in the main paper.

3.2 Pruning Layers

In this experiment, we aim to evaluate the gain in computational memory by using the pruning layer before each upsampling stage. Tab. 12 presents the size of the latent features for each upsampling layer in our VAE model. Without pruning voxels, the memory requirement grows exponentially since all voxels, occupied and unoccupied, are still processed. When using the pruning layers, the model learns which voxels are unoccupied, removing them from the latent features, enabling a more efficient computation and avoiding an exponential increase in memory requirements. Without pruning, the large memory requirements make it intractable to process and generate scene at the target resolution of 0.1 m, requiring either reducing the resolution of the target scene or leveraging image projections as done by prior works [9], [7].

3.3 Computation Efficiency

In this experiment, we assess the gain in efficiency of our coarse-to-fine scene generation modeling using a single model compared to standard multi-model hierarchical approaches [15], [2], [12], [6]. In this evaluation, we compute the inference time across 20 samples using our proposed approach and the hierarchical baseline XCube [15] and report the average inference time in Tab. 13. As

	Feature size [MB]				Forward Pass Time [s]
	Up Layer 1	Up Layer 2	Up Layer 3	Up Layer 4	
Without pruning	32.0	256.0	1056.0	4224.0	8.955
With pruning	4.9	20.1	64.4	121.7	0.720

TABLE 12. Per-layer comparison between the model with and without pruning, computing both feature sizes (MB) and forward pass time (seconds), averaged over 100 scene samples.

	Inference [s]	#params VAE	#params DDPM
XCube [15]	181.1	63.2M	778.6M
Ours	58.3	6.5M	66.2M

TABLE 13. Comparison of inference time and number of parameters between our method and the multi-model baseline XCube [15].

	MMD ↓	mIoU [%] ↑
Validation set	-	41.21
XCube [15]	**0.033**	39.10
Ours	0.050	**40.20**

TABLE 14. Maximum mean discrepancy (MMD) and mIoU evaluation between real and synthetic data on Waymo dataset.

shown, in addition to achieving more realistic results (as reported in the main paper), our method is also more efficient in terms of computation time and memory requirements. As shown in Tab. 13, our method requires about $3\times$ less time to generate a scene since it relies solely on a single stage to directly generate the high-resolution scene. In comparison, the hierarchical baseline performs the denoising diffusion process over each hierarchical stage, leading to slower inference.

In addition, our approach requires about $10\times$ fewer parameters compared to the hierarchical baseline. Besides the slower inference, the training of hierarchical decoupled models also leads to larger memory requirements given that multiple models are necessary to perform the scene generation. Our approach, in comparison, models the coarse-to-fine scene generation within a single model, allowing for a more efficient representation. This further highlights the advantage of our approach, achieving better performance while also being more efficient compared to the hierarchical scene generation approaches.

4 Waymo Dataset

To further assess the capabilities of our method, we train our model on the Waymo Open Dataset [17]. We train the VAE for 14 epochs and train the DDPM for 30 epochs, using the same optimizer and learning rate as for the SemanticKITTI dataset [1], [5]. Given that the Waymo dataset is about five times larger than the SemanticKITTI, we train the semantic segmentation network for 3 epochs on Waymo to achieve roughly the same number of training iterations.

We compare our method with XCube [15] using the publicly available checkpoint for their model. We note that this comparison is not fair with our method, as our model was trained with limited GPU resources compared to the provided checkpoint. In the main paper, however, the assessment provided a fairer comparison given that all methods were trained with the same hardware specifications. Despite that, as shown in Tab. 14 our method achieves competitive performance with the baseline even though it was trained with fewer GPU resources, and also while being more efficient as presented in Tab. 13. Fig. 11 presents a qualitative comparison between the scenes generated by our method and XCube [15]. As seen in the main experiments, the baseline generates smoother scenes, which might be visually appealing but are far from real scenes. Our method better imitates the sharper structures present in the real data, which makes it more suitable for use as training data, as shown in the main paper. This assessment with Waymo dataset [17] agrees with the experiments on SemanticKITTI present in the main paper. This suggests that the gain in performance when training the model with synthetic generated data is not limited to SemanticKITTI dataset, potentially also leading to improved performance in other datasets.

5 Quantitative Results

In this section we provide further quantitative results regarding the class IoU from the scenes generated by our method and the baselines. In Tab. 15, we report the class IoU from the model trained with real data evaluated on the real validation set and the scenes generated by the different methods. Similar to Tab. 2 from the main article, we report the numbers at two resolutions, 0.2 m, and 0.1 m, since PDD [9] and SemCity [7] can only generate scenes up to 0.2 m resolution. Similar to what was presented in Tab. 2, at 0.2 m resolution, all the methods achieve similar performance still our method achieves the best mIoU, as shown in Tab. 2. At 0.1 m resolution, our method surpasses the method able to generate scenes at this resolution, achieving the best performance in all classes compared to the scene generated by the baseline. Tab. 16 depicts the class IoU results from the experiment presented in Tab. 9 from the main article. As seen, with the curation process described in Sec. 4.4 the model performance achieves comparable to the best performing model, i.e., trained with additional 75% of labels, even if using less synthetic training data. This suggests that the curation process can improve the model performance when using synthetic scenes as training data by selecting only the most realistic scenes. This curation helps to bridge the gap between real and synthetic data by removing less realistic samples.

6 Qualitative Results

In this section, we show further examples of scenes generated by our method. Figs. 13 to 15 shows the unconditionally generated scene examples. As shown, the randomly generated point clouds present different scenarios. This variability agrees with the discussion in Sec. 4.2 where the training with a mixed set of samples from real and synthetic scenes performed better than using only real scenes. The synthetic scenes add variability to the data compared to the real training set since the real scenes were collected sequentially, where consecutive scenes may have

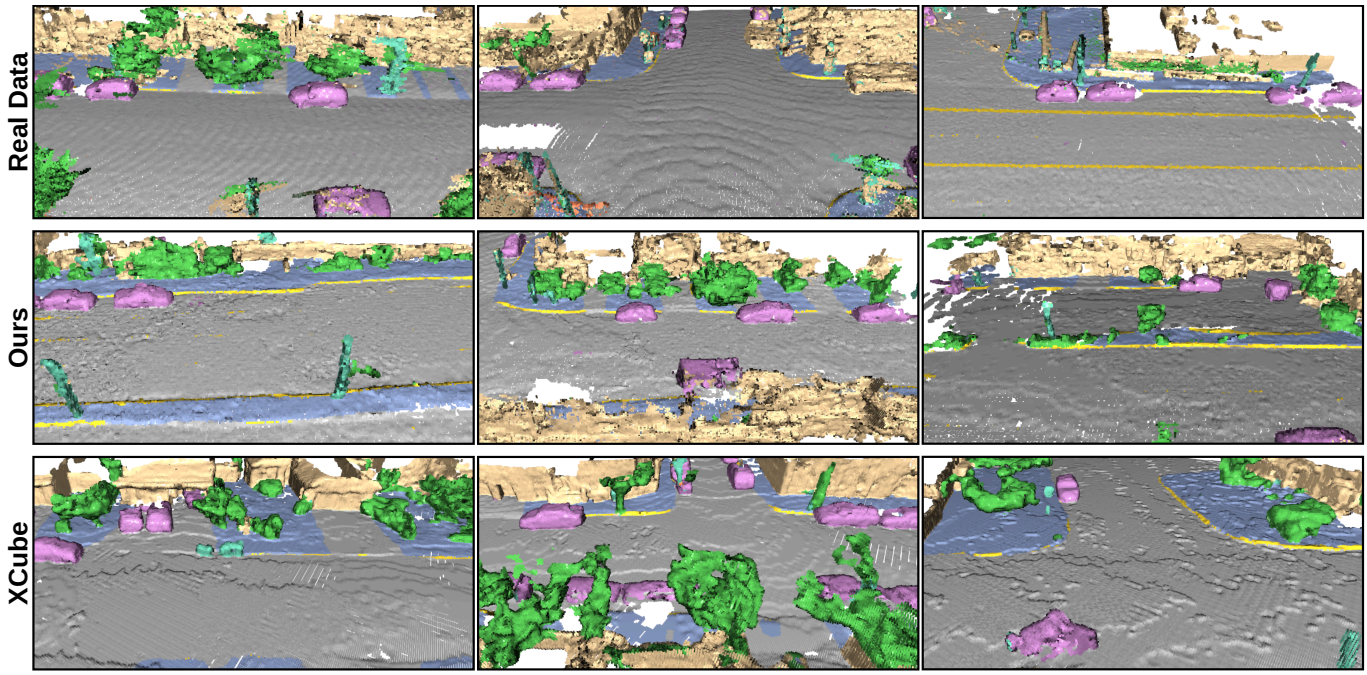

Fig. 11. Qualitative comparison of Waymo results between groun truth and scenes generated by our method and XCube [15].

	IoU [%] ↑													
	Res.	car	truck	OV	road	park.	sidewalk	build.	fence	veg.	trunk	terrain	pole	sign
Val. scenes	0.2	91.10	45.90	31.73	87.76	34.87	58.18	86.33	25.32	86.64	42.79	64.11	54.75	47.72
PDD [9]	0.2	80.24	**11.81**	0.29	81.14	8.99	57.00	79.19	34.82	78.71	7.96	54.65	15.67	0.81
SemCity [7]	0.2	80.46	3.72	0.71	85.81	24.22	66.61	78.40	36.32	**82.85**	21.50	51.95	24.18	1.78
XCube [15]	0.2	47.94	11.62	13.28	82.75	18.19	54.70	67.86	36.08	76.91	25.57	51.01	35.47	17.74
Ours	0.2	**90.72**	8.27	**19.07**	**89.08**	**31.05**	**72.52**	**89.67**	**42.20**	79.55	**42.73**	**60.33**	**47.39**	**27.06**
Val. scenes	0.1	93.77	42.24	49.00	92.16	46.52	69.24	89.28	41.08	88.07	51.88	65.81	61.12	58.36
XCube [15]	0.1	15.42	1.68	3.33	71.26	2.84	42.39	45.69	23.69	68.30	22.04	39.39	24.22	9.70
Ours	0.1	**92.29**	**10.42**	**24.10**	**92.48**	**33.74**	**77.34**	**89.75**	**47.73**	**81.81**	**42.99**	**61.72**	**43.55**	**28.79**

TABLE 15. Class-wise IoU evaluated on real data validation set and synthetic data generated by the different methods with semantic segmentation model trained on real data.

few changes. In contrast, by using randomly generated scenes, each individual sample will be different from each other, adding variability to the data, thus improving the performance of the model when trained with this synthetic training set. Regarding the conditioned scene generation, Figs. 16 and 17 show examples of scenes generated conditioned to LiDAR scans from KiTTI-360 [8] and to our own collected data, respectively. In those examples, we can notice that the model is able to generate reasonable scenes according to the scan used as condition. Such conditioning also works for point clouds collected with a LiDAR sensor different from the one used to train the model, as seen in Fig. 17 from the scenes generated conditioned to Ouster LiDAR collected by us.

7 Limitations

In this section, we discuss the limitations of our proposed method. Fig. 12 presents examples of failure cases of the conditional generation samples from our method. Our conditional model was trained on SemanticKITTI scans, collected with a Velodyne HDL-64E with 64 beams. Our data was collected with an Ouster OS-1 with 128 beams. Both LiDARs have different numbers of beams and laser beam patterns, leading to a domain gap between the scans used during training and those used during conditioning. As shown in Fig. 12, when generating scenes from point clouds collected with different LiDARs, artifacts can be generated, leading to non-realistic semantic scenes. Still, this limitation can be tackled with a curation step, as discussed in the main paper.

References

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019. 3

[2] A. Bokhovkin, Q. Meng, S. Tulsiani, and A. Dai, "Scenefactor: Factored latent 3d diffusion for controllable 3d scene generation," *arXiv preprint*, vol. arXiv:2412.01801, 2024. 1, 2

[3] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020. 1

[4] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask transformer for universal image segmentation," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1

	IoU [%] ↑												
	car	truck	OV	road	park.	sidewalk	build.	fence	veg.	trunk	terrain	pole	sign
Real only	93.77	42.24	49.00	92.16	46.52	69.24	89.28	41.08	88.07	51.88	65.81	61.12	**58.36**
Real + 75% Synth.	94.01	70.49	48.27	**92.59**	**50.04**	**73.11**	**90.00**	**44.30**	**88.29**	52.26	**66.96**	**62.40**	57.51
Real + 25% Synth.†	**94.78**	**73.85**	**60.90**	92.47	49.85	71.91	89.65	43.16	87.83	**53.36**	65.56	62.10	55.94

TABLE 16. Dense scenes class-wise IoU evaluated on the real data validation set comparing the network trained only with the full real training set with the training with the full training set with additional 75% synthetic data generated with our method. OV refers to other-vehicle. † refers to the curated conditional generated scenes.

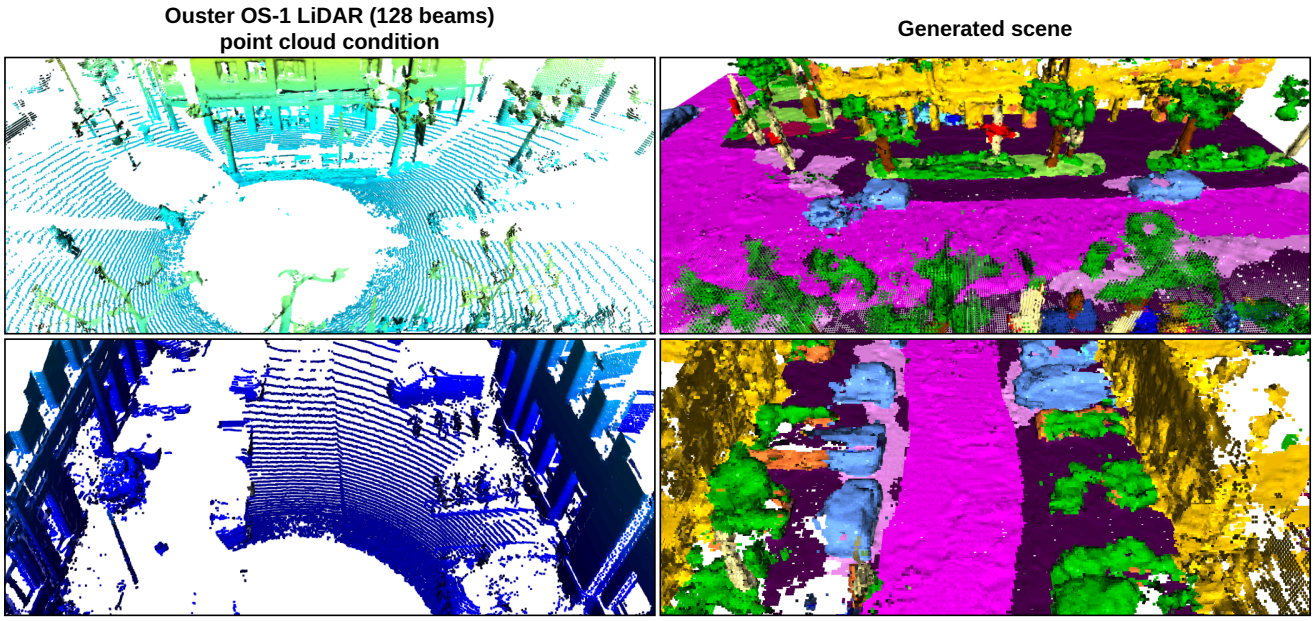

Fig. 12. Failure case example. Condition point cloud collected with a LiDAR (Ouster OS-1 128 beams) with a different resolution than the one used to collect the training data (Velodyne HDL-64E 64 beams).

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 3

[6] T. Hua, L. Jiang, Y.-C. Chen, and W. Zhao, "Sat2city: 3d city generation from a single satellite image with cascaded latent diffusion," in *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2025. 1, 2

[7] J. Lee, S. Lee, C. Jo, W. Im, J. Seon, and S.-E. Yoon, "Semcity: Semantic scene generation with triplane diffusion," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2, 3, 4

[8] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 45, no. 3, pp. 3292–3310, 2022. 4, 9

[9] Y. Liu, X. Li, X. Li, L. Qi, C. Li, and M.-H. Yang, "Pyramid diffusion for fine 3d large scene generation," in *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024. 2, 3, 4

[10] R. Marcuzzi, L. Nunes, L. Wiesmann, J. Behley, and C. Stachniss, "Mask-Based Panoptic LiDAR Segmentation for Autonomous Driving," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1141–1148, 2023. 1

[11] R. Marcuzzi, L. Nunes, L. Wiesmann, E. Marks, J. Behley, and C. Stachniss, "Mask4D: End-to-End Mask-Based 4D Panoptic Segmentation for LiDAR Sequences," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 11, pp. 7487–7494, 2023. 1

[12] Q. Meng, L. Li, M. Nießner, and A. Dai, "Lt3sd: Latent trees for 3d scene diffusion," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025. 1, 2

[13] L. Nunes, R. Marcuzzi, B. Mersch, J. Behley, and C. Stachniss, "Scaling Diffusion Models to Real-World 3D LiDAR Scene Completion," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 1

[14] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023. 1

[15] X. Ren, J. Huang, X. Zeng, K. Museth, S. Fidler, and F. Williams, "Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 1, 2, 3, 4

[16] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," in *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2022. 1, 2

[17] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 3

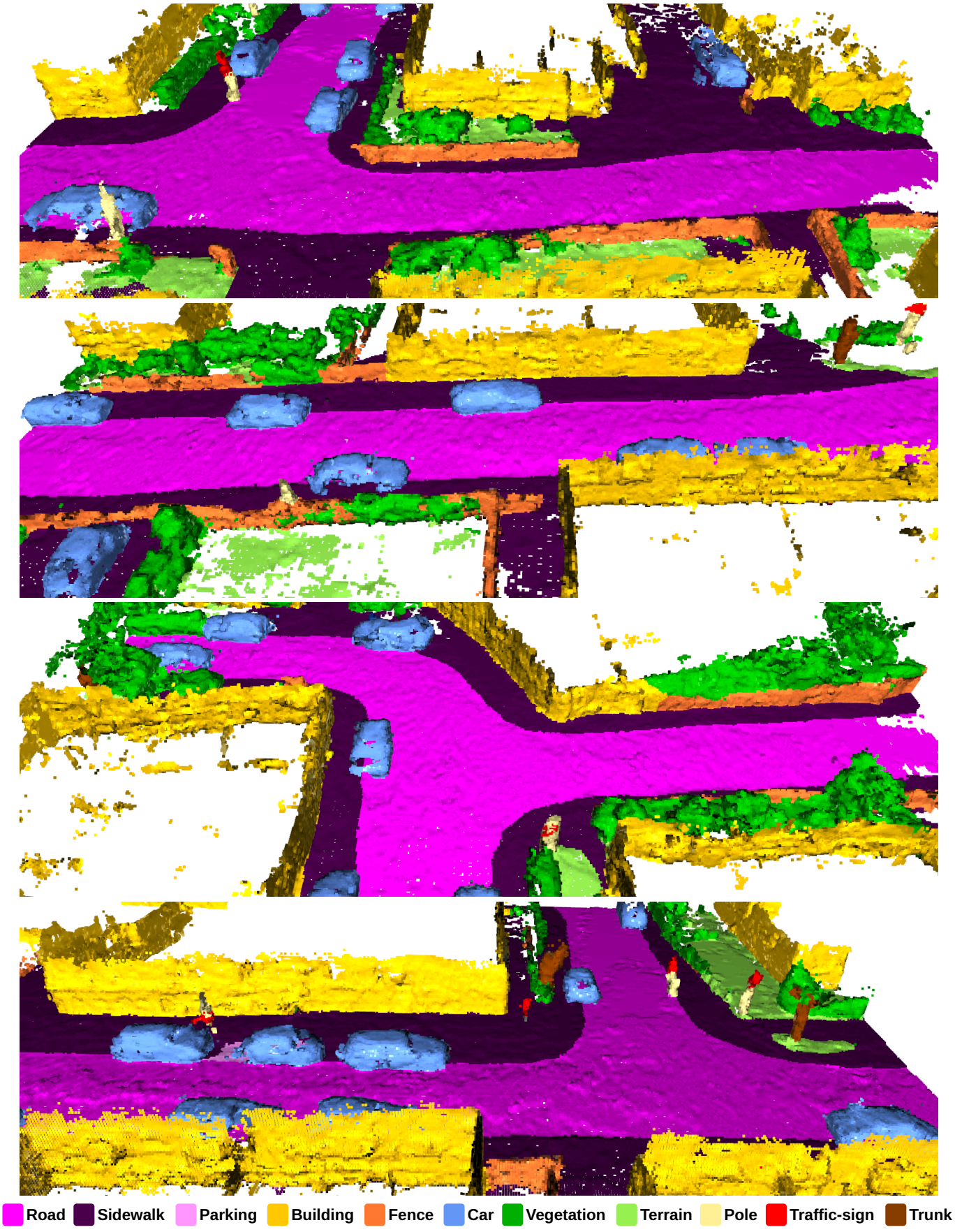

Fig. 13. Unconditional scenes generated by our method.

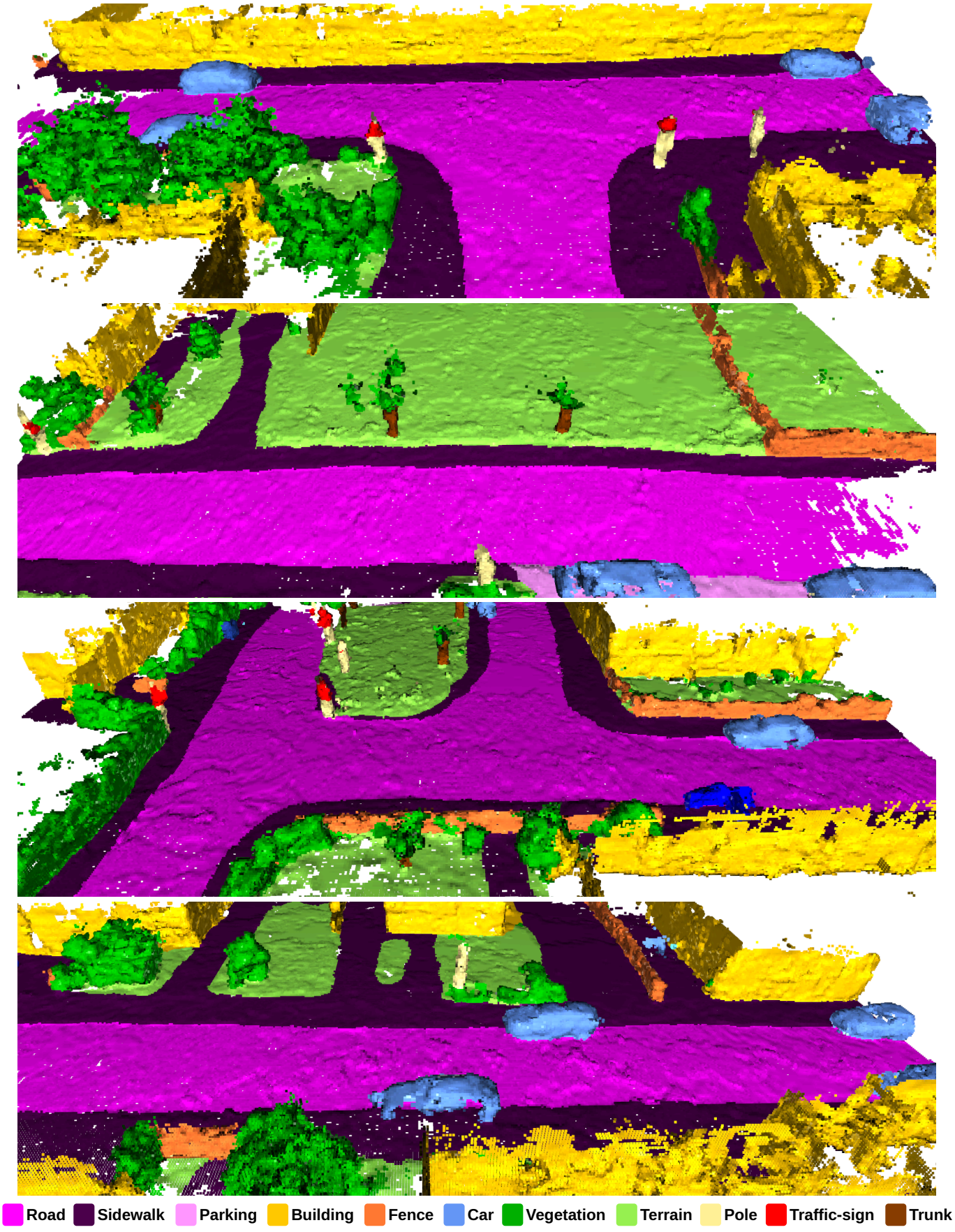

Fig. 14. Unconditional scenes generated by our method.

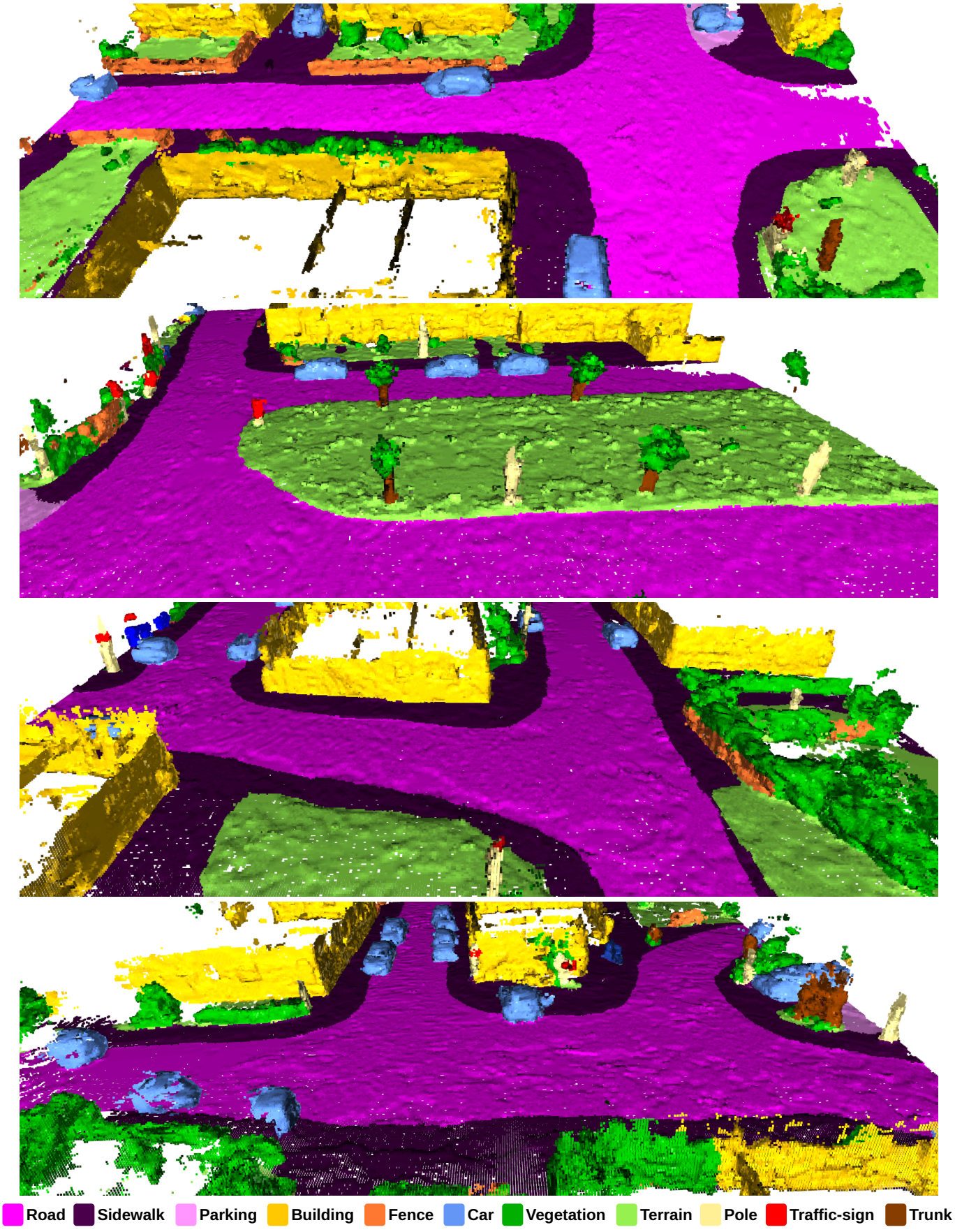

Fig. 15. Unconditional scenes generated by our method.

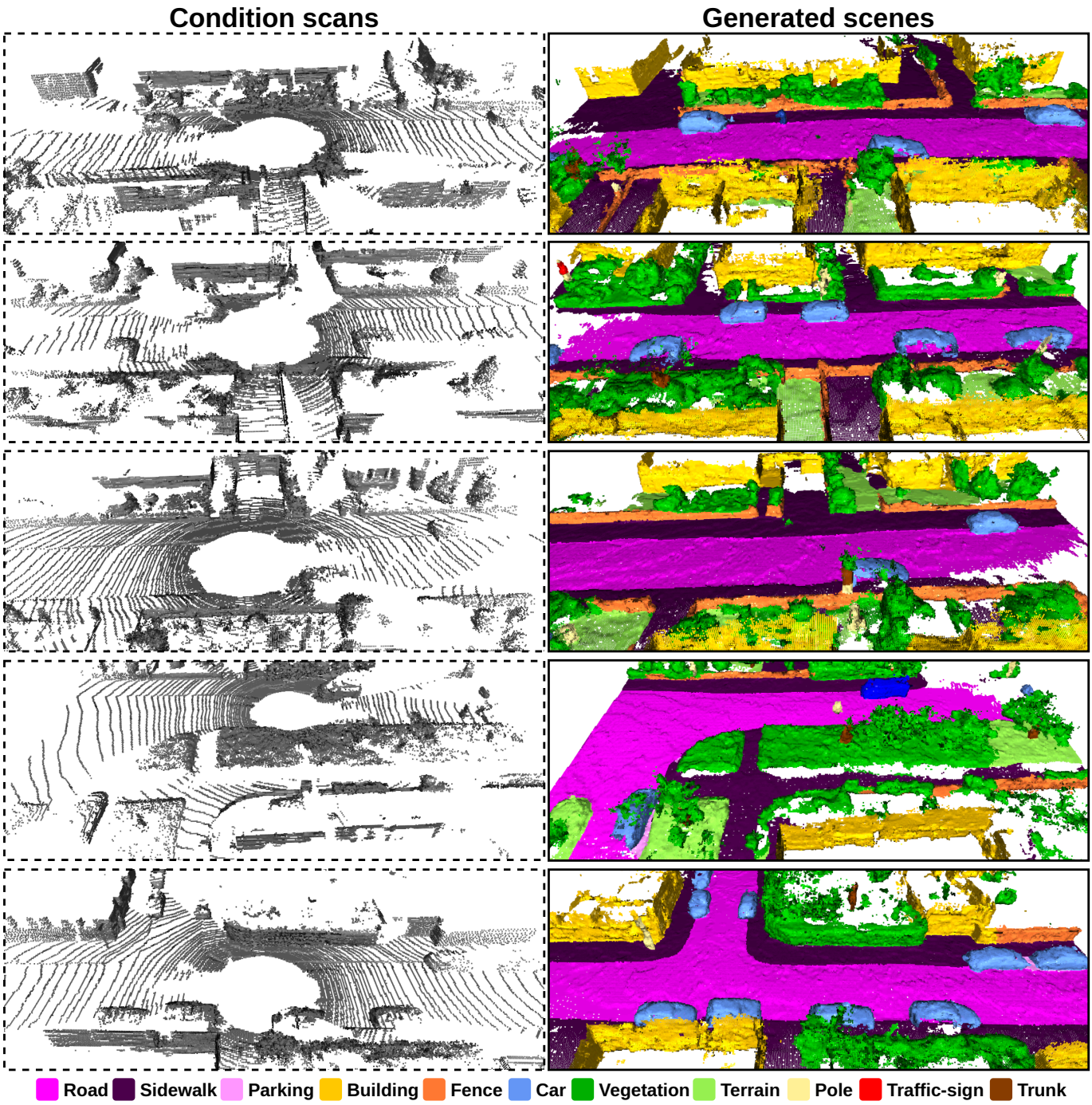

Fig. 16. Generated scenes conditioned with unseen scans from KITTI-360 dataset [8].

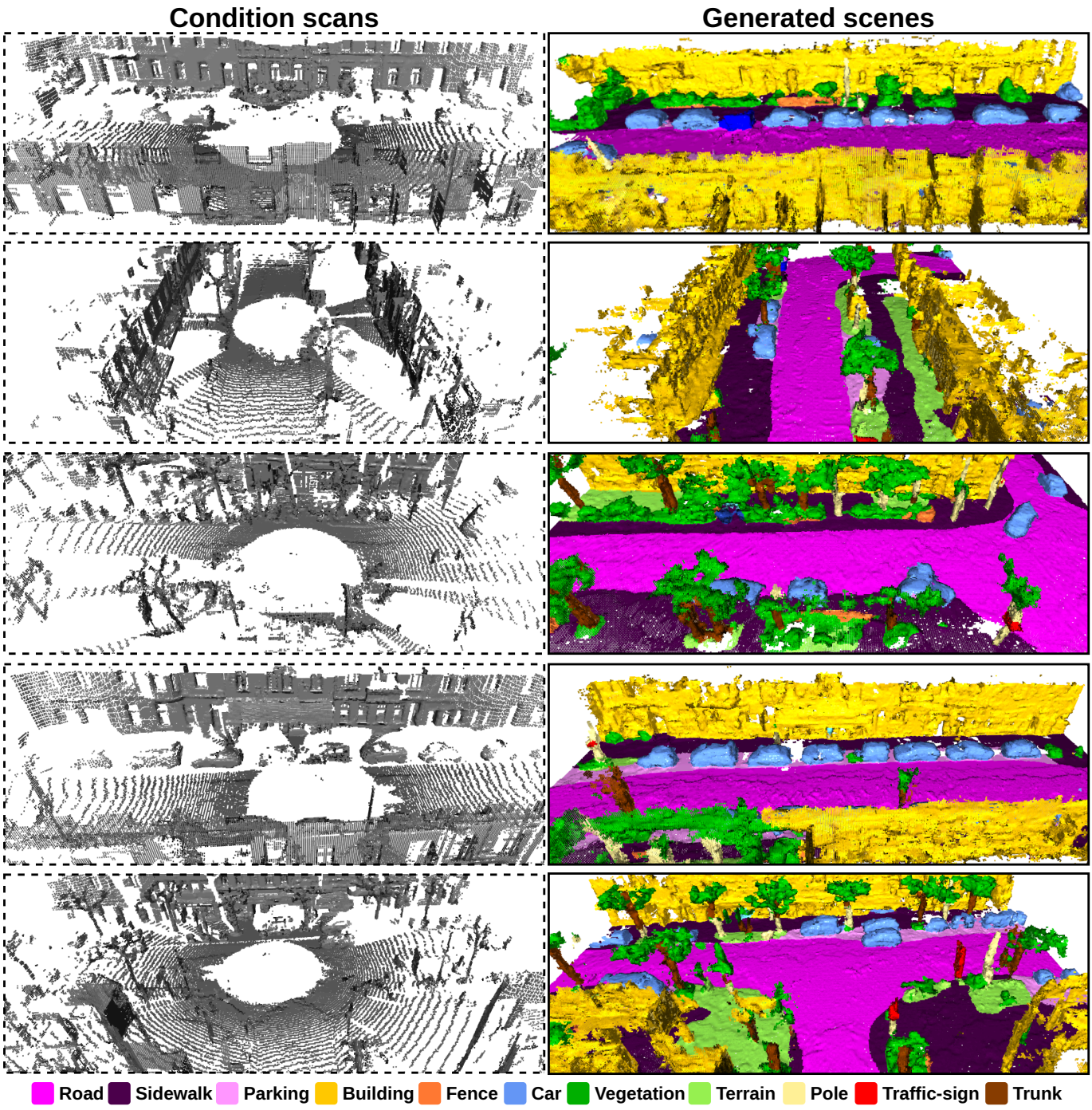

Fig. 17. Generated scenes conditioned with unseen scans from our data collected with an Ouster OS-1 128 beam LiDAR.