XCube: Large-Scale 3D Generative Modeling using Sparse Voxel Hierarchies – Supplementary Material –

Xuanchi Ren[1,2,3] Jiahui Huang[1] Xiaohui Zeng[1,2,3] Ken Museth[1]
Sanja Fidler[1,2,3] Francis Williams[1]
[1]NVIDIA [2]University of Toronto [3]Vector Institute

In this supplementary material, we provide additional details on our method and experiments. In Sec. 1, we describe our sparse 3D deep learning framework, and compare it to state-of-the-art implementations. In Sec. 2, we provide more implementation details for our method as well as precise definitions of our loss function and evaluation metrics. In Sec. 3, we provide more qualitative results on all the datasets we trained/evaluated on in the main paper. We additionally provide a **supplementary video** in the accompanying files to better illustrate our results.

1. Sparse 3D Learning Framework

All of our networks are implemented using a customized sparse 3D deep learning framework built on top of PyTorch. To represent sparse grids of features and perform efficient deep-learning operations (convolution, pooling, etc.) over them, we leverage NanoVDB [7], a GPU-friendly implementation of the VDB data structure [6]. A VDB tree is a variant of B+-Tree with four layers where the top layer is a hash table, followed by two internal layers (with branching factor 32^3, 16^3), followed by leaf nodes with 8^3 voxels.

To demonstrate the effectiveness of our VDB-based deep learning framework, we benchmark it against TorchSparse [11], a state-of-the-art sparse deep learning framework. As shown in Tab. 1, our custom framework is both fast and memory-efficient, especially for large input grid resolutions. Built upon the highly efficient VDB data structure, our 3D representation is compactly stored in memory and supports more efficient nearest neighbor lookup and processing than its hash table counterpart in [11]. Such a framework lays the foundation for our high-resolution 3D generative model and has potential to applications in many other downstream tasks such as reconstruction and perception.

	Voxel Grid Memory (MB) ↓		Convolution Forward Time (ms) ↓		
Grid Resolution	512^3	1024^3	32^3	256^3	1024^3
TorchSparse [11]	15.0	104.6	2.1	8.5	446.0
Ours	**3.6**	**8.4**	**0.5**	**5.0**	**149.6**

Table 1. Sparse 3D benchmark results.

2. Implementation Details

2.1. Loss Definition

Our model is able to output various attributes $\mathbf{A}$ defined on the voxel grids. Here we omit the level subscript l for simplicity. The direct output of the network at each voxel at each level includes surface normal $\boldsymbol{n} \in \mathbb{R}^3$, semantic label $\boldsymbol{s} \in \mathbb{R}^S$, and neural kernel features $\boldsymbol{\phi} \in \mathbb{R}^4$. Here the neural kernel features $\boldsymbol{\phi}$ are used for computing continuous TSDF values in 3D space for highly-detailed *subvoxel*-level surface extraction (using the techniques from [3]), and it could also be replaced with implicit features $\boldsymbol{q}$ to extract TUDF values for open surfaces as in [8]. The attribute loss $\mathcal{L}^{\text{Attr}}$, as mentioned in Eq. (6) of the main text, is a mixture of different supervisions, written as follows:

$$\mathcal{L}^{\text{Attr}} = \lambda_1 \underbrace{\|\boldsymbol{n} - \boldsymbol{n}_{\text{GT}}\|_2^2}_{\text{normal loss}} + \lambda_2 \underbrace{\text{BCE}(\boldsymbol{s}, \boldsymbol{s}_{\text{GT}})}_{\text{semantic loss}} + \lambda_3 \underbrace{\mathbb{E}_{\boldsymbol{x}\in\mathbb{R}^3} \|f(\boldsymbol{x}) - \text{TSDF}(\boldsymbol{x}, \boldsymbol{X}_{\text{GT}})\|_1}_{\text{surface loss}}, \tag{1}$$

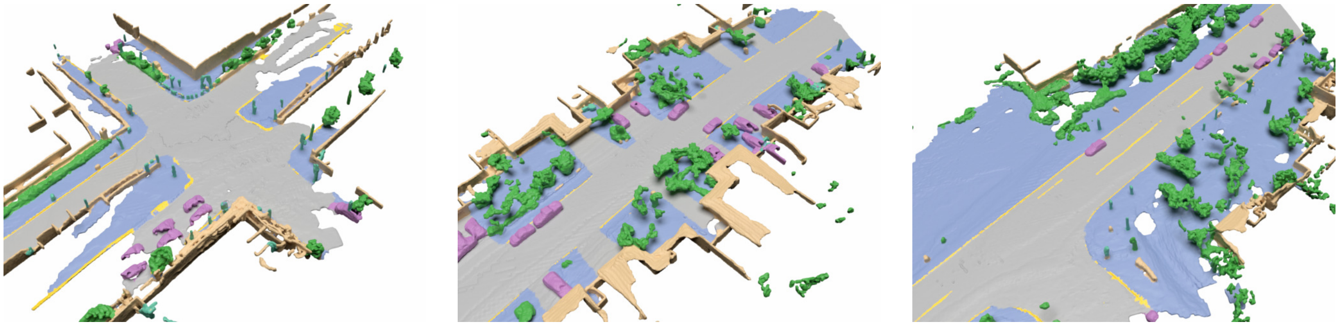


Figure 1. Results of micro-conditioning on Waymo dataset. The voxel number conditioning increases from left to right. There is a clear trend of increasing number of voxels and more diverse contents in the sampled scenes.

where $\boldsymbol{n}_{\mathrm{GT}}$ and $\boldsymbol{s}_{\mathrm{GT}}$ are the ground-truth normal and semantic label at each voxel, and $\boldsymbol{X}_{\mathrm{GT}}$ is the ground-truth dense point cloud of the surface. The surface loss is computed by sampling points $\boldsymbol{x}$ in the 3D space and comparing the predicted TSDF values $f(\boldsymbol{x})$ with the ground-truth TSDF values TSDF($\boldsymbol{x}$, $\boldsymbol{X}_{\mathrm{GT}}$). To compute $f(\boldsymbol{x})$ given arbitrary input positions, we leverage the predicted neural kernels $\boldsymbol{\phi}$ to solve for a surface fitting problem as in [3]:

$$f(\boldsymbol{x}) = \sum_v \alpha_v K(\boldsymbol{x}, \boldsymbol{x}_v) = \sum_v \alpha_v \boldsymbol{\phi}_v^\top \boldsymbol{\phi}(\boldsymbol{x}) K_b(\boldsymbol{x}, \boldsymbol{x}_v), \tag{2}$$

where v is the index of the voxels, $\boldsymbol{\phi}(\boldsymbol{x})$ is the neural kernel evaluated at the input position $\boldsymbol{x}$ using bezier interpolation from its nearby voxels, and $K_b(\boldsymbol{x}, \boldsymbol{x}_v) = B(\boldsymbol{x} - \boldsymbol{x}_v)$ is a shift-invariant Bezier kernel. The coefficients α_v are obtained by performing a linear solve as detailed in [3]. Similarly, for open surfaces we can replace the neural kernels $\boldsymbol{\phi}$ with implicit features $\boldsymbol{q}$ and define $f(\boldsymbol{x})$ as a local MLP function digesting trilinearly interpolated $\boldsymbol{q}$ at position $\boldsymbol{x}$ [8]. We set $\lambda_1 = 1$, $\lambda_2 = 15$, and $\lambda_3 = 1$ in our experiment. For the KL divergence, we normalize it by the number of voxels of the voxel grid and then use a loss weight $\lambda = 0.0015$ for all our experiments.

2.2. Conditioning

We explore diverse condition settings for our voxel diffusion models: (1) For the associated attributes from the previous level, we optionally concatenate them to the latent feature $\mathbf{X}$ before the latent diffusion. For example, for user-control cases, we do not concatenate them for flexibly adding or deleting voxels. (2) For the text prompts, we use cross-attention to fuse them into the latent. (3) For the category condition, we use AdaGN and fuse them with timestep embedding by adding. (4) For single scan conditions, we use an additional point encoder to quantize the single scan point cloud to a voxel grid and concatenate it with the latent feature $\mathbf{X}$.

Micro-conditioning. We found that the Waymo dataset suffers from missing voxels due to the sparsity of the LiDAR scans. To mitigate this issue, we use a micro-conditioning scheme following SD-XL [9] to inject additional condition to the diffusion backbone describing the number of the voxels. This helps when the dataset itself contains multi-modal distributions, and allows fine-grained control of the generated scale of the scene.

2.3. Texture Synthesis

While our model is focused on generating 3D geometry, we also explore the possibility of generating textures for the generated shapes. To this end, we use a state-of-the-art texture generator TEXTure [10] to create texture maps for the generated shapes. The method works by applying a sequence of depth-conditioned stable diffusion models to multiple views of the shape. Later steps in the process are conditioned on the previous steps, allowing the model to generate consistent textures with smooth transitions. We choose to decouple the geometry and texture generation processes to allow for more flexibility and controlability – *e.g.*, given the same geometry, different textures can be generated and selected. We demonstrate the effectiveness of the full pipeline on Objaverse dataset and use the same text prompts for both the geometry and the texture. Results are shown in Fig. 2.

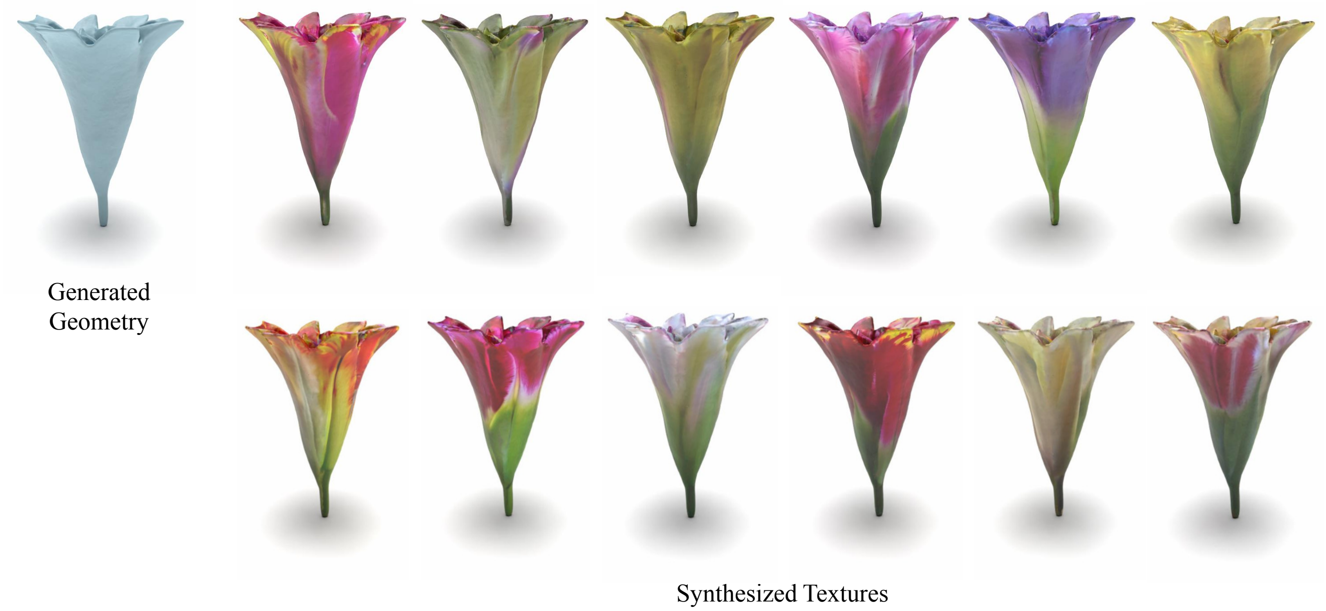


Figure 2. Diverse texture synthesis results. Based on the same generated geomtry, we could generate diverse textures by using TEXTrue [10].

2.4. Network Architecture

Variational Autoencoders (VAE). We use a custom Autoencoder architecture for our VAE. Given an input voxel grid, $\mathbf{G}_l$ at level l, and associated per-voxel attributes $\mathbf{A}_l$, we first positionally encode each voxel using the same function as [5] and then concatenate the positional encoding of each voxel with the corresponding attribute. We then apply a linear layer to the concatenated positional embedding and attribute to lift it to a d-dimensional feature (Where d is chosen depending on the task and described in Table 2). Our VAE then applies successive convolution and max pooling layers, coarsening the voxels to a bottleneck dimension. When $l = 1$ (*i.e.* the coarsest level of the hierarchy), we zero pad the bottleneck layer into a dense tensor, otherwise, the bottleneck is a sparse tensor. We then apply 4 convolutional layers to convert the bottleneck tensor into a latent tensor $\mathbf{X}$ of the same shape and sparsity pattern as the bottleneck. Latent diffusion is done over the tensor $\mathbf{X}$. At the end of the decoder, we apply attributes-specific heads (MLPs) to predict the associated attributes within each voxel. Hyperparameters for our VAEs are listed in Tab. 2.

Diffusion UNet. As mention in the main paper, we adopt a a 3D sparse variant of the backbone used in [1] for our voxel latent diffusion. Hyperparameters for training them are in Tab. 3

2.5. Training Details

We train all of our models using Adam [4] with $\beta_1 = 0.9$ and $\beta_1 = 0.999$. We use an EMA rate of 0.9999 for all experiments and use PyTorch Lightning [2] for training. For ShapeNet models, we use $8\times$ NVIDIA Tesla V100s for training. For other datasets, we use $8\times$ NVIDIA Tesla A100s for training.

2.6. Metric Definition

To perform a quantitative comparison of our generative model on the ShapeNet dataset, we leverage the framework used in [12] which uses the *1-NNA* metric defined as follows: Given a generated set of point clouds S_g, a reference set of point clouds S_r, and a metric $D(\cdot,\cdot) : 2^{\mathbb{R}^3} \times 2^{\mathbb{R}^3} \to \mathbb{R}$ between two point clouds, the 1-NNA metric is defined as

$$\text{1-NNA}(S_g, S_r) = \frac{\sum_{X\in S_g} \mathbb{1}[N_X \in S_r] + \sum_{Y\in S_r} \mathbb{1}[N_Y \in S_r]}{|S_g| + |S_r|}, \tag{3}$$

where N_A is the closest point cloud to $A \in 2^{\mathbb{R}^3}$ in the set $S_g \cup S_r - \{X\}$ under the metric $D(\cdot,\cdot)$ (*i.e.* the closest point cloud to A in the generated and reference set not including A itself), and $\mathbb{1}[\cdot]$ is the indicator function which returns 1 if the argument is true and 0 otherwise.

Intuitively, the 1-NNA distance is the classification accuracy when using nearest neighbors under D to determine if a point cloud was generated ($\in S_g$) or not ($\in S_r$). If the generated set is close in distribution to the reference set, then the classification accuracy should be around 50% which is the best 1-NNA score achievable.

In our experiment, we sampled 2048 points from the surface of each shape (following [12]) to generate S_g and S_r and used the Chamfer and Earth Mover's distances as metrics D to compute the 1-NNA.

3. More results

In this section, we provide more qualitative results on all datasets. First, we show more text-to-3D results on Objaverse in Fig. 4 to 6. Then, we show more results on ShapeNet in Fig. 3 and 7 to 9. Despite the high quality of our generated shapes, we show that our model does not overfit the training samples and is able to generate novel shapes in Fig. 3 by retrieving the most similar shapes in the training set given the generated samples. Furthermore, we show more results on Waymo in Fig. 10 and 11. Finally, we show more results on Karton City in Fig. 12.

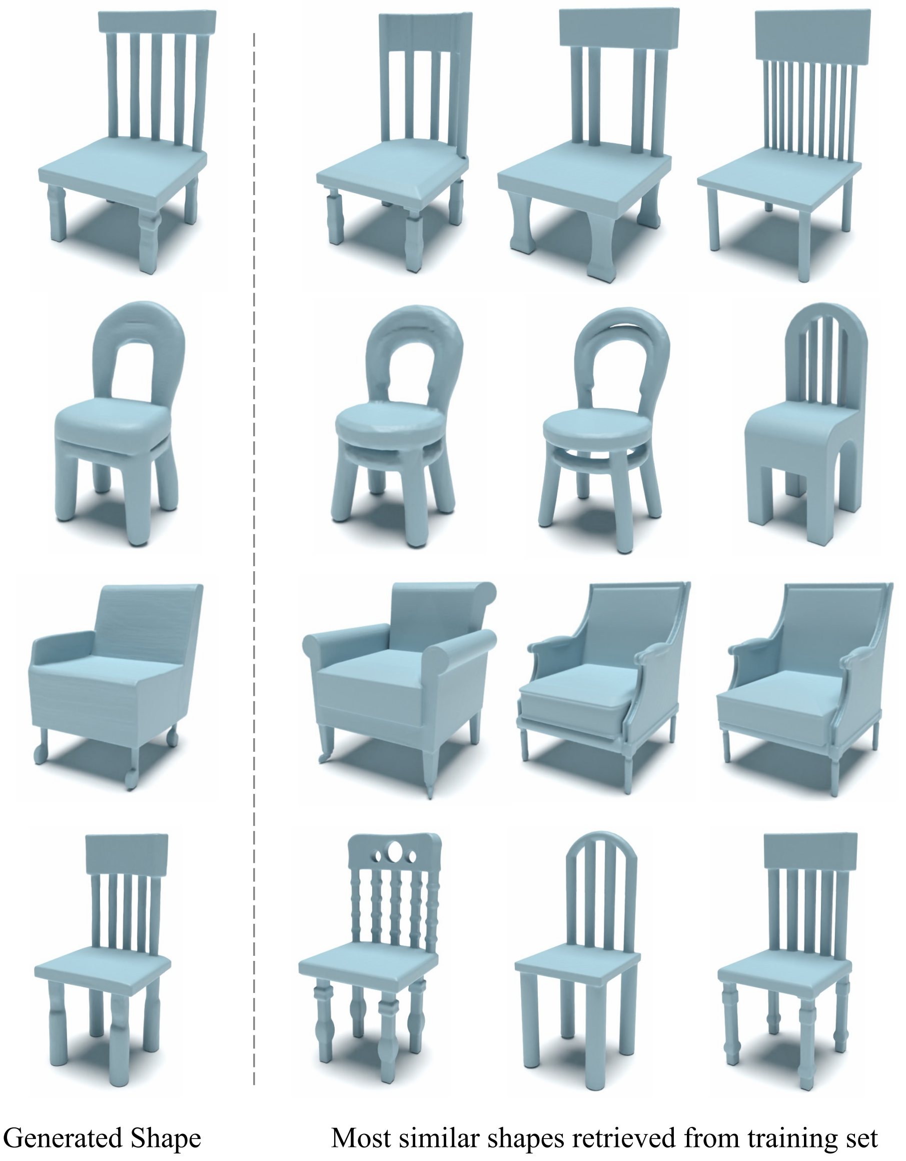
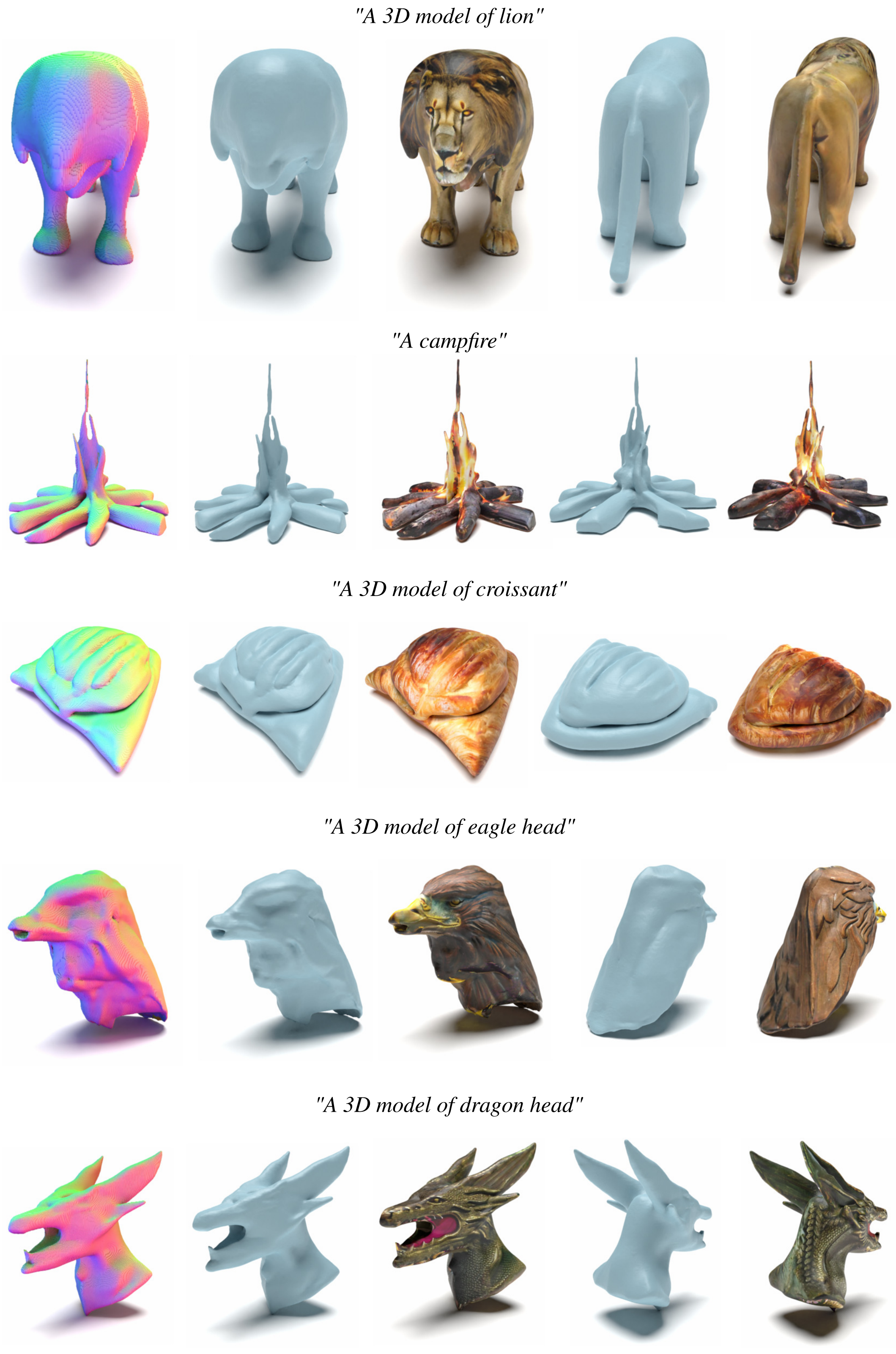


Figure 3. Shape Novelty Analysis. From our generated shape (left), we retrieve top-three most similar shapes in training set by CD distance



"A 3D model of lion"

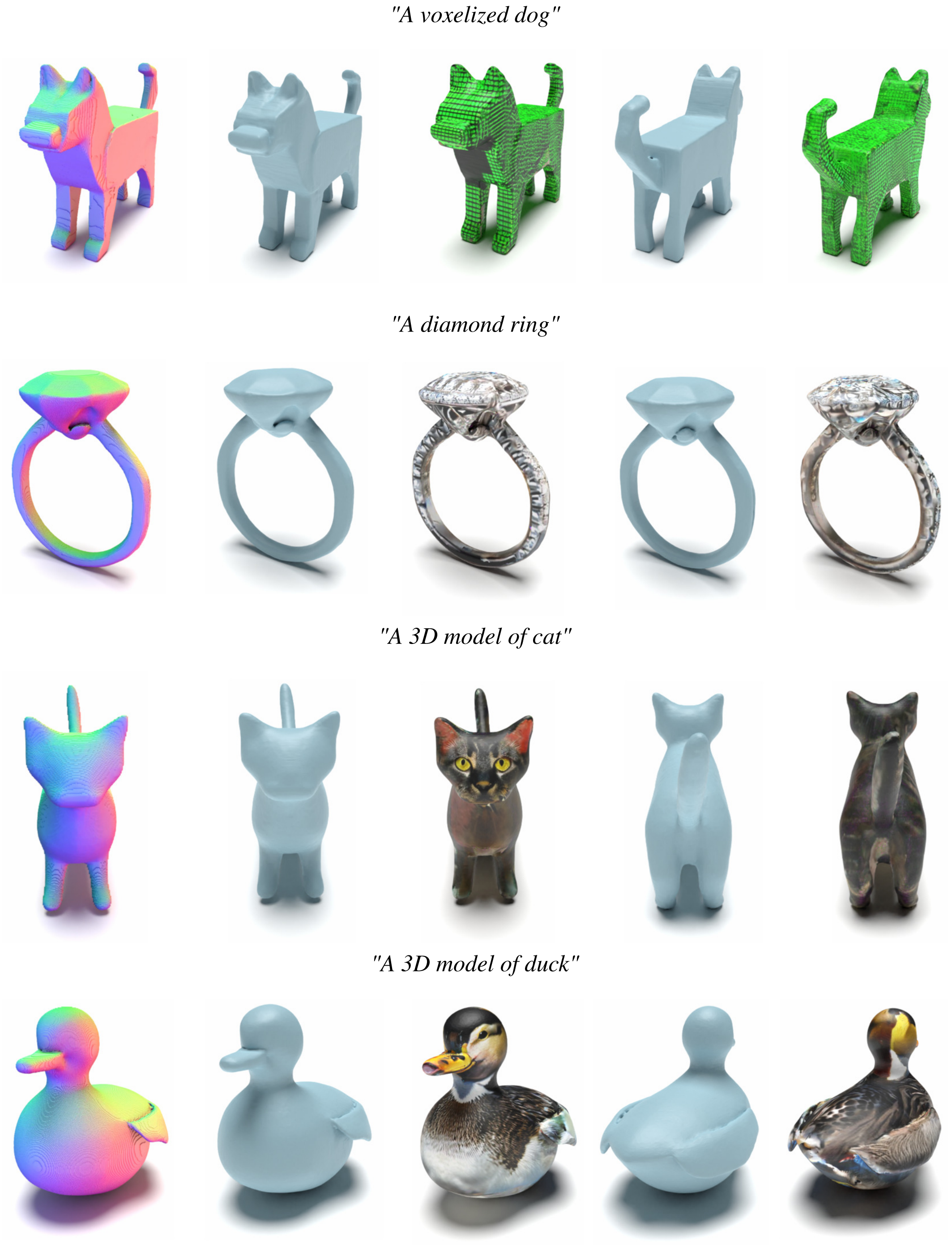
"A campfire"

"A 3D model of croissant"

"A 3D model of eagle head"

"A 3D model of dragon head"

Figure 4. More qualitative results on text-to-3D.



"A voxelized dog"

"A diamond ring"

"A 3D model of cat"

"A 3D model of duck"

Figure 5. More qualitative results on text-to-3D.

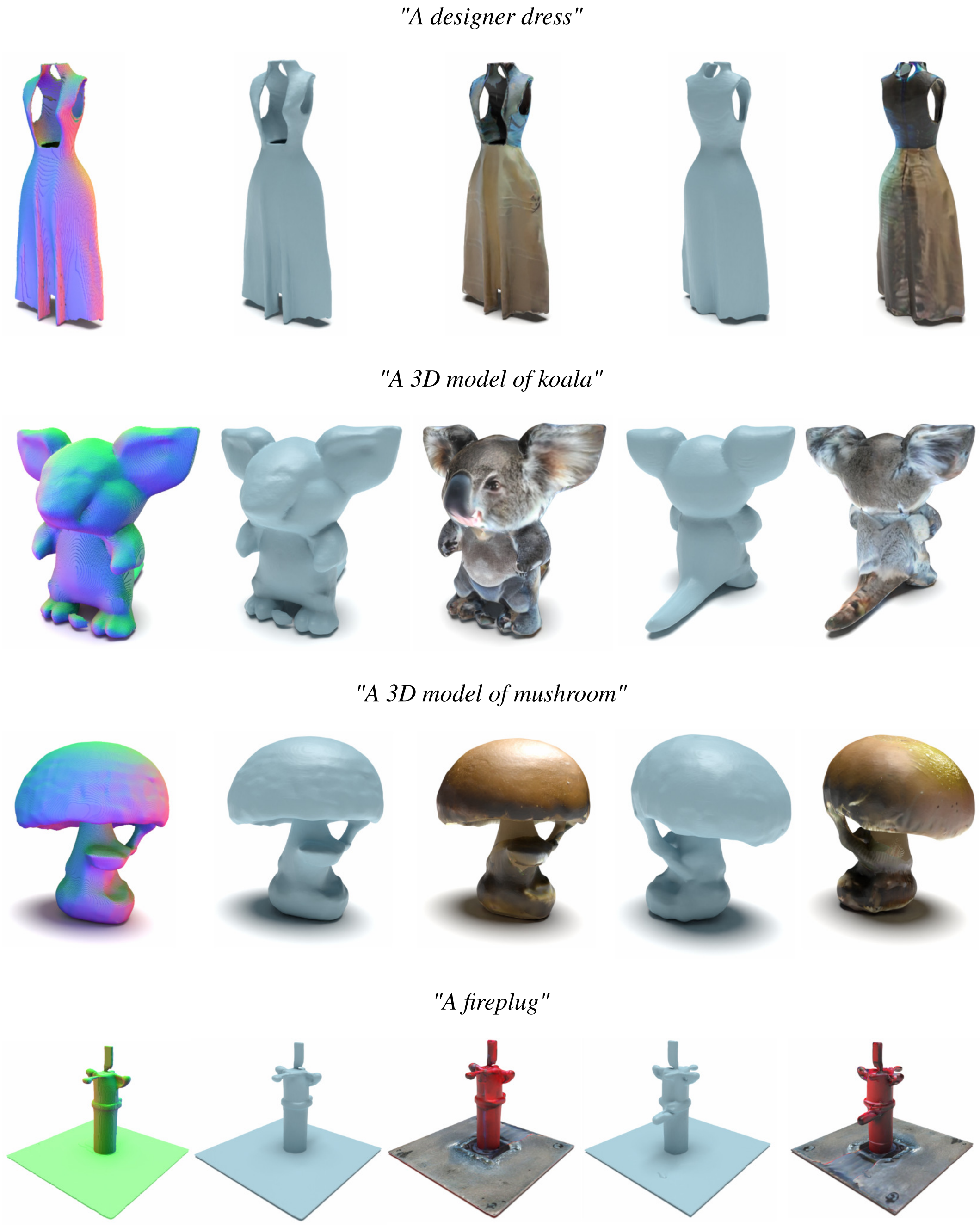


Figure 6. More qualitative results on text-to-3D.

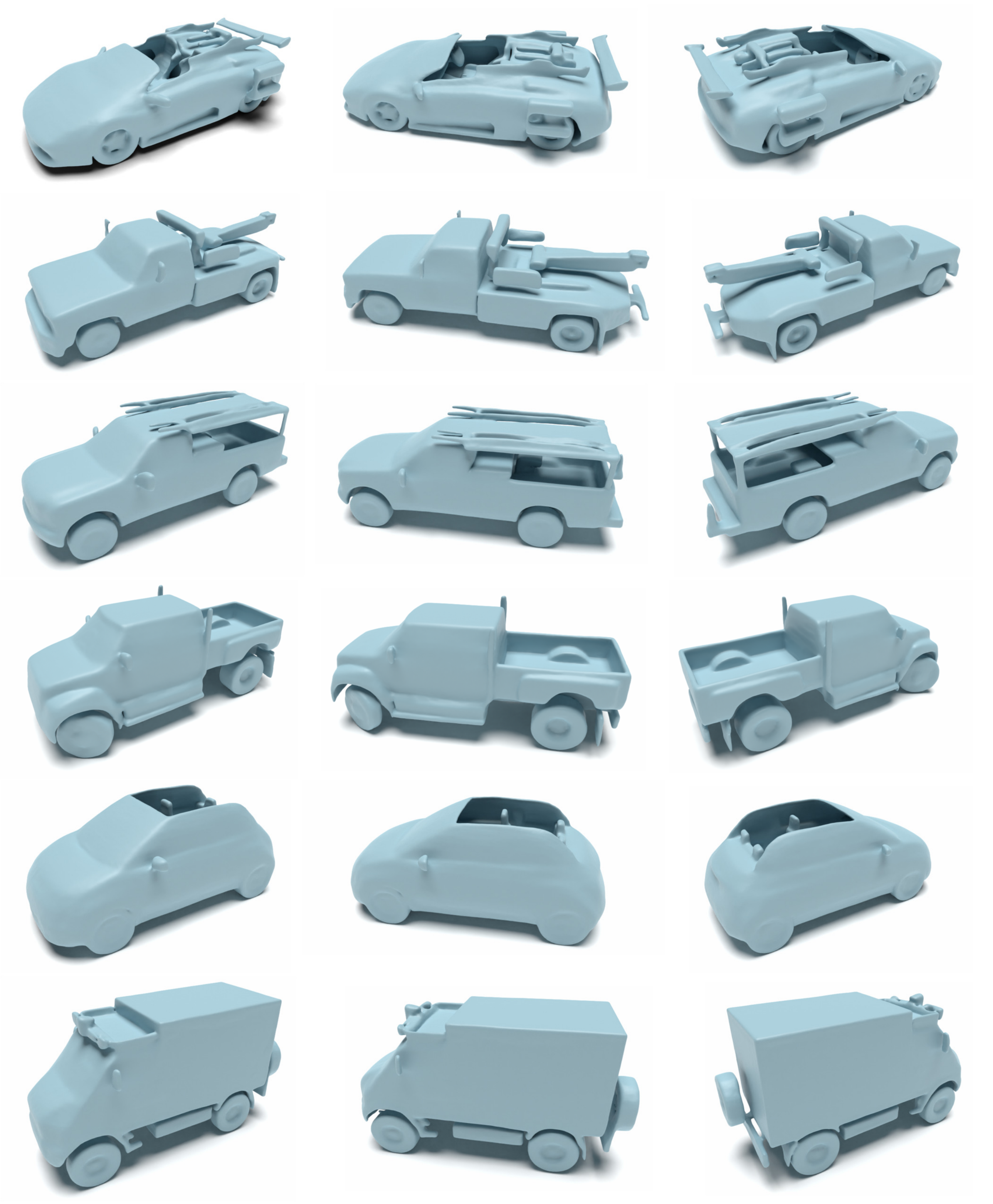


Figure 7. More qualitative results on ShapeNet Car.

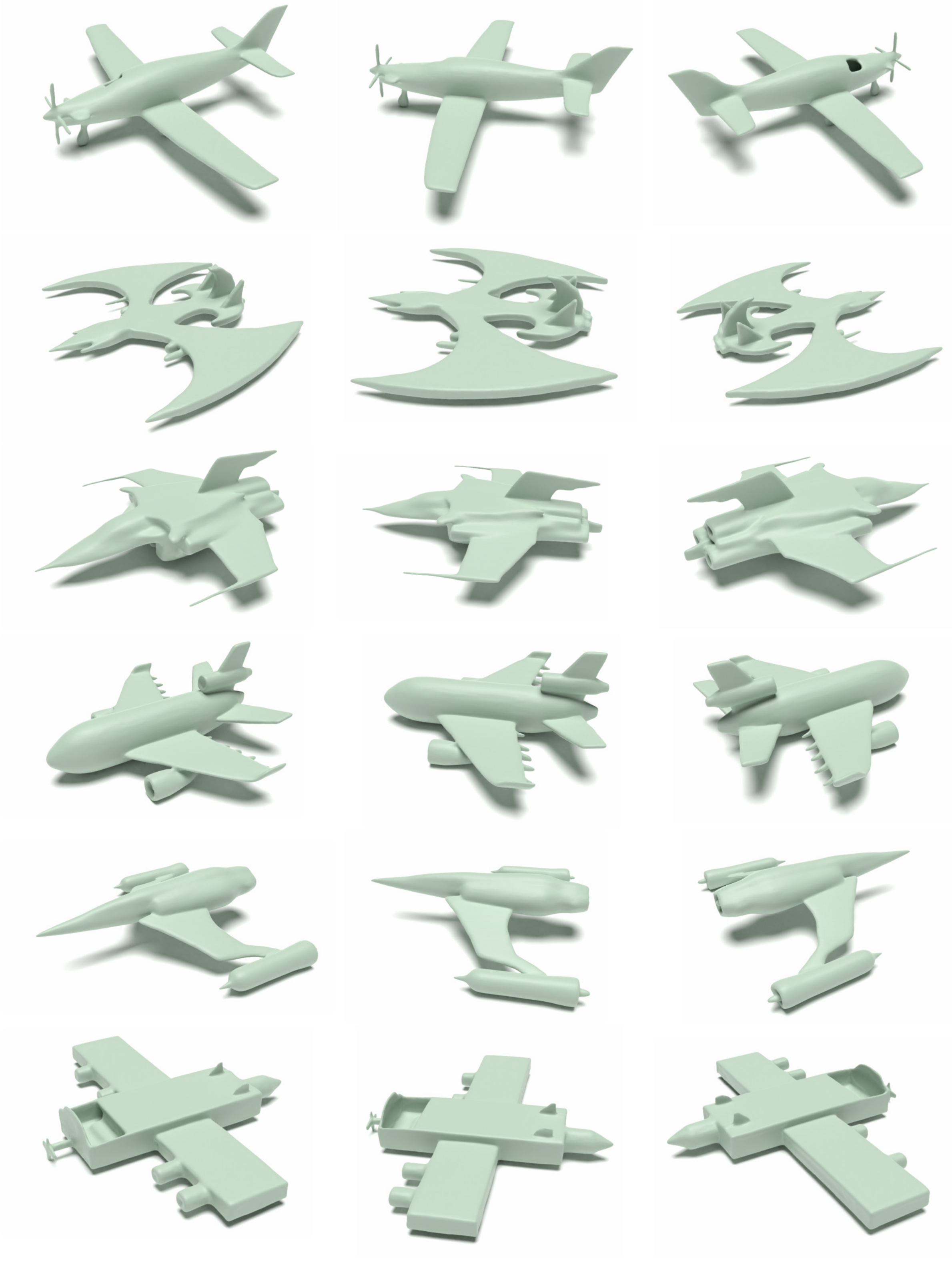


Figure 8. More qualitative results on ShapeNet Airplane.

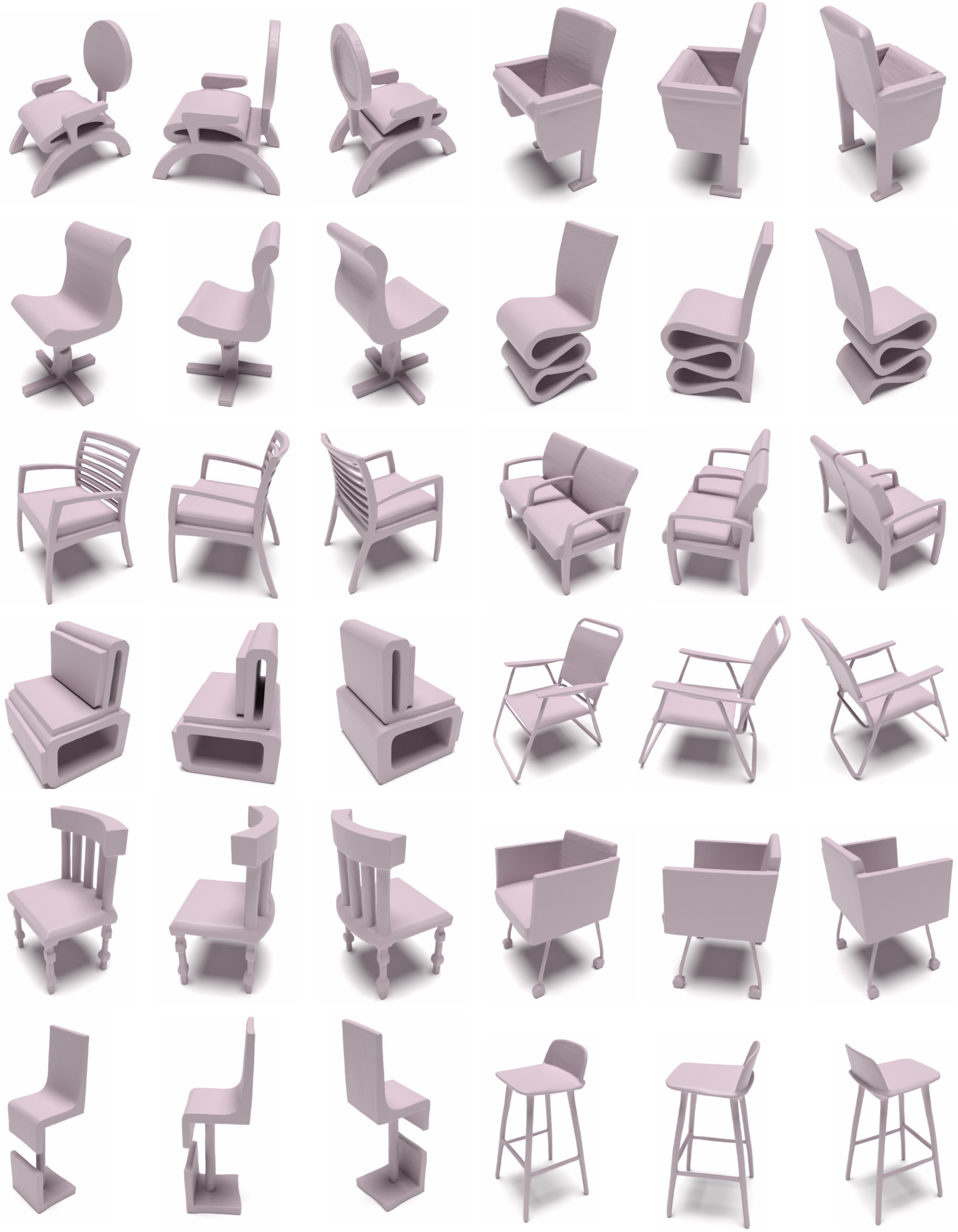


Figure 9. More qualitative results on ShapeNet Chair.

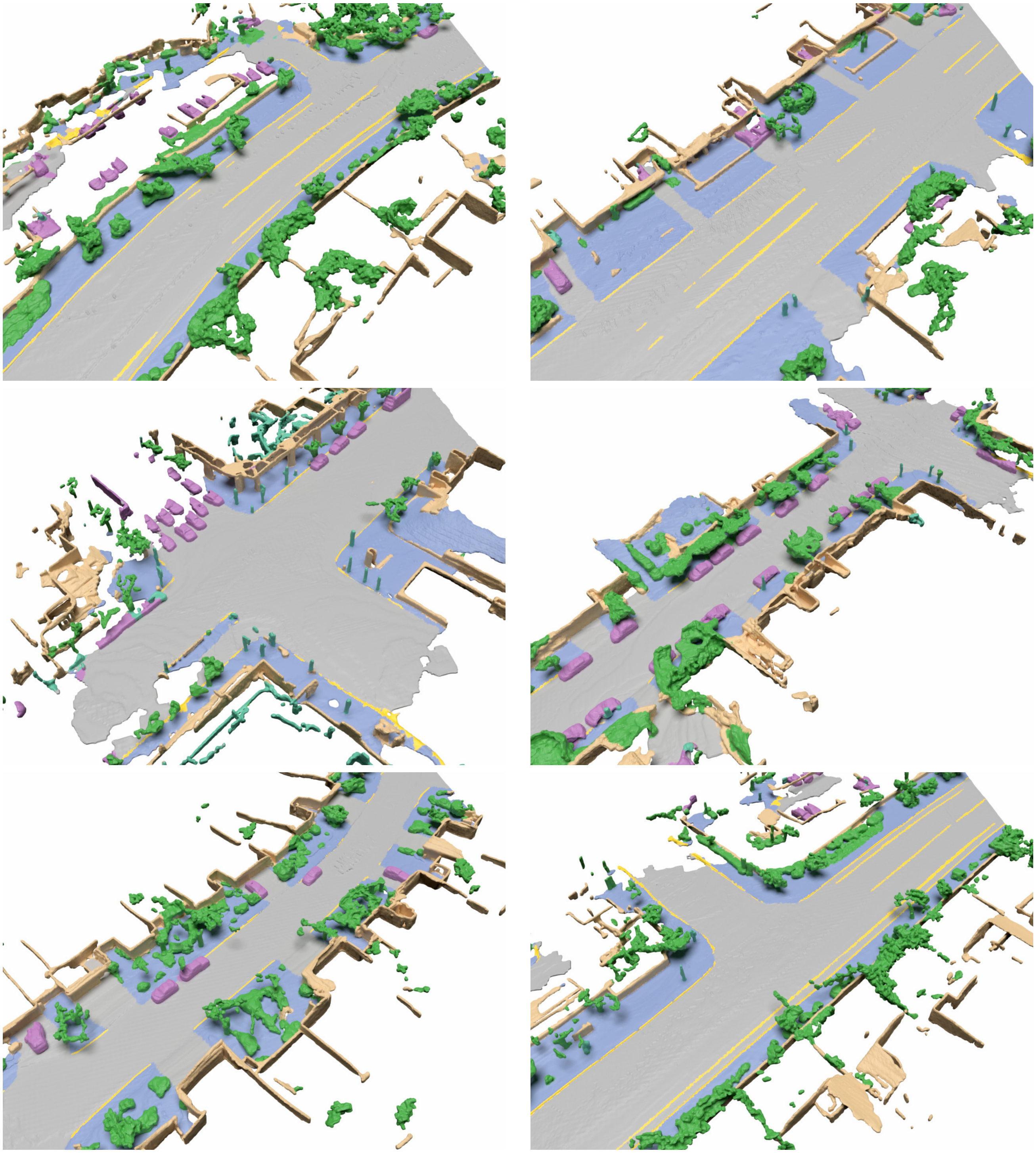


Figure 10. More qualitative results on Waymo.

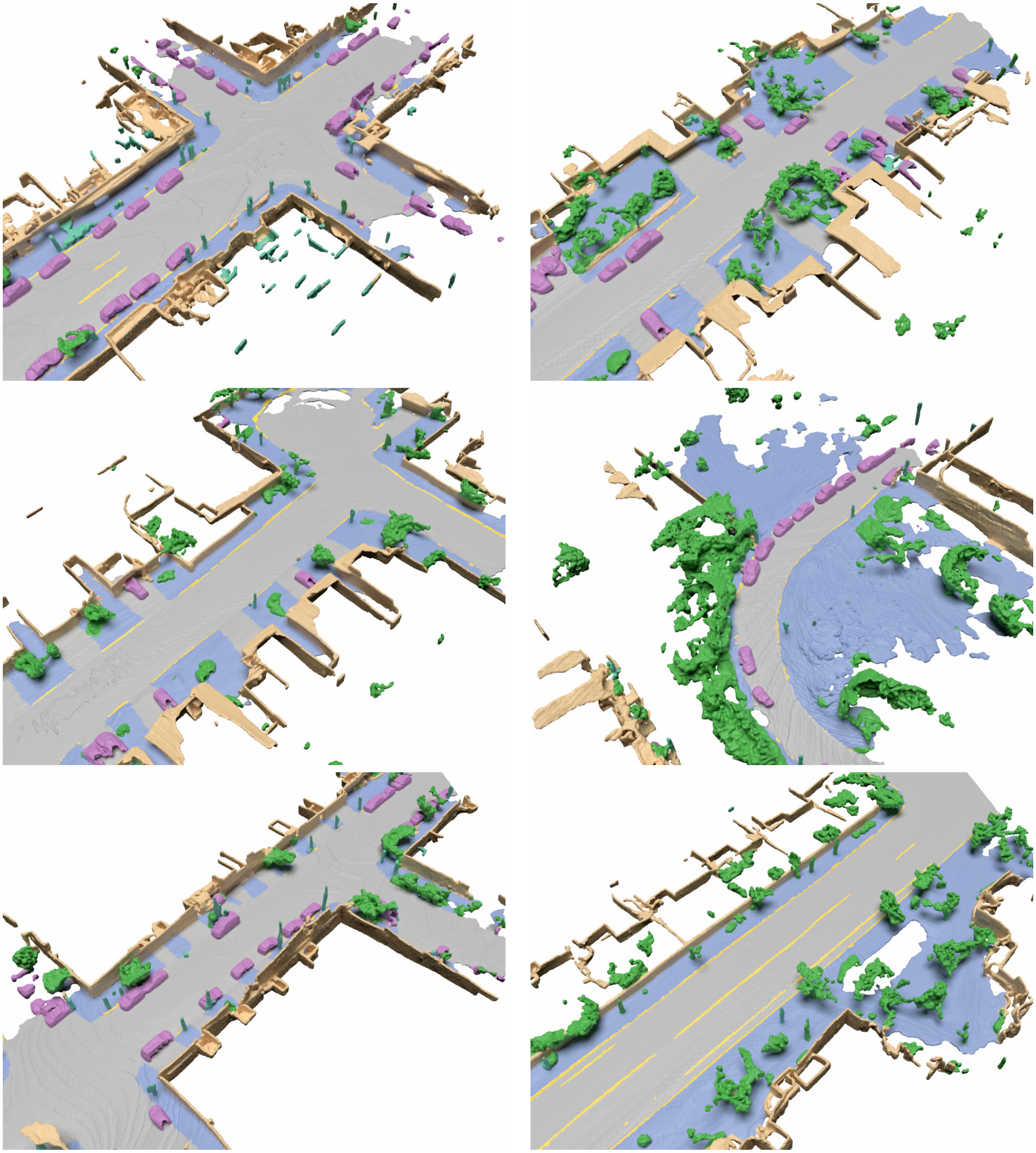


Figure 11. More qualitative results on Waymo.

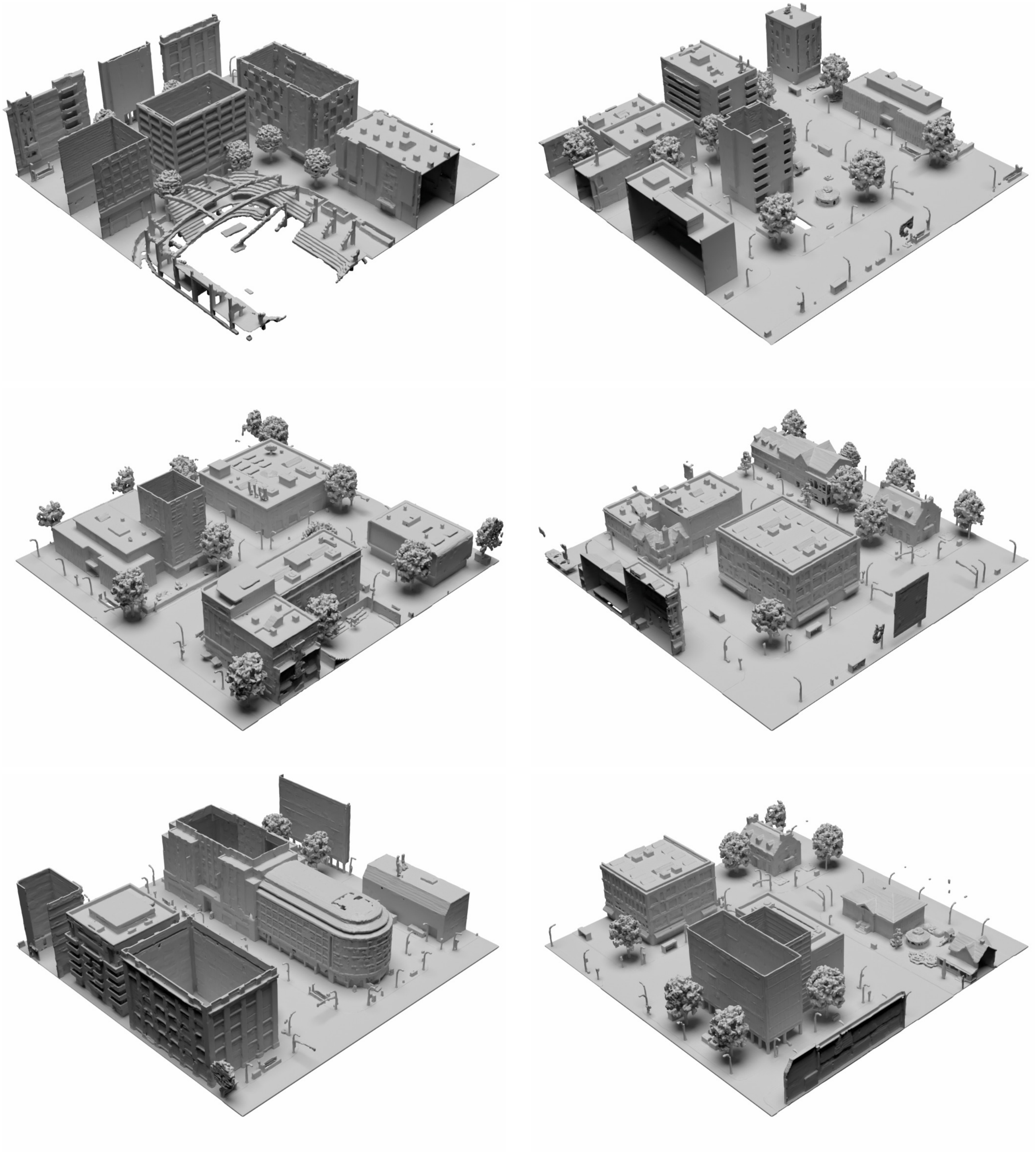


Figure 12. More qualitative results on Karton City.

	ShapeNet	ShapeNet	Objaverse	Objaverse	Waymo	Waymo
	$16^3 \rightarrow 128^3$	$128^3 \rightarrow 512^3$	$16^3 \rightarrow 128^3$	$128^3 \rightarrow 512^3$	$32^3 \rightarrow 256^3$	$256^3 \rightarrow 1024^3$
Model Size	59.6M	3.8M	236M	14.9M	59.4M	3.8M
Base Channels	64	32	128	64	64	32
Channels Multiple	1,2,4,8	1,2,4	1,2,4,8	1,2,4	1,2,4,8	1,2,4
Latent Dim	16	8	16	8	16	8
Batch Size	16	32	32	32	32	32
Epochs	100	100	25	10	50	50
Learning Rate	1e-4					

Table 2. **Hyperparameters for VAE.** For the Karton City dataset, we used the same hyperparameters as the Waymo dataset.

	ShapeNet - 16^3	ShapeNet - 128^3	Objaverse - 16^3	Objaverse - 128^3	Waymo - 32^3	Waymo - 256^3
Diffusion Steps	1000					
Noise Schedule	linear					
Model Size	691M	79.6M	1.5B	79.6M	702M	76.6M
Base Channels	192	64	256	64	192	64
Depth	2					
Channels Multiple	1,2,4,4	1,2,2,4	1,2,4,4	1,2,2,4	1,2,4,4	1,2,2,4
Heads	8					
Attention Resolution	16,8,4	32,16	16,8,4	32,16	16,8	32
Dropout	0.1	0.0	0.0	0.0	0.0	0.0
Batch Size	256	256	512	128	512	256
Iterations	varies*	15K	95K	80K	40K	20K
Learning Rate	5e-5					

Table 3. **Hyperparameters for voxel latent diffusion models.** *We train our model with 25K iterations for ShapeNet Airplane, 45K iterations for ShapeNet Car, and 35K iterations for ShapeNet Chair. For the Karton City dataset, we used the same hyperparameters as the Waymo dataset and trained the models to converge.

References

[1] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems*, pages 8780–8794, 2021. 3

[2] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. 3

[3] Jiahui Huang, Zan Gojcic, Matan Atzmon, Or Litany, Sanja Fidler, and Francis Williams. Neural kernel surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4369–4379, 2023. 1, 2

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

[5] 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. 3

[6] Ken Museth. VDB: high-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics (TOG)*, 32(3):27:1–27:22, 2013. 1

[7] Ken Museth. Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*, New York, NY, USA, 2021. Association for Computing Machinery. 1

[8] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, pages 523–540, 2020. 1, 2

[9] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 2

[10] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes. In *SIGGRAPH*, 2023. 2, 3

[11] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. Torchsparse: Efficient point cloud inference engine. *MLSys*, 2022. 1

[12] 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 *Advances in Neural Information Processing Systems*, 2022. 3, 4