Representing 3D Shapes with 64 Latent Vectors for 3D Diffusion Models

Supplementary Material

	Sampling	Decoding	Full
VecSet ($M = 512$)	2.40	2.10	1.12
Ours ($M = 64$)	20.49	56.44	15.03
Ours ($M = 32$)	37.42	58.22	22.78
VecSet ($M = 512$) with MR	2.40	10.61	1.94
Ours ($M = 64$) with MR	20.67	162.85	18.34
Ours ($M = 32$) with MR	37.70	174.88	31.00

Table 8. **Generation throughput (sample/s) at each step.** 'Sampling' denotes the latent generation, 'Decoding' denotes the latent-to-fields decoding process, including both the decoder and the neural fields decoding, and 'Full' denotes the entire generation process. 'MR' denotes the multi-resolution query points sampling technique. The throughput is measured using a single RTX A6000 GPU with a batch size 64.

A. Additional discussion

A.1. Improved query points sampling

As discussed in the main paper, we follow VecSet [17] and adopt a naive query point sampling approach, which employs a single-resolution test grid and evaluates all points within it. This results in an enormous number of query point evaluations. While our triplane-based neural field decoding significantly reduces computational overhead in the query decoding process, the excessive number of query points often hinders maximizing the generation throughput.

The overall throughput can be further accelerated by adopting an improved query points sampling technique. We can employ a simple multi-resolution query point sampling strategy, similar to the method proposed in [8]. For instance, in the case of a 128^3 test grid, we first use coarse-level points from a 64^3 grid to identify occupied and empty regions. Based on this coarse-level evaluation, we then evaluate finer-level points (128^3) only within occupied voxels.

As shown in Table 8, this strategy reduces unnecessary computations, leading to further acceleration in the neural field decoding process. We can boost up our model with $M = 32$ by alleviating the bottleneck in the neural fields decoding process, thereby maximizing its throughput. While the same sampling technique can also be applied to VecSet ($M = 512$), its computationally intensive sampling procedures impose a substantial overhead, limiting the efficiency gains in the generation process. Notably, this method can be implemented as batch-wise computations by using an appropriate zero-padding.

A.2. Reducing latent vectors and generation

As shown in Tab. 3 in the main paper, compact latent vectors not only improve efficiency but also lead to better generation performance in all metrics except MMD. We attribute this to the fact that a smaller total latent size simplifies the training of diffusion models, thereby enhancing overall generation quality. Specifically, employing more latent vectors allows each vector to cover a smaller region, resulting in higher fidelity (MMD) of VecSet ($M = 512$). However, this comes at a cost of a larger total latent size, making diffusion models more challenging to model and resulting in degraded performance on all other metrics. This results in degraded performance of VecSet with $M = 512$ across all other metrics. While more latent vectors may offer better higher fidelity, they also impose a greater burden on the diffusion model to capture the full distribution. Similarly, recent diffusion models also employ VAEs for latents with a small channel dimension to make the total latent size small.

	Speed (iter/s)↑	Memory (GB)↓	Params.↓
VecSet ($M = 512$)	2.98	13.40	113M
Ours ($M = 64$)	3.02	15.64	187M
Ours ($M = 32$)	3.18	14.81	187M

Table 9. **Training efficiency analysis.** We analyze the efficiency of VAEs on ShapeNet, without the two-stage training strategy for a clear comparison.

A.3. Future extension of COD-VAE

Integration with VecSet-based methods. Recently, many feed-forward 3D diffusion models have extended the VecSet pipeline with various improvements. GaussianAnything [14] introduces a Gaussian-based decoder and adopts FPS-based 1D latent vectors, similar to those used in VecSet. Dora [3] addresses the issue of uniform point sampling in VecSet by proposing a sharp-edge sampling strategy, which better captures regions with complex geometry. Since COD-VAE also builds upon the VecSet framework, we believe our model can naturally benefit from these recent advances.

Towards textured mesh reconstruction. While this work follows the experimental setups of VecSet and only reconstructs shapes of the objects, extending our model to generate textured meshes can be an interesting future direction. This can possibly be accomplished by employing multi-view diffusion models to generate materials and textures [18], training VAE with an additional rendering loss [7, 14], or employing Gaussian-based VAE decoder [14].

B. VAE efficiency analysis

We present the training efficiency analysis of VAEs in Table 9. While our model with $M = 64$ achieves a compara-

	Encoder	Decoder	Fields decoding
VecSet ($M = 512$)	0.02	0.27	13.60
Ours ($M = 64$)	0.30	0.10	0.46
Ours ($M = 32$)	0.30	0.08	0.46

Table 10. **Runtime breakdown of VAEs.** We report the latency (s) with a batch size 64.

ble training speed to VecSet, it requires slightly more GPU memory ($1.16\times$). Additionally, the total parameter count increases by a factor of 1.65. These increases in memory usage and parameter count can be viewed as the trade-offs for achieving improved generative efficiency. Although VAEs are less frequently trained compared to generative models in practice, we believe that reducing these costs remains an interesting direction for future research.

We also provide a runtime breakdown of the inference of VAEs in Table 10. Our models demonstrate greater efficiency in the decoder and the neural fields decoding. However, the computational cost of the encoder is higher compared to VecSet, as VecSet employs a lightweight encoder consisting of a single cross-attention layer. Considering that the forward computation of the encoder is usually included in the generative model training, reducing the computations of the encoder can further accelerate the training of generative models, which we leave for future works.

C. Method details

In this section, we provide further details of our COD-VAE architecture design. We provide details of each component in the following paragraphs.

Encoder. For the initial positions of the compact vectors $\mathcal{F}$, we follow VecSet [17] and adopt input-dependent positions, sampled from the input point cloud. We note that our method can also incorporate learnable positional embeddings, similar to VecSet. To encode point features, as well as the positions of point patches and compact vectors, we utilize a shared positional embedding function. The positional embedding function follows the design used in VecSet, which comprises learnable weights and a linear layer. When processing point patches with self-attention layers, we introduce an additional `CLS` token, following [13], to aggregate global information. After the last encoder block, we additionally apply one cross-attention layer to further project high-resolution features into compact latent vectors. We use the KL block design commonly used in 2D generative models, which is also employed in VecSet.

Decoder. After pruning, the remaining tokens are processed using ViT-style transformer blocks. In this stage, we aggregate the pruned tokens into 8 merged tokens via cross-attention and concatenate them with the input sequence to leverage additional information effectively. To reconstruct the full triplanes, we first linearly project the initial triplane tokens into dense triplane features. The processed tokens are then linearly projected and scattered into the dense triplanes. Finally, we apply a sigmoid activation to the uncertainty values to compute importance scores, which are multiplied with the scattered triplane features. The final triplane features are obtained by adding these weighted features to the initial dense triplane features.

D. Training objective

First stage. In the first stage training, we train the autoencoder model to minimize the reconstruction loss. Similar to VecSet, the reconstruction loss computed as binary cross entropy between the final occupancy predictions and the ground truth:

$$\begin{aligned}\tilde{\mathcal{L}}_{recon} = \mathbb{E}_{\mathbf{q}\in\mathcal{Q}_{vol}}\left[\text{BCE}(o(\mathbf{q}), \hat{o}(\mathbf{q}))\right] \\ + 0.1 \cdot \mathbb{E}_{\mathbf{q}\in\mathcal{Q}_{near}}\left[\text{BCE}(o(\mathbf{q}), \hat{o}(\mathbf{q}))\right],\end{aligned} \tag{8}$$

where $o(\mathbf{q})$ is the predicted occupancy using the final triplane features, and $\hat{o}(\mathbf{q})$ is the corresponding ground truth occupancy. The training objective of the first stage can be expressed as

$$\mathcal{L}_{ae} = \tilde{\mathcal{L}}_{recon} + \mathcal{L}_{recon} + \lambda_{unc} \cdot \mathcal{L}_{unc}, \tag{9}$$

where $\mathcal{L}_{recon}$ is the reconstruction loss computed using the initial triplane tokens, and $\mathcal{L}_{unc}$ is the loss for training the uncertainty head. Both losses follow the procedure described in the main paper. To properly train the uncertainty head, we clip the initial query-wise reconstruction loss $\mathcal{L}_{recon}(\mathbf{q})$, computed using the initial triplane features, to be in a range $[0, 1]$.

Second stage. In the second stage VAE training, we freeze the autoencoder components and train the KL block and the latent decoder. The training objective of the second stage consists of the MSE loss between the features with two regularization terms:

$$\mathcal{L}_{vae} = \text{MSE}(n(\hat{\mathcal{F}}), n(\mathcal{F})) + \mathcal{L}_{ae} + \lambda_{kl} \cdot \mathcal{L}_{kl}, \tag{10}$$

where $n(\cdot)$ is a layer-wise normalization [1] without affine operations. This normalization can be applied since our decoder only consists of transformer layers, which normalize the input features before processing. Note that only the KL block and the latent decoder are trained in the second stage.

E. Experimental setup details

E.1. Additional implementation details

We set the channel dimension of the transformer 512 with 8 heads in all components. Our models and training pipelines

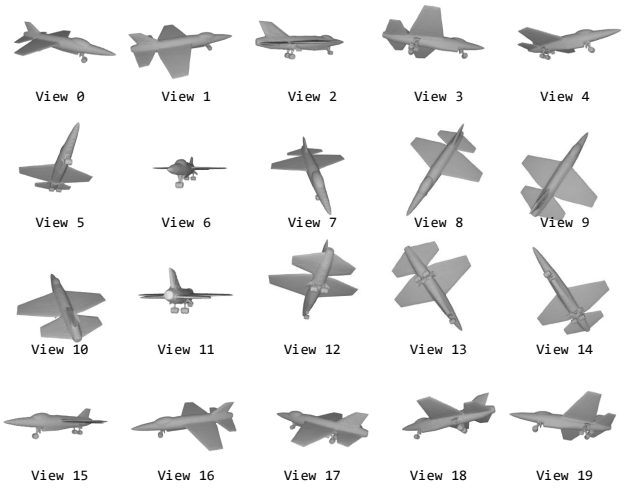

Figure 9. **Example visualizations of 20 rendered images**, used for computing Rendering-FID. The image resolution is 299×299.

are implemented using the PyTorch framework. For the ShapeNet experiments, we use the AdamW optimizer with a learning rate 1e-4, weight decay 0.01, and the effective batch size 256 to train the autoencoders and 1024 to train the VAEs. In the second stage, the learning rate is decayed by 0.5 after 960, 1120, 1280, 1440 epochs. For the Objaverse experiments, we use the same configurations for the first stage, and we decay the learning rate after 120, 140, 160, 180 epochs. We set $\lambda_{unc} = 0.01$ and $\lambda_{kl} = 0.001$. We use the automatic mixed precision (AMP) and the flash attention [5] built in for the PyTorch framework, which are disable when measuring the efficiency of our method for precise evaluations. While we do not manually align orientations of objects and do not apply rotation augmentations, our model could be improved by applying random rotation augmentations – particularly on the Objaverse dataset, which includes objects with inconsistent orientations.

E.2. Evaluation protocols

In our experiments, we follow the commonly used evaluation protocols from SDF-StyleGAN [19] and VecSet [17] as described in the main paper.

For reconstruction experiments, we measure Chamfer Distance (CD), volumetric Intersection-over-Union (IoU) and F-score (F1) computed using the chamfer distance. We use the threshold 0.02 to compute F1 for ShapeNet, while we slightly increase the threshold to 0.05 for Objaverse, as the Objaverse dataset comprises more complicated objects.

To evaluate the generation performance, we employ the FID-based scores as our main metrics, which are Rendering-FID and Surface-FPD. We note that these FID-based metrics are wide in more recent research [12, 15–17, 19], as it can better take human perception into consideration [19]. For Rendering-FID, we first normalize the surface mesh to a unit sphere and then render shading images from 20 uniformly distributed views (see Figure 9). To measure Surface-FPD, we sample 4,096 points from the mesh surface and then feed them into pretrained PointNet++ [9] to extract global feature vectors. This PointNet++ network is pretrained on ShapeNet-v2 for shape classification. using the same train/valid/test split as in our experiments.

We also measure additional metrics, MMD, COV, 1-NNA, computed using Chamfer distance (CD) and Earth Mover's distance (EMD) [10]. To evaluate generation performance, we use Coverage (COV), Minimum Matching Distance (MMD), 1-Nearest Neighbor Accuracy (1-NNA). To compute these metrics, we compute the pairwise distances between the generated set S_g and the reference set S_r. We compute COV and MMD by using the test test as S_r, and generate $5|S_r|$ shapes as S_g. To compute 1-NNA, we set $|S_g| = |S_r|$. For these metrics, we sample 2,048 points from the surface of each mesh.

For category-conditioned generation, we generate 2,000 objects per category for evaluation. The test set distribution is as follows: airplane (202), car (175), chair (338), table (421), rifle (118). For COV and MMD, we sample from the generated objects per category: airplane (1,010), car (875), chair (1,690), table (2,000), and rifle (590). For Rendering-FID and Surface-FPD, 2000 objects are sampled from the training set and compared with the generated objects.

To evaluate efficiency, we measure throughput on a single A6000 GPU. The batch size is adjusted to maximize GPU utilization, based on the model with the highest GPU memory consumption.

Method	Freeze	IoU (%)↑	CD↓	F1 (%)↑
PerceiverIO (FPS query)	✓	91.7	0.018	93.4
PerceiverIO (learnable query)	✓	92.5	0.017	94.6
PerceiverIO (learnable query)		94.0	0.015	96.1
Ours ($M = 32$)		96.1	0.012	98.0

Table 11. **Results of PerceiverIO** with $M = 32$ and pretrained VecSet ($M = 512$) on ShapeNet. We also report variants with frozen VecSet, and with learnable queries replaced by FPS as used in VecSet. Results of autoencoders are reported.

F. Additional experiments and results

F.1. Comparison with PerceiverIO

To validate the compression capability of our method, we compare it with PerceiverIO [6], which is trained to compress the original latent space of VecSet. Specifically, PerceiverIO takes the latent vectors from the decoder of a pretrained VecSet as input and outputs compressed latent vectors. The remaining architecture of this model (*i.e.*, decoder, neural fields decoding) follows the VecSet pipeline. We also evaluate several variants: one with the frozen VecSet, and another that uses FPS-based queries instead of learnable

VecSet	$M = 32$	$M = 64$	$M = 512$	Ours ($M = 64$)
Surface-IoU (%)↑	72.7	78.0	87.8	88.0

Pruning ratio	0%	50%	75%	90%
Surface-IoU (%)↑	87.9	87.8	87.6	87.1

Table 12. **Near-surface reconstruction results on ShapeNet.** To better assess the quality of reconstructed surfaces, we measure IoU using 50K points sampled near object surfaces. **(top)** VAE comparisons with VecSet. **(bottom)** Ablation results on the pruning ratio, obtained using autoencoders with $M = 32$.

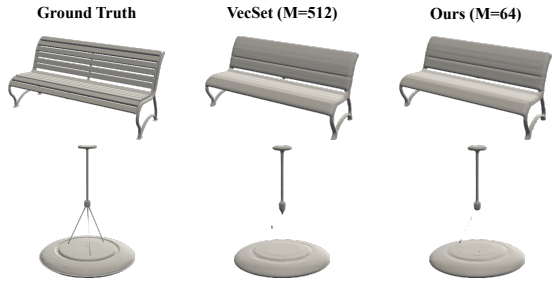

Figure 10. **Failure cases** of our model and VecSet ($M = 512$). We report the results of the autoencoders.

queries, following the design of VecSet.

As shown in Table 11, our model outperforms several variants of PerceiverIO by a large margin. These results indicate that existing latent compression methods, primarily explored in the 2D computer vision and NLP domains, require careful considerations in architecture design to effectively process 3D modality.

F.2. Near-surface reconstruction results

We additionally provide near-surface reconstruction IoU (Surface IoU) to better assess the quality of reconstructed surfaces. As reported in Table 12, our VAE with $M = 64$ achieves comparable performance in near-surface regions to VecSet with $M = 512$. In addition, while pruning up to 75% results in only a minor drop, we observe a more noticeable degradation in shape details beyond this ratio. Based on this trade-off, we set the pruning ratio to 75% to balance efficiency and quality (see Tab. 6 of the main paper for efficiency gains).

F.3. Failure cases

Since our model employs a training strategy similar with VecSet, it shares similar failure cases with VecSet using 512 latent vectors. As presented in Figure 10, both our model and VecSet struggles to model extremely thin structures of the objects, often producing over-smooth surfaces. This can be addressed by improving the points sampling strategy for both query points and point patches, such as sharp edge sampling proposed in [3].

	IoU (%)↑	CD↓	F1 (%)↑
Ours ($M = 16$)	95.6	0.013	97.8
Ours ($M = 32$)	96.1	0.012	98.0
Ours ($M = 64$)	96.5	0.012	98.2
Ours ($M = 128$)	96.7	0.012	98.2
Learnable positions	96.0	0.012	98.0
Input-dependent positions	96.1	0.012	98.0
Single-stage training	94.4	0.014	97.1
Two-stage training	96.1	0.012	98.0
Confidence [11] based uncertainty	95.3	0.015	97.1
Recon. based uncertainty	96.1	0.012	98.0

Table 13. **Additional ablation results on ShapeNet**. Top-to-bottom: ablation results on the number of latent vectors, results with the learnable latent positions, results without two-stage training, and results with the confidence-based training objective for the uncertainty head.

F.4. Text-conditioned generation

We further evaluate the generation performance of our model on text-conditioned 3D object generation. To enhance quality, we train the VAE with 128 latent vectors, incorporate two additional layers before the uncertainty-guided pruning module in the decoder, and add a convolutional refinement layer after triplane reconstruction. We also filter out low-quality objects (those with too few occupied points) from the training dataset. Following [14], we use the captions provided in [4] and extract text embeddings using CLIP. The diffusion model is trained on approximately 20K objects, and evaluation is performed on unseen captions randomly selected from the validation set.

As shown in Figure 11, our model generates high-quality objects using only 128 latent vectors, while existing methods typically require significantly more. Note that the compared methods are trained with different dataset sizes and are designed to generate colored meshes, making direct comparisons difficult. Our model may also benefit from larger-scale training to improve prompt-following ability. Nevertheless, we highlight that our approach can achieve high-quality generation with greater efficiency.

F.5. Additional ablation study

We further explore the behaviors of our method through the additional ablation studies. Consistent with the main paper, all ablations are conducted using the autoencoder with $M = 32$ on ShapeNet [2].

The number of latent vectors. As shown in Table 13 (first group), the performance of our method improves as the number of latent vectors increases. However, the performance gain diminishes beyond a certain number of latent vectors. This suggests that most of the essential information can be effectively encoded with $M = 32$ or $M = 64$ latent vectors, while additional vectors contribute only marginal improvements beyond this point.

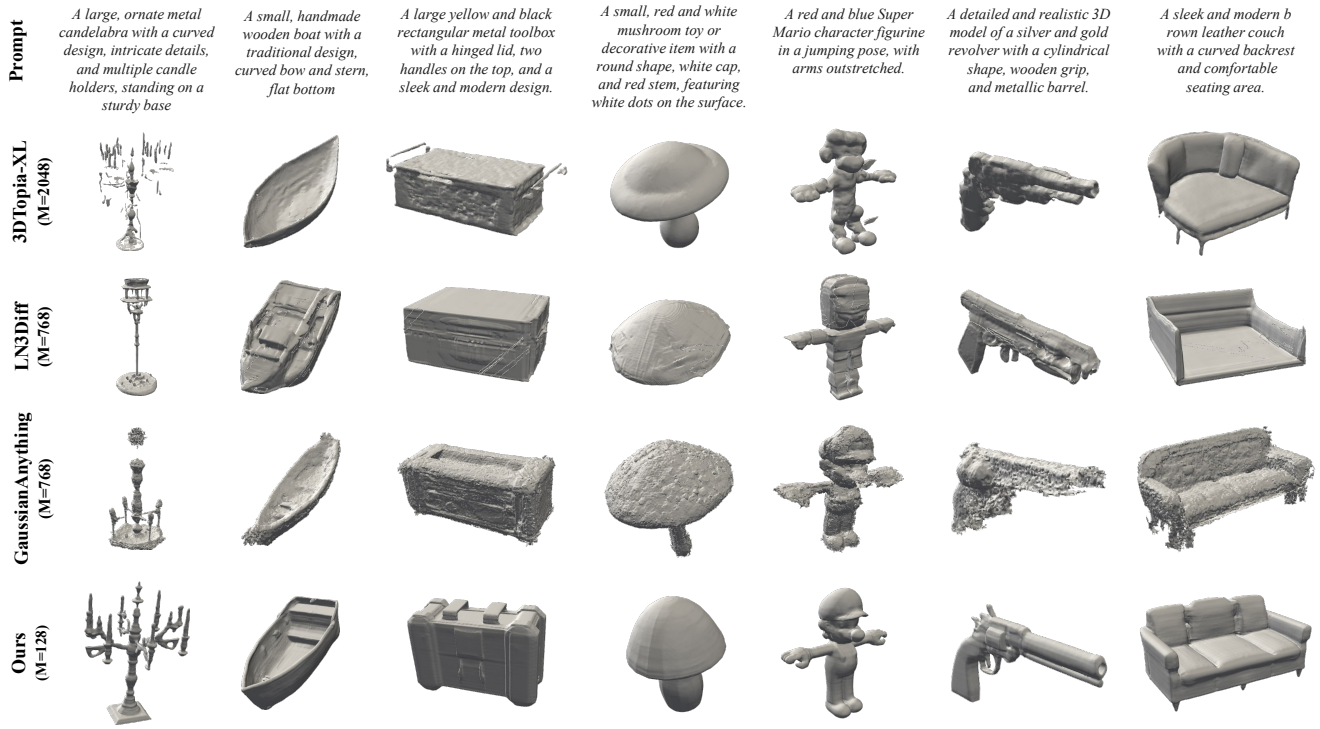

Figure 11. **Text-conditioned generation results.** Results of competing methods are obtained using their official source codes and pre-trained weights. We report mesh outputs of these methods without textures and colors.

Positions of the latent vectors. Our method can also employ learnable positional embeddings for the latent vectors. As reported in Table 13 (second group), the model with learnable latent positions achieves similar reconstruction quality with the model with input-dependent positions. Since the learnable latent positions can provide orders of the latent vectors, they can also be a useful option for the cases where we need to model the latent vectors as a ordered sequence, *e.g.*, autoregressive generation.

Two-stage training. We evaluate the two-stage training scheme as presented in Table 13 (third group). The two-stage training effectively improves the overall performance. In contrast, single-stage training makes autoencoders learn to compress both the number of latent vectors and along the channel dimension. Additionally, training uncertainty head with an auxiliary loss further complicates the training. These complicated training objectives lead to a noticeable performance drop. Therefore, we separate the autoencoder training from the training of channel compression modules.

Uncertainty head objective. Finally, we replace the training objective of the uncertainty head with the confidence-based objective used in [11]. As shown in Table 13 (fourth group), the model trained to predict reconstruction errors achieves better performance. We attribute this to the fact that training the uncertainty head with a more explicit objective–namely, estimating the reconstruction error of each triplane token–is more effective than using a confidence-based objective for token pruning.

F.6. Class-conditioned generation

We provide per-category evaluation results of the category-conditioned generation in Table 14. We also present additional qualitative generation results of all 5 categories in Figures 12 and 13.

F.7. Additional reconstruction results

Finally, we provide additional qualitative reconstruction results. Figure 14 presents reconstruction results on ShapeNet, and Figure 15 presents reconstruction results on Objaverse.

	Method	Airplane	Car	Chair	Table	Rifle
MMD-CD	3DILG	3.702	4.353	9.243	10.526	3.529
	GEM3D	3.587	4.160	8.680	7.652	2.828
	VecSet (M=32)	3.097	4.155	8.173	6.883	2.689
	VecSet (M=64)	3.121	4.173	8.360	6.955	2.841
	VecSet (M=512)	3.059	3.921	7.821	6.527	2.707
	Ours (M=32)	3.275	4.054	8.067	6.885	2.820
	Ours (M=64)	3.240	3.971	8.465	6.695	2.755
MMD-EMD	3DILG	9.04	10.28	13.35	13.46	8.78
	GEM3D	8.75	9.62	13.12	11.91	8.67
	VecSet (M=32)	8.88	10.18	13.12	11.71	8.29
	VecSet (M=64)	8.82	10.02	13.04	11.84	8.71
	VecSet (M=512)	8.64	9.78	12.69	11.55	8.40
	Ours (M=32)	8.79	9.84	13.01	11.76	8.52
	Ours (M=64)	8.71	9.83	13.09	11.59	8.66
COV-CD	3DILG	67.33	41.71	70.41	46.08	65.25
	GEM3D	74.26	49.71	68.93	75.06	65.25
	VecSet (M=32)	88.12	74.29	88.76	90.02	81.36
	VecSet (M=64)	86.14	76.57	85.50	88.12	88.14
	VecSet (M=512)	85.15	76.00	87.28	88.60	88.98
	Ours (M=32)	84.65	77.71	87.28	87.89	87.29
	Ours (M=64)	84.16	79.43	84.32	90.74	91.53
COV-EMD	3DILG	72.77	41.71	76.33	52.02	71.19
	GEM3D	73.27	58.86	64.2	74.58	59.32
	VecSet (M=32)	85.64	60.57	89.05	91.69	87.29
	VecSet (M=64)	90.1	65.71	84.91	91.69	89.83
	VecSet (M=512)	88.61	63.43	85.8	89.55	91.53
	Ours (M=32)	86.14	69.71	89.05	89.79	86.44
	Ours (M=64)	89.6	70.29	86.69	92.16	88.14
1-NNA-CD	3DILG	61.39	61.43	57.69	68.88	58.05
	GEM3D	54.46	58.00	53.55	53.44	54.24
	VecSet (M=32)	54.21	64.29	53.85	50.95	48.31
	VecSet (M=64)	52.97	63.71	54.29	52.97	51.27
	VecSet (M=512)	51.98	62.00	54.73	50.59	49.58
	Ours (M=32)	53.71	60.00	52.07	50.59	49.15
	Ours (M=64)	53.22	61.43	53.99	52.02	53.81
1-NNA-EMD	3DILG	55.69	60.57	58.73	66.86	57.63
	GEM3D	52.72	59.43	56.21	56.65	55.08
	VecSet (M=32)	56.93	63.14	52.96	52.02	48.73
	VecSet (M=64)	51.98	62.29	52.96	53.68	53.81
	VecSet (M=512)	53.96	61.71	53.7	52.14	53.81
	Ours (M=32)	52.72	57.71	53.11	53.09	53.81
	Ours (M=64)	53.96	60.57	52.22	51.07	52.54
Rendering-FID	3DILG	40.49	134.00	38.20	62.63	48.81
	GEM3D	34.97	108.86	32.24	31.95	31.89
	VecSet (M=32)	60.25	132.91	56.26	39.60	38.76
	VecSet (M=64)	52.75	107.52	47.96	34.77	29.35
	VecSet (M=512)	31.57	108.41	26.31	31.78	22.79
	Ours (M=32)	33.57	79.91	27.65	26.89	18.67
	Ours (M=64)	31.72	79.97	27.08	27.53	18.95
Surface-FPD	3DILG	1.13	4.82	1.45	2.55	2.97
	GEM3D	0.82	1.05	0.99	1.02	1.13
	VecSet (M=32)	0.39	2.51	0.39	0.29	0.42
	VecSet (M=64)	0.27	1.87	0.41	0.29	0.31
	VecSet (M=512)	0.26	1.27	0.36	0.31	0.41
	Ours (M=32)	0.26	1.28	0.33	0.24	0.25
	Ours (M=64)	0.25	1.21	0.34	0.26	0.24

Table 14. **Class-conditioned generation results on ShapeNet.** We report per-category evaluation results. The scales of MMD are 10^{-3}, and 10^{-2} for CD, and EMD, respectively.

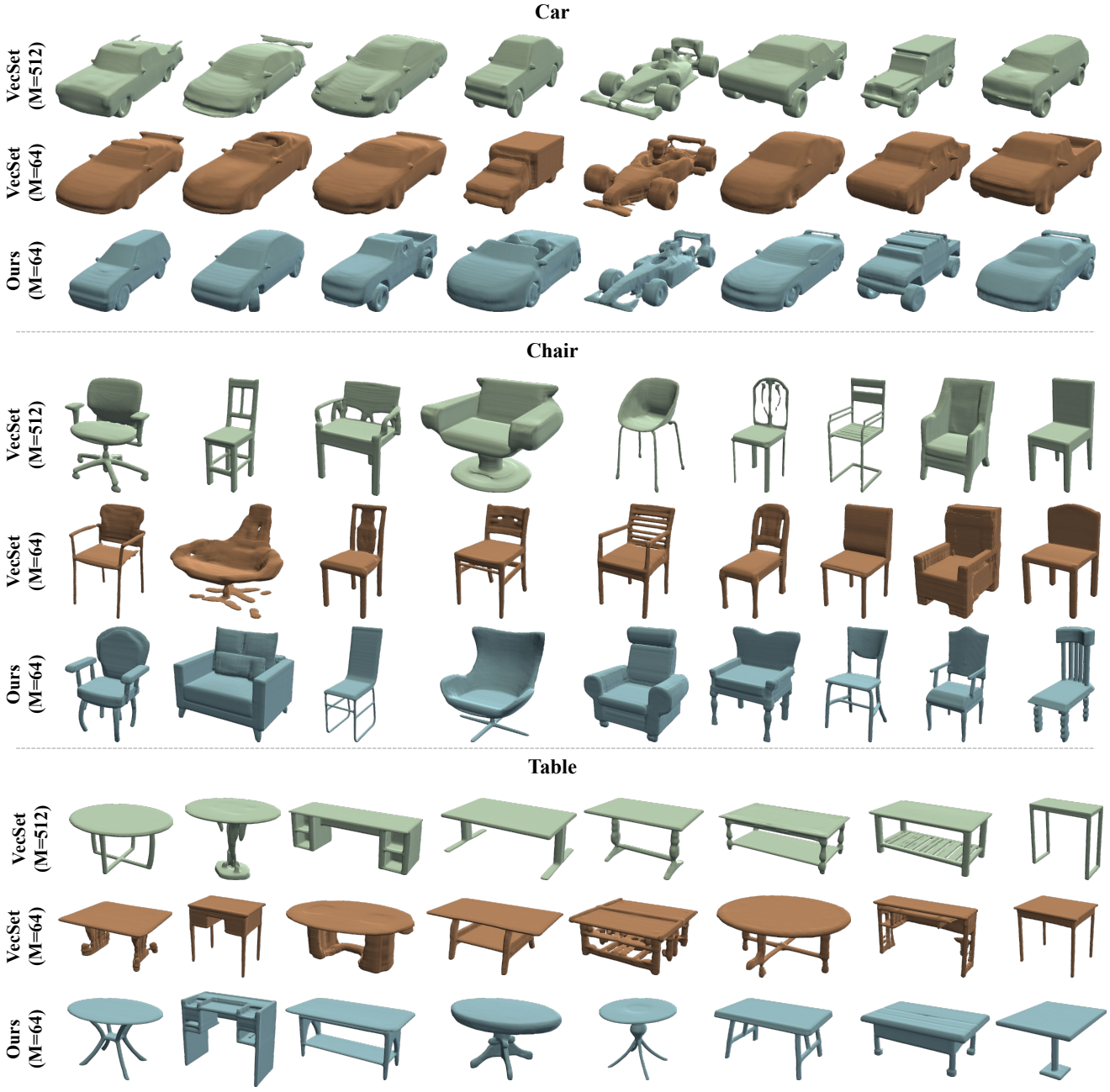

Figure 12. **Additional class-conditioned generation results.** We present the generated results of *car*, *chair*, and *table*.

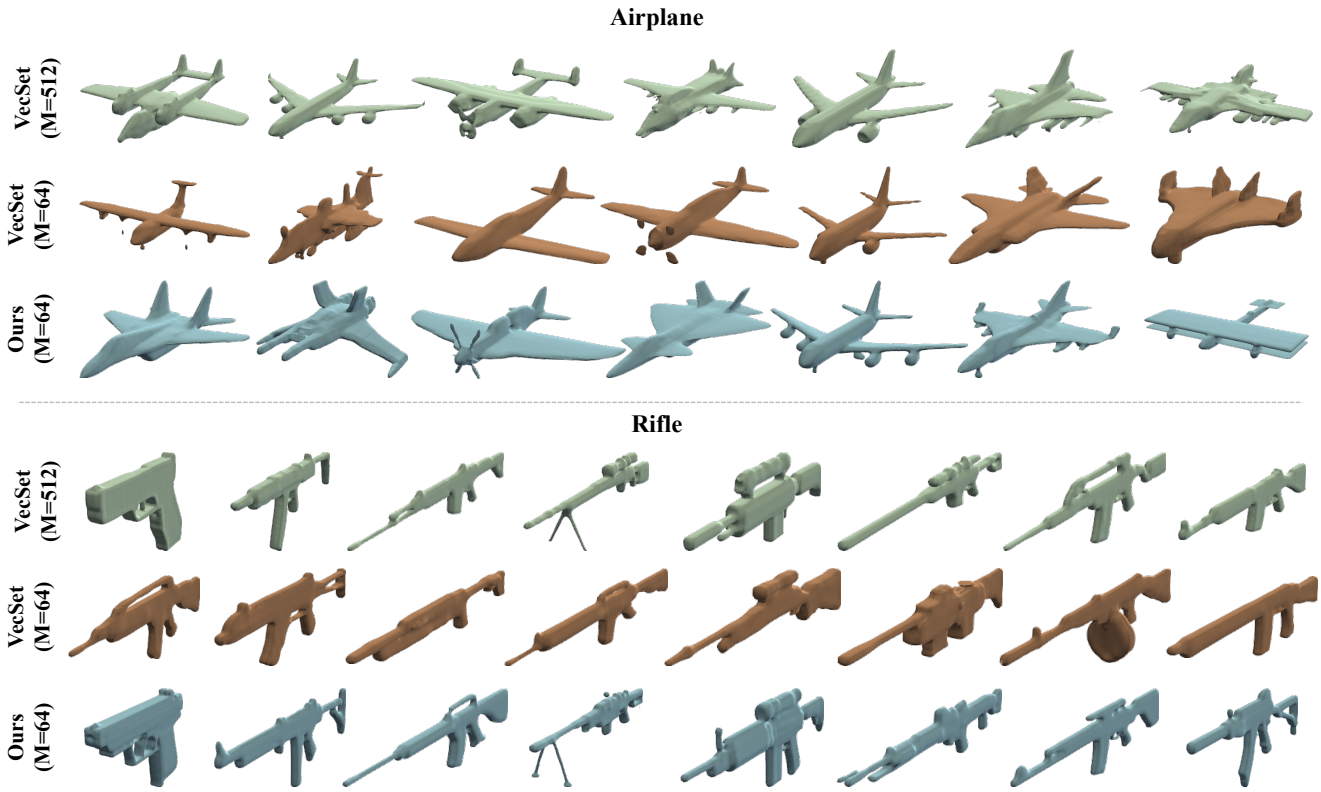

Figure 13. **Additional class-conditioned generation results.** We present the generated results of *airplane* and *rifle*.

Figure 14. **Additional reconstruction results on ShapeNet.** We present the reconstruction results of VAEs.

Figure 15. **Additional reconstruction results on Objaverse.** We present the reconstruction results of VAEs.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 4

[3] Rui Chen, Jianfeng Zhang, Yixun Liang, Guan Luo, Weiyu Li, Jiarui Liu, Xiu Li, Xiaoxiao Long, Jiashi Feng, and Ping Tan. Dora: Sampling and benchmarking for 3d shape variational auto-encoders. *arXiv preprint arXiv:2412.17808*, 2024. 1, 4

[4] Zhaoxi Chen, Jiaxiang Tang, Yuhao Dong, Ziang Cao, Fangzhou Hong, Yushi Lan, Tengfei Wang, Haozhe Xie, Tong Wu, Shunsuke Saito, et al. 3dtopia-xl: Scaling high-quality 3d asset generation via primitive diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26576–26586, 2025. 4

[5] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:16344–16359, 2022. 3

[6] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations (ICLR)*, 2022. 3

[7] Yushi Lan, Fangzhou Hong, Shuai Yang, Shangchen Zhou, Xuyi Meng, Bo Dai, Xingang Pan, and Chen Change Loy. Ln3diff: Scalable latent neural fields diffusion for speedy 3d generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 112–130. Springer, 2024. 1

[8] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470, 2019. 1

[9] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 3

[10] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision (IJCV)*, 40:99–121, 2000. 3

[11] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20697–20709, 2024. 4, 5

[12] Bojun Xiong, Si-Tong Wei, Xin-Yang Zheng, Yan-Pei Cao, Zhouhui Lian, and Peng-Shuai Wang. Octfusion: Octree-based diffusion models for 3d shape generation. *arXiv preprint arXiv:2408.14732*, 2024. 3

[13] Qihang Yu, Mark Weber, Xueqing Deng, Xiaohui Shen, Daniel Cremers, and Liang-Chieh Chen. An image is worth 32 tokens for reconstruction and generation. *arXiv preprint arXiv:2406.07550*, 2024. 2

[14] LAN Yushi, Shangchen Zhou, Zhaoyang Lyu, Fangzhou Hong, Shuai Yang, Bo Dai, Xingang Pan, and Chen Change Loy. Gaussiananything: Interactive point cloud flow matching for 3d generation. In *International Conference on Learning Representations (ICLR)*, 2025. 1, 4

[15] Biao Zhang and Peter Wonka. Lagem: A large geometry model for 3d representation learning and diffusion. *arXiv preprint arXiv:2410.01295*, 2024. 3

[16] Biao Zhang, Matthias Nießner, and Peter Wonka. 3dilg: Irregular latent grids for 3d generative modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 21871–21885, 2022.

[17] Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023. 1, 2, 3

[18] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 43(4):1–20, 2024. 1

[19] Xinyang Zheng, Yang Liu, Pengshuai Wang, and Xin Tong. Sdf-stylegan: implicit sdf-based stylegan for 3d shape generation. In *Computer Graphics Forum*, pages 52–63. Wiley Online Library, 2022. 3