Learning Versatile 3D Shape Generation with Improved Auto-regressive Models

Simian Luo[1,*], Xuelin Qian[1,*], Yanwei Fu[1,†], Yinda Zhang[2], Ying Tai[3]
Zhenyu Zhang[3], Chengjie Wang[3], Xiangyang Xue[1]
[1]Fudan University; [2]Google; [3]Tencent Youtu Lab
{18300180157,xlqian,yanweifu,xyxue}@fudan.edu.cn, yindaz@gmail.com,
zhangjesse@foxmail.com, {yingtai,jasoncjwang}@tencent.com

Supplementary Materials

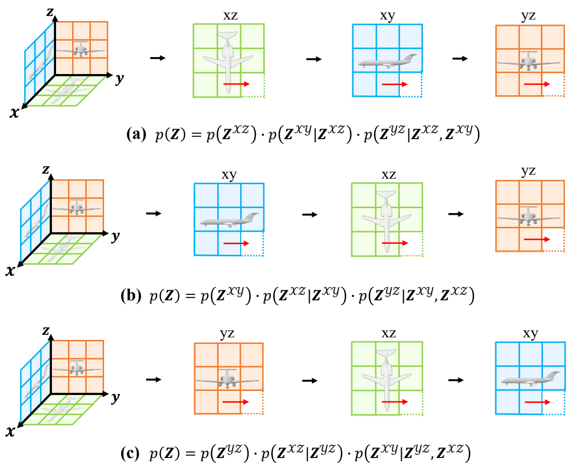
The supplementary document is organized as follows:

- Sec. A provides in-depth discussions of our motivation and further analyze its necessity and significance.
- Sec. B elaborates the technical details of our framework ImAM, including model architectures, data preprocessing, training procedures and implementation of competitors.
- Sec. C reports details of evaluation metrics used in our paper, and insightful discussions about the improved COV metric with a threshold (CovT).
- Sec. D investigates more results of conditional generation tasks as well as baselines, demonstrating the powerful versatility of faithful and diverse shape generation with various conditioning inputs
- Sec. E shows more visualizations of unconditional and conditional generation, further proving the superiority of our model in versatile 3D shape generation.
- Sec. F discusses the broader impact, limitation and future work of this paper.

A. Motivation of ImAM

A.1. What Is the 'Ambiguity' in AR Models?

Formally, *'ambiguity' appears in the order of a series of conditional probabilities, which affects the difficulty of likelihood learning, leading to approximation error of the joint distribution.* Critically, in the second stage of AR models, it requires sequential outputs, autoregressively predicting the next code conditioned on all previous ones. Thereby, the order of the flattened sequence determines the order of conditional probabilities (Eq. 5 and 6 in the main paper). Although some methods (*e.g.* position embedding [18]) can be aware of positions of codes, it cannot eliminate approximation error caused by the condition



(a) $p(\mathbf{Z}) = p(\mathbf{z}^{xz}) \cdot p(\mathbf{z}^{xy}|\mathbf{z}^{xz}) \cdot p(\mathbf{z}^{yz}|\mathbf{z}^{xz},\mathbf{z}^{xy})$

(b) $p(\mathbf{Z}) = p(\mathbf{z}^{xy}) \cdot p(\mathbf{z}^{xz}|\mathbf{z}^{xy}) \cdot p(\mathbf{z}^{yz}|\mathbf{z}^{xy},\mathbf{z}^{xz})$

(c) $p(\mathbf{Z}) = p(\mathbf{z}^{yz}) \cdot p(\mathbf{z}^{xz}|\mathbf{z}^{yz}) \cdot p(\mathbf{z}^{xy}|\mathbf{z}^{yz},\mathbf{z}^{xz})$

Figure 1: Illustration of auto-regressive generation for tri-planar representation. Here, we show three different flattening orders as examples.

order. Notably, this 'ambiguity' phenomenon is also discussed in [8] (Fig. 47). Figure 1 illustrates how the flattening order affects the way of autoregressive generation. For grid-based representation, it is ambiguous if the flattening order along axes is x-y-z, z-x-y or other combinations. Similarly, the flattening order for tri-planar representation is ambiguous, *e.g.*, $p\left(\mathbf{z}^{xz}\right) p\left(\mathbf{z}^{xy}|\mathbf{z}^{xz}\right) p\left(\mathbf{z}^{yz}|\mathbf{z}^{xz}, \mathbf{z}^{xy}\right)$, $p\left(\mathbf{z}^{yz}\right) p\left(\mathbf{z}^{xz}|\mathbf{z}^{yz}\right) p\left(\mathbf{z}^{xy}|\mathbf{z}^{yz}, \mathbf{z}^{xz}\right)$ or others.

A.2. Effect of Flattening Orders

We first investigate how the flattening order affects the quality of shape generation. This study takes tri-planar representation as an example ('Tri-Plane' for short), since *our improved discrete representation ('Vector') can naturally degenerate into 'Tri-Plane' by removing the proposed coupling network.* As illustrate in Fig. 1, different flattening order will affect different auto-regressive generation order of the three planes. Quantitatively, we con-

Metrics	Methods		Categories			
			Plane	Rifle	Chair	Car
ECD ↓	Tri-Plane	Iter-A	744	405	4966	3599
		Iter-B	3501	**36**	1823	4735
		Iter-C	3098	282	4749	3193
	Vector (*ours*)	Row-Major	236	65	**27**	**842**
		Col-Major	**205**	79	102	980
1-NNA ↓	Tri-Plane	Iter-A	73.67	68.35	78.15	87.16
		Iter-B	83.37	**56.54**	70.92	87.42
		Iter-C	81.83	65.61	78.38	88.53
	Vector (*ours*)	Row-Major	**59.95**	57.28	**57.31**	**76.58**
		Col-Major	62.48	57.70	58.38	78.09
COV ↑	Tri-Plane	Iter-A	**81.70**	75.10	79.33	65.31
		Iter-B	74.16	75.52	**82.95**	63.97
		Iter-C	71.32	**76.69**	78.89	72.25
	Vector (*ours*)	Row-Major	79.11	74.26	80.81	**73.25**
		Col-Major	77.87	73.52	81.03	71.31
CovT ↑	Tri-Plane	Iter-A	43.51	41.56	23.10	50.30
		Iter-B	26.57	49.78	35.50	49.43
		Iter-C	30.28	41.35	24.94	51.23
	Vector (*ours*)	Row-Major	**45.12**	**55.27**	**49.82**	**56.64**
		Col-Major	44.87	53.58	48.93	56.63
MMD ↓	Tri-Plane	Iter-A	3237	3962	3392	1373
		Iter-B	3860	3624	3119	1404
		Iter-C	3631	3958	3430	1385
	Vector (*ours*)	Row-Major	3124	3628	**2703**	**1213**
		Col-Major	**3102**	**3623**	2707	1214

Table 1: The effect of flattening order on different discrete representations. Here, we take unconditional generation as an example and train one model per class. Please find statistic and qualitative analyses in Fig. 2 and 3, respectively.

sider three variants to learn joint distributions of tri-planar representation without loss of generality, Iter-A: $p(\mathbf{z}) = p(\mathbf{z}^{xz}) \cdot p(\mathbf{z}^{xy}|\mathbf{z}^{xz}) \cdot p(\mathbf{z}^{yz}|\mathbf{z}^{xz}, \mathbf{z}^{xy})$, Iter-B: $p(\mathbf{z}) = p(\mathbf{z}^{xy}) \cdot p(\mathbf{z}^{xz}|\mathbf{z}^{xy}) \cdot p(\mathbf{z}^{yz}|\mathbf{z}^{xy}, \mathbf{z}^{xz})$ and Iter-C: $p(\mathbf{z}) = p(\mathbf{z}^{yz}) \cdot p(\mathbf{z}^{xz}|\mathbf{z}^{yz}) \cdot p(\mathbf{z}^{xy}|\mathbf{z}^{yz}, \mathbf{z}^{xz})$.

Results are presented in Tab. 1 and Fig. 2. As observed, different orders have significant impact on performance, resulting in a large value of standard deviation. For instance, Iter-A achieves a better result on Plane category (Iter-A: 73.67 *vs.* Iter-B: 83.37 on 1-NNA), while for Rifle, it prefers the order of Iter-B (Iter-B: 56.54 *vs.* Iter-A: 68.35 on 1-NNA). We attempt to explain this phenomenon by visualizing projected shapes on three planes. As illustrated in Fig. 3, for Plane category (First Column), the projection onto the xy-plane provides limited shape information, while the xz-plane reveals more representative geometries of the aircraft. We agree that if the xz-plane that contains more shape information is generated first, the generation of subsequent planes may be much easier. Consequently, it is beneficial for Iter-A to generate more faithful 3D shapes than Iter-B. In contrast, Rifle and Chair exhibit more details on xy-plane, so the autoregressive order of Iter-B yields better results for these two categories. In addition, we notice that Car has a relatively simple shape, *e.g.*, a cuboid, leading to similar impacts on the generation quality for different flattening orders.

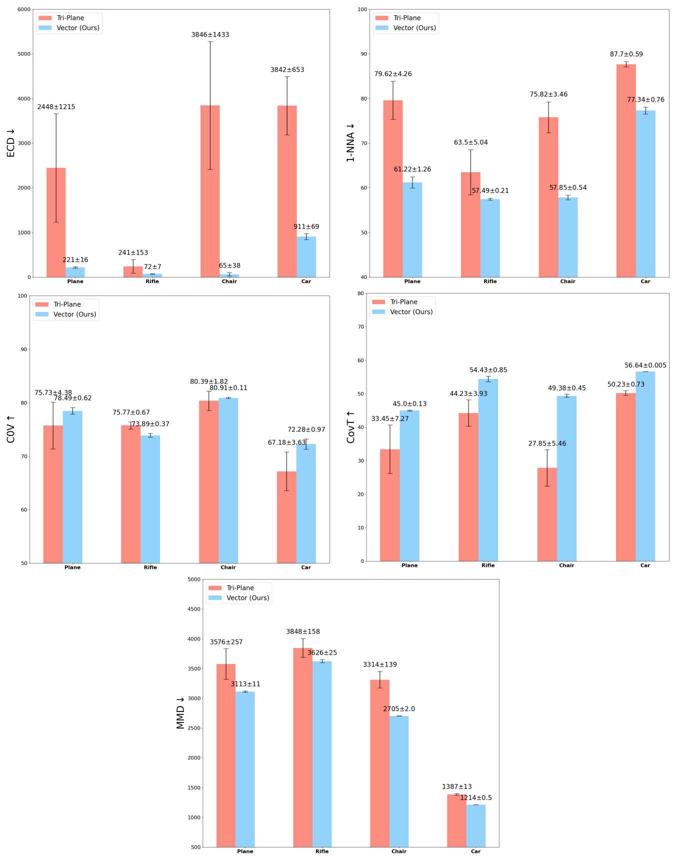


Figure 2: Statistic analysis of the effect of flattening order. We report mean and standard deviation as histogram and error bar, respectively. Best viewed in color and zoom in.

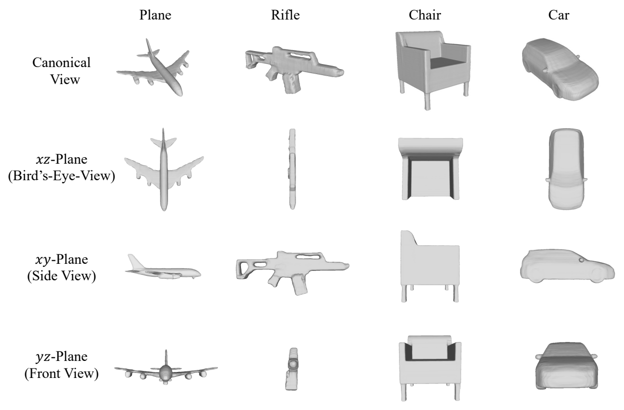


Figure 3: Visualizations of projected shapes on three planes.

A.2. Efficacy of Improved Discrete Representation

In this section, we further evaluate the advantage of our improved discrete representation in terms of efficacy and ef-

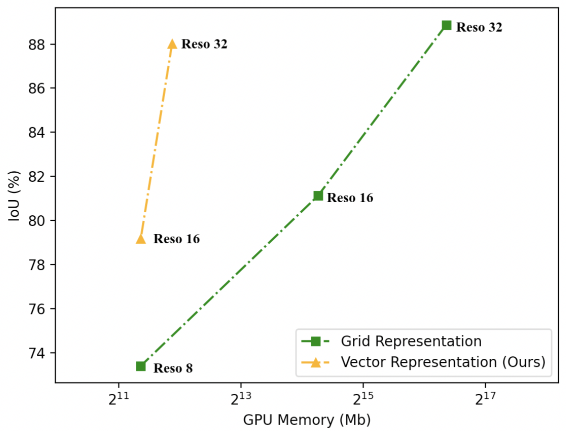


Figure 4: Comparisons of the first-stage IoU (%) accuracy and the second-stage memory cost (Mb) with different resolutions and discrete representations. The memory cost is calculated with a batch size of 1.

ficiency. We explore two variants of our full model by flattening coupled feature maps into vectors with row-major or column-major order. In Tab. 1, our proposed method achieves similar well performance even with different serialization orders. Figure 2 shows that our standard deviation is significantly lower than 'Tri-Plane', demonstrating the robustness of our improved representation to generation orders. The proposed coupling network has facilitated AR learning by introducing more tractable order. Additionally, the overall quality of synthesized shapes for all categories are balanced and excellent across all metrics, indicating the superiority of our design.

Furthermore, we also investigate the advantage of low computation overhead. We use 'Vector' to denote our design since we apply vector quantization to latent vector, and 'Grid' refers to the baseline method that applies vector quantization to volumetric grids. Figure 4 compares the performance of IoU at the first stage and the corresponding memory cost in the second stage. Since we cannot afford the training of transformers with volumetric grid representations, as an alternative, we report the first-stage IoU accuracy for comparisons. From Fig. 4, two conclusion can be drawn. **(1)** The resolution r of feature grids ('Grid' and 'Vector') significantly affects the quality of reconstructed shapes. If r is too small, it lacks the capacity to represent intricate and detailed geometries (Grid Reso-32: 88.87 *v.s* Grid Reso-16: 81.12). However, if r is large, it will inevitably increase the computational complexity in the second stage, since the number of required codes explodes as r grows (Grid Reso-32: $\geq$ 80G *v.s* Grid Reso-16: 19.6G). **(b)** Our proposed 'Vector' representation not only achieves comparable reconstruction results (Vector Reso-32: 88.01 *v.s* Grid Reso-32: 88.87), but also significantly reduces the computation overhead (Vector Reso-32: 3.8G *v.s* Grid Reso-32: $\geq$ 80G).

A.3. Inference Speed

For unconditional generation, the wall time ImAM takes to sample a single shape is roughly 14 seconds, and we can also generation 32 shapes in parallel in 3 minutes.

B. Technical Details on ImAM

B.1. Model Architectures

Our proposed framework ImAM consists of a two-step procedure for 3D shape generation. The first step is an auto-encoder structure, aiming to learn discrete representations for input 3D shapes. And the second step introduces a transformer structure to learn the joint distribution of discrete representations. Below we will elaborate the details of these two structures.

Auto-encoder. As shown in the left of Fig. 5, the auto-encoder takes as input point clouds $\mathcal{P} \in \mathbb{R}^{n \times 3}$ with n means the number of points, and outputs the predicted 3D mesh $\mathcal{M}$. More concretely, the encoder starts by feeding point clouds into a PointNet [15] with local pooling, results in point features with dimensions $\mathbb{R}^{n \times 32}$. Then, we project points on three axis-aligned orthogonal planes with resolution of 256. Features of points falling into the same spatial grid cell are aggregated via mean-operation, so that input point clouds are represented as tri-planar features instead of volumetric features. To further improve the representation, we concatenate three feature planes and couple them with three convolution layers. Next, four stacked convolution layers are adopted to not only down-sample the feature resolution three times, but also highly encode and abstract the position mapping of each spatial grid in 3D space. Thus, the output has a tractable order to be serialized as a feature vector. Before we perform the vector quantization on the flattened outputs, we follow [22] to utilize the strategy of low dimensional codebook lookup, by squeezing the feature dimension from 256 to 4. Consequently, an arbitrary 3D shape can be represented with a compact quantized vector, whose elements are indices of those closest entries in the codebook.

The decoder is composed of two 2D U-Net modules and one symmetric upsample block. After reshaping the quantized vector and unsqueezing its feature dimension from 4 to 256, we apply a 2D U-Net module to complement each spatial grid feature with global knowledge. Subsequently, the same number of 2D convolution layers as the downsmaple block are appended to upsample the feature resolution back to 256. Symmetric convolution layers further decouple it into tri-planer features. To further improve the smoothness between the spatial grids in each plane, we use the other shared 2D U-Net module to separately process tri-plane features. The structures of both 2D U-Net are in alignment

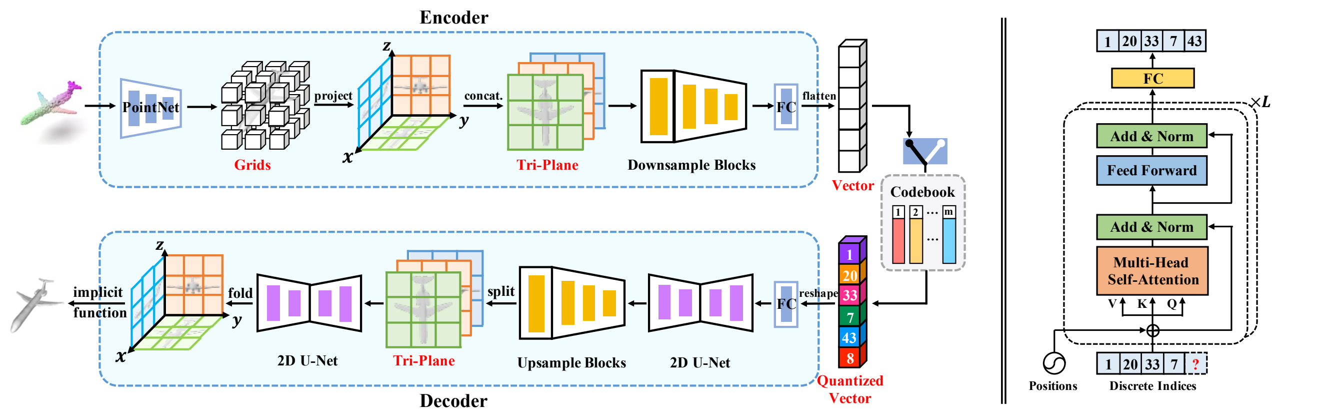


Figure 5: The architectures of ImAM, consisting of an auto-encoder at the first stage (left) and a transformer at the second stage (right).

with [12]. Finally, we build a stack of fully-connected residual blocks with 5 layer, as implicit function, to predict the occupancy probability of each query position.

Transformer. Benefiting from the compact discrete representation with a tractable order for each input 3D shape, we adopt a vanilla decoder-only transformer without any specific-designed module to learn the joint distributions among discrete codes. Our transformer consists of T decoder layers, each of which has one multi-head self-attention layer and one feed-forward network. The decoder layer has the same structure as [8], and is illustrated in the right of Fig. 5. Specifically, we use a learnable start-of-sequence token ([SOS] token) to predict the first index of the discrete vector, and auto-regressively predict the next with the previous ones. For example, given an input containing the first t indices along with one [SOS] token, we first use an embedding layer to encode them as features. Then, we feed them into several transformer layers. The position embedding is also added to provide positional information. At the end of the last transformer layer, we use two fully-connected layers to predict the *logit* of each token. However, we only keep the last one which is a meaningful categorical distribution for the next (t+1)-th index.

Implementation Details. Table 2 summarizes all parameter settings for both auto-encoder and transformer structures. We apply them as default to all experiments unless otherwise stated. The feature dimension d in the transformer varies for different tasks. We set $d = 512$ for unconditional generation; 768 for class-guide generation and partial point completion; and 1024 for image- or text-guide generation. We set the number of transformer layers $T = 12$ for all tasks, except for text-guide generation, where we set $L = 24$ and $h = 16$. Lower triangular mask matrix is used in all multi-head self-attention layers to prevent information leakage, that is, the prediction of the current index is only related to the previous known indices. For various conditioning inputs, we adopt the most common way to encode them. For example, we use a learnable embedding layer to get the feature $\in \mathbb{R}^{1\times768}$ of each category. Gvien partial point clouds, our proposed auto-encoder encodes them into discrete representations $\in \mathbb{R}^{1024\times1}$, which are fed into another embedding layer to get features with 768 dimensions. We adopt pre-trained CLIP models to extract features $\in \mathbb{R}^{1\times512}$ for images or texts, and further use one fully-connected layer to increase the dimension from 512 to 1024; All of encoded conditioning inputs are simply prepended to [SOS] token via concatenation to guide the generation.

B.2. Training and Testing Procedures

Training: All models are trained on a single NVIDIA 3090 or A100, without any learning rate decay strategy. For the first stage, we take dense point clouds with $n = 30,000$ as input, and train the auto-encoder with 13 categories on ShapeNet dataset for total 600k iterations. The learning rate is set as 1e-4, and the batch size is 16. Once trained, it is shared for all generation tasks. For the second stage, we adopt the same learning rate to train the transformer. Except for the partial point completion which has the batch size of 2, we set the batch size of 8 for the other generation tasks. Models for all experiments are trained for around 600k iterations.

Testing: During inference, we first use the well-trained transformer to predict discrete index sequences with or without conditioning inputs. For each index, we sample it with the multinomial distribution according to the predicted probability, where only the top-k indices with the highest confidence are kept for sampling. We progressively sample the next index, until all elements in the sequence are completed. Then, we feed the predicted index sequence into the

Table 2: The detailed architecture of our framework. ‘k’, ‘s’ and ‘p’ denote kernel size, stride and padding, respectively, in the convolution layer. ‘h8’ means the number of head is 8 in multi-head self-attention layer. The feature dimension d in the transformer varies for different tasks. m stands for the dimension of middle layer in the feed-froward network. ‘K’ and ‘L’ are the sequence length of conditioning inputs and discrete representation, ‘1’ indicates the length of [SOS] token.

Layer Name	Notes	Input Size
Auto-encoder		
PointNet		$n \times 3$
Coupler		
ConvLayer	k3s1p1	$256 \times 256 \times 3 \times 32$
ConvLayer	k3s1p1	$256 \times 256 \times 96$
ConvLayer	k1s1p0	$256 \times 256 \times 32$
Downsampler		
ConvLayer	k2s2p0	$256 \times 256 \times 32$
ConvLayer	k2s2p0	$128 \times 128 \times 64$
ConvLayer	k2s2p0	$64 \times 64 \times 128$
ConvLayer	k1s1p0	$32 \times 32 \times 256$
Squeezer	k1s1p0	$32 \times 32 \times 256$
Quantizer		$32 \times 32 \times 4$
Unsqueezer	k1s1p0	$32 \times 32 \times 4$
2D U-Net		$32 \times 32 \times 256$
Upsampler		
DeconvLayer	k3s1p1	$32 \times 32 \times 256$
DeconvLayer	k3s1p1	$64 \times 64 \times 128$
DeconvLayer	k3s1p1	$128 \times 128 \times 64$
ConvLayer	k1s1p0	$256 \times 256 \times 32$
Decoupler		
ConvLayer	k3s1p1	$256 \times 256 \times 32$
ConvLayer	k3s1p1	$256 \times 256 \times 96$
ConvLayer	k1s1p0	$256 \times 256 \times 96$
2D U-Net		$256 \times 256 \times 3 \times 32$
Transformer		
Embedding Layer		$(1 + \mathrm{L}) \times 1$
Decoder Layers $\times$ 12		
Self-Attention	h8	$(\mathrm{K} + 1 + \mathrm{L}) \times d$
Feed-Forward	m4d	$(\mathrm{K} + 1 + \mathrm{L}) \times d$
Head Layer		
LinearLayer		$\mathrm{L} \times d$
LinearLayer		$\mathrm{L} \times d$

decoder to get tri-planar features. Subsequently, we interpolate feature of each point on a grid of resolution 128^3 from tri-planar features, and adopt the implicit function to query the corresponding occupancy probability. Finally, the iso-surface of 3D shapes are extracted with threshold of 0.2 via Marching Cubes [11].

B.3. Data Preparation

We give a more detailed explanation of data preparation. We conduct all the experiments on ShapeNet dataset [3]. Following previous works [6, 14, 12], we use 13 classes of ShapeNet dataset from 3D-R2N2 [7]. The data are processed similarly to C-OccNet [14]. Following the same setting from IM-GAN[6] and GBIF[9], for each category, we sorted the shapes by name and select the first 80% as training samples and the rest for testing. For the task of shape completion, partial point clouds are obtained following the similar strategy in [23]. Specifically, during training, we randomly select a viewpoint and then remove the $25 \sim 75\%$ furthest points from the viewpoint to obtain partial point clouds. 2048 points are further sampled to guarantee the fixed number of partial points as inputs. For a fair comparison, we follow AutoSDF [13] to devise a new completion setting, by removing all the points from the top half of shapes. For the task of image-guide generation, we use rendered images provided by 3D-R2N2[7]. For text-guide generation task, we use Text2Shape dataset [5] with the same data splitting.

B.4. Implementation of Competitors

We select several representative works to verify the effectiveness of our method on five tasks (*i.e.*, T1: unconditional generation, T2: class-guide generation, T3: partial point completion, T4: image-guide generation, T5: text-guide generation). These works follow the priority criteria such as whether they have similar motivation, whether they conduct similar tasks, whether they release source codes, and so on. Here, we list all codebases used in our paper.

- IM-GAN [6] (T1): https://github.com/czq142857/implicit-decoder
- GBIF [9] (T1,T2): https://gitlab.vci.rwth-aachen.de:9000/mibing/localizedimplicitgan
- PointFlow [21] (T1): https://github.com/stevenygd/PointFlow
- ShapeGF [2] (T1): https://github.com/RuojinCai/ShapeGF
- PVD [24] (T1,T3): https://github.com/alexzhou907/PVD
- AutoSDF [13] (T2,T3,T4,T5): https://github.com/yccyenchicheng/AutoSDF
- cGAN [19] (T3): https://github.com/ChrisWu1997/Multimodal-Shape-Completion
- ShapeFormer [20] (T3): https://github.com/QhelDIV/ShapeFormer
- Clip-Forge [16] (T4,T5): https://github.com/AutodeskAILab/Clip-Forge
- ITG [10] (T4): https://github.com/liuzhengzhe/Towards-Implicit-Text-Guided-Shape-Generation

Among them, we slightly modify codes of GBIF and AutoSDF for class-guide generation. Specifically, GBIF designs a generator to synthesize shape embeddings from a random noise, and a discriminator to determine whether the input embedding is real or fake. Thus, we improve it into a conditional GAN by additionally adding a class embedding as input to both the generator and discriminator. AutoSDF is also an auto-regressive model, it has the same two stages as our ImAM, including an auto-encoder and a transformer. Similar to our model, we prepend a class token to the [SOS] token, making AutoSDF be able to perform class-guide generation. Except for these two cases, we either use their provided models or re-train them with official codes if necessary for fair comparisons.

C. Evaluation Metrics

C.1. Implementation Details

As a generator, the key to evaluate our proposed method is not only to measure the *fidelity*, but also to focus on the *diversity* of the synthesized shapes. Therefore, we adopt eight metrics for different generation tasks, including Coverage (COV) [1], Minimum Matching Distance (MMD) [1], Edge Count Difference (ECD) [9] and 1-Nearest Neighbor Accuracy (1-NNA) [21], Total Mutual Difference (TMD) [19], Unidirectional Hausdorff Distance (UHD) [19], Fréchet Point Cloud distance (FPD) [17] and Accuracy (Acc.) [16]. In particular, we use the Light Field Descriptor (LFD) [4] as our primary similarity distance metric for COV, MMD and ECD, as suggested by [6]. Since both FPD and Acc. metrics require a classifier to calculate, we thus train a PointNet [1] with 13 categories on ShapeNet datasets, which achieves the classification accuracy of 92%.

For shape generation task, COV and MMD measures the diversity and fidelity of the generated shapes, respectively. Both suffer from some drawbacks [21]. FPD and Acc. measures the fidelity of the generated shapes from the viewpoint of feature space and probability, respectively. On the contrary, ECD and 1-NNA measure the distribution similarity of a synthesized shape set and a ground-truth shape set in terms of both diversity and quality. Therefore, ECD and 1-NNA are two more reliable and important metrics to quantify the shape generation performance. For shape completion tasks, TMD is meant to the diversity of the generated shapes for a partial input shape, and UHD is proposed to evaluate the completion fidelity. Both metrics are specifically designed for the completion task [19].

C.2. *Coverage* with threshold

As discussed above, *Coverage* [1] measures the diversity of a generate shape set. However, it doesn't penalize outliers since a ground truth shape is still considered as covered even if the distance to the closest generated shape is large [21]. To rule out the false positive coverage, we count as match between a generation and ground truth shape only if LFD [4] between them is smaller than a threshold t. In practice, t could vary across different semantic categories based on the scale and complexity of the shape, and we empirically use MMD [1] as the threshold. In this paper, we set t as mean MMD of all competitors.

To evaluate the effectiveness of the improved COV in identifying correct matches, we visualize the matched pairs with the largest MMD before and after using threshold filtering. As shown on the left of Fig. 6, when there is no threshold constraining the quality of the generated shape, outliers (*e.g.*, a messy shape) could match any possible ground-truth shape, which is clearly unreasonable. On the contrary, when the threshold is applied for filtering, as illustrated on the right of Fig. 6, the generated shape has certain similarity in texture or parts with the matched ground-truth shape, even if they have the maximum shape distance. It strongly demonstrates the validity and reliability of our improvement in measuring the diversity of generation.

D. More Experimental Analysis

D.1. More Baselines on Image-guide Generation

To further evaluate the effectiveness of ImAM on image-guide generation, we compare it with AutoSDF [13], which is also an auto-regressive model for 3D shape generation. We follow its official codes to extract per-location conditionals from the input image and then guide the non-sequential auto-regressive modeling to generate shapes. On the contrary, our method only utilize a single image feature extracted from ResNet or CLIP, guiding the vanilla transformer, to auto-regressively synthesize shapes. As shown in Tab. 3, ImAM beats AutoSDF on three metrics by a large margin, which clearly suggests the effectiveness of our proposed method. In addition, AutoSDF requires a specific form of conditioning inputs, but our approach can accept input in either 1- or 2-D form, giving it more flexibility. To verify it, we conduct a baseline by using ViT32 to extract patch embeddings from the input image as condition. The conditional generation can be achieved by simply prepending patch embeddings to [SOS] token. Results in Tab. 3 indicate that it is competitive with models using ResNet or CLIP as feature extractor, further suggesting the powerful versatility of our ImAM in either generative ways or conditional forms.

D.2. More Baselines on Text-guide Generation

Different images, texts have a natural form of sequence. Each word is closely related to its context. Thereby, we further discuss the ability of ImAM to text-guide genera-

[1]https://github.com/jtpils/TreeGAN/blob/master/evaluation/pointnet.py

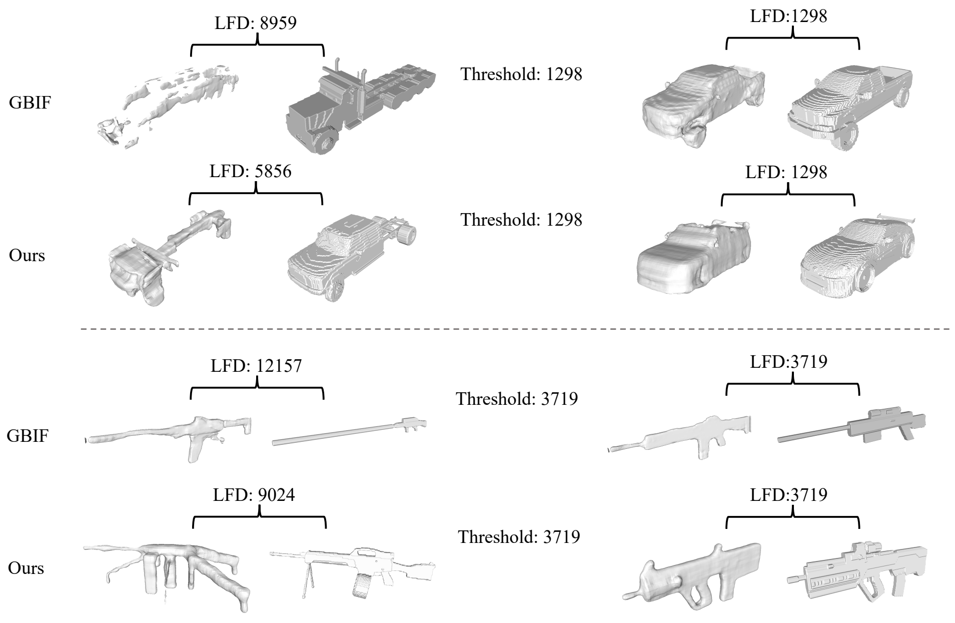


Figure 6: We show the matched pairs with largest MMD before and after using threshold shifting when computing the metric of *Coverage*. For each pair, the left is the generated shape and the right is the closest ground-truth shape.

Table 3: Results of image-guide generation with more baselines.

METHOD	TMD ($\times 10^2$) ↑	MMD ($\times 10^3$) ↓	FPD ↓
AutoSDF [13]	2.523	**1.383**	2.092
Clip-Forge [16]	2.858	1.926	8.094
Ours (ViT32)	3.677	1.617	2.711
Ours (ResNet)	**4.738**	1.662	3.894
Ours (CLIP)	4.274	1.590	**1.680**

Table 4: Quantitative results of text-guide generation.

METHOD	TMD ($\times 10^1$) ↑	MMD ($\times 10^3$) ↓	Acc ↑
ITG [10]	N/A	2.187	29.13
AutoSDF [13]	0.342	2.165	36.95
CLIP-Forge [16]	0.400	2.136	53.68
Ours (BERT)	**0.677**	1.931	**60.68**
Ours (CLIP-seq.)	0.524	**1.778**	58.17
Ours (CLIP)	0.565	1.846	59.93

tion conditioned on sequence embeddings. Concretely, we adopt BERT and CLIP [2] model to encode texts into a fixed length of sequence embeddings. From Tab. 4, we find that using sequence embeddings indeed boost the performance of ImAM for text-guide generation task. Thanks to our proposed compact discrete representation with more tractable orders, our ImAM can be easily adapted to different conditional forms for different tasks, thus improving the quality of the generated shapes.

E. More Qualitative Results

E.1. More Visualizations of Generated Shapes

We show qualitative comparisons of unconditional generation in Fig. 7. Our ImAM can generate more faithful and diverse shapes of multiple categories. Comprehensive visualizations of generated shapes are illustrated in Fig. 8. ImAM can generate shapes with fine geometric structures (*e.g.*, airplanes and cars), as well as some shapes with composite or hollowed-out designs (*e.g.*, tables and chairs). One major key is that it enjoys the advantages of more compact discrete representations while endowing tractable orders to learn shape priors of complicated geometries.

Moreover, we show more synthesized samples of class-guide generation in Fig. 9, partial point completion in Fig. 10, image-guide generation in Fig. 11, and text-guide generation in Fig. 13. ImAM can generates high-quality shapes not only being faithful to the given conditions, but also showing large imagination on diversity, which significantly indicates a more unified ability to freely turn unconditional generation into conditional generation.

[2]In this case, we output features of all tokens instead of the last [END] token as the text embedding.

E.2. Image-guide Generation in Real-world

To further investigate the application of ImAM on real-world image-guide generation, we show more synthesized shapes by given images captured in the real world. We use the same model as described in Sec. 4.4 of the manuscript. As illustrated in Fig. 12, results on five categories suggest that our model can sensitively capture major attributes of objects in the image. Take the first two rows as examples, ImAM can generate plausible shapes while being faithful to objects in images. In addition, two airplane images in Fig. 12 show that our synthesized samples enjoy the advantage of diversity by partially sticking to the major attributes, such the types of wings and tail.

E.3. Zero-shot Text-to-shape Generation

Figure 14 shows more qualitative results of zero-shot text-to-shape generation. Specifically, our model is trained on multiple pairs of image and shapes, and directly take text conditions as input during inference, as the same setting in [16]. The high-quality of synthesized shapes clearly demonstrate the powerful versatility of our proposed ImAM in 3D shape generation, showing great potential to the real-world applications.

F. Broader Impact and Limitation

Broader Impact. Synthesizing high quality 3D content has strong application in the fields of AR/VR, graphics, robotics and metaverse. In the past, creating high quality 3D content requires great effort from professional designers and also a great amount of time. Our work ImAM can serve as a tool for automatically generating high quality 3D shapes, providing convenience for 3D designers to create more sophisticated 3D content. However, at the same time, special care must be taken not to infringe the copyright of other 3D content creators during the process of data collection for our model training.

Limitation. Despite we have made huge effort in adopting an efficient 3D representation in our model, ImAM still inherit the limitation of auto-regressive model. The inference time is relatively consuming, since it requires multiple forward operations to generate one sample. Besides, the way of auto-regression may suffer from the problem of error accumulation. In particular, if conditioning inputs contain some noise, it is very fragile to synthesize incorrect shapes, or even collapsed ones. In future work, we will explore more efficient auto-regressive architectures and representations to overcome these limitations.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018. 6

[2] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pages 364–381. Springer, 2020. 5

[3] 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. 5

[4] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003. 6

[5] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. In *Asian conference on computer vision*, pages 100–116. Springer, 2018. 5

[6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 5, 6

[7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016. 5

[8] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021. 1, 4

[9] Moritz Ibing, Isaak Lim, and Leif Kobbelt. 3d shape generation with grid-based implicit functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13559–13568, 2021. 5, 6

[10] Zhengzhe Liu, Yi Wang, Xiaojuan Qi, and Chi-Wing Fu. Towards implicit text-guided 3d shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17896–17906, 2022. 5, 7

[11] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 5

[12] 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*, pages 4460–4470, 2019. 4, 5

[13] Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. Autosdf: Shape priors for 3d completion, reconstruction and generation. In *Proceedings of*

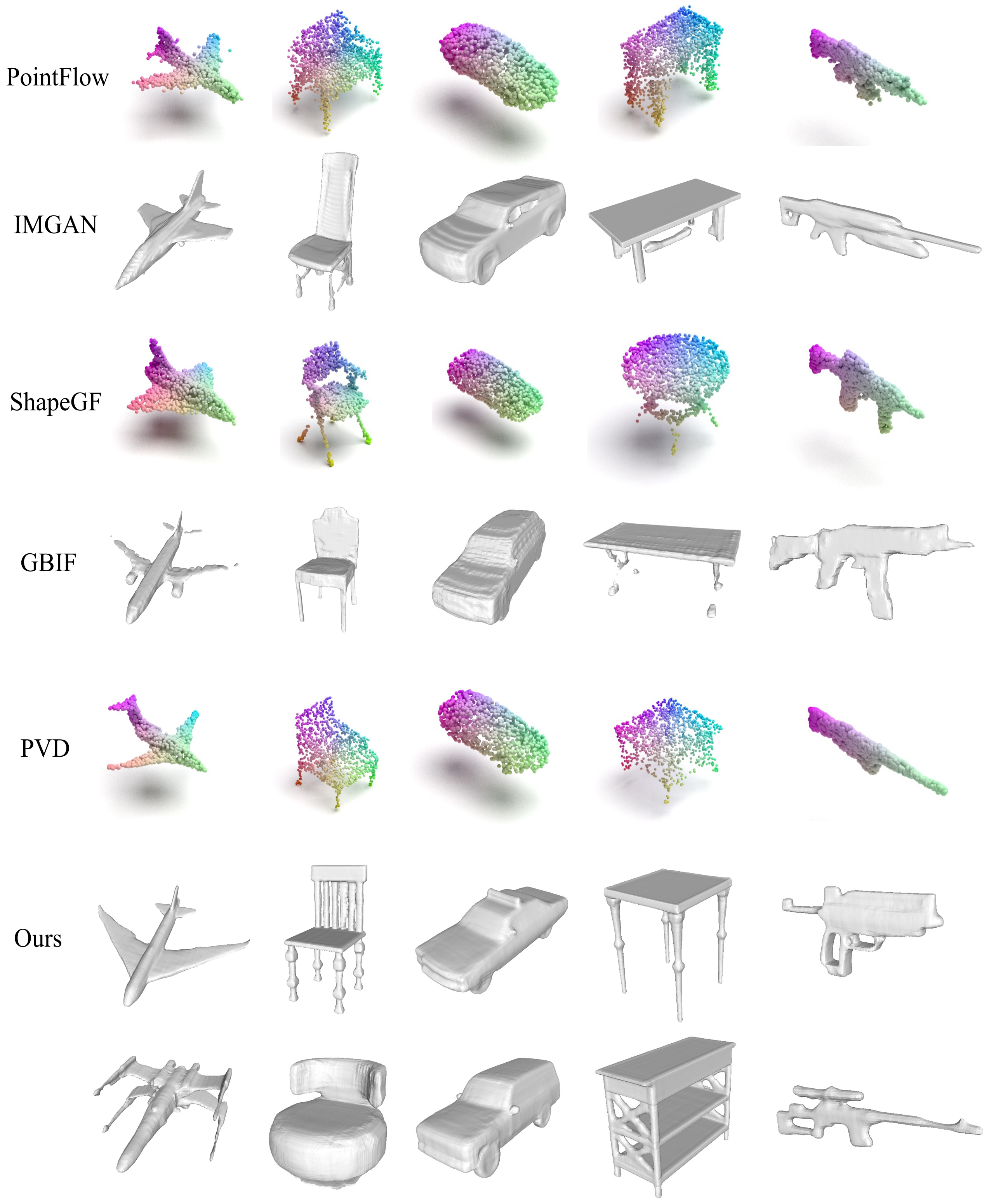


Figure 7: More qualitative comparisons of unconditional generation on 5 categories.

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 306–315, 2022. 5, 6, 7

[14] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, pages 523–540. Springer, 2020. 5

[15] Gernot Riegler, Ali Osman Ulusoy, Horst Bischof, and Andreas Geiger. Octnetfusion: Learning depth fusion from data. In *2017 International Conference on 3D Vision (3DV)*, pages 57–66. IEEE, 2017. 3

[16] Aditya Sanghi, Hang Chu, Joseph G Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. Clip-forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18603–18613, 2022. 5, 6, 7, 8

[17] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3859–3868, 2019. 6

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. 1

[19] Rundi Wu, Xuelin Chen, Yixin Zhuang, and Baoquan Chen. Multimodal shape completion via conditional generative adversarial networks. In *European Conference on Computer Vision*, pages 281–296. Springer, 2020. 5, 6

[20] Xingguang Yan, Liqiang Lin, Niloy J Mitra, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6239–6249, 2022. 5

[21] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019. 5, 6

[22] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. *arXiv preprint arXiv:2110.04627*, 2021. 3

[23] Xumin Yu, Yongming Rao, Ziyi Wang, Zuyan Liu, Jiwen Lu, and Jie Zhou. Pointr: Diverse point cloud completion with geometry-aware transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12498–12507, 2021. 5

[24] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021. 5

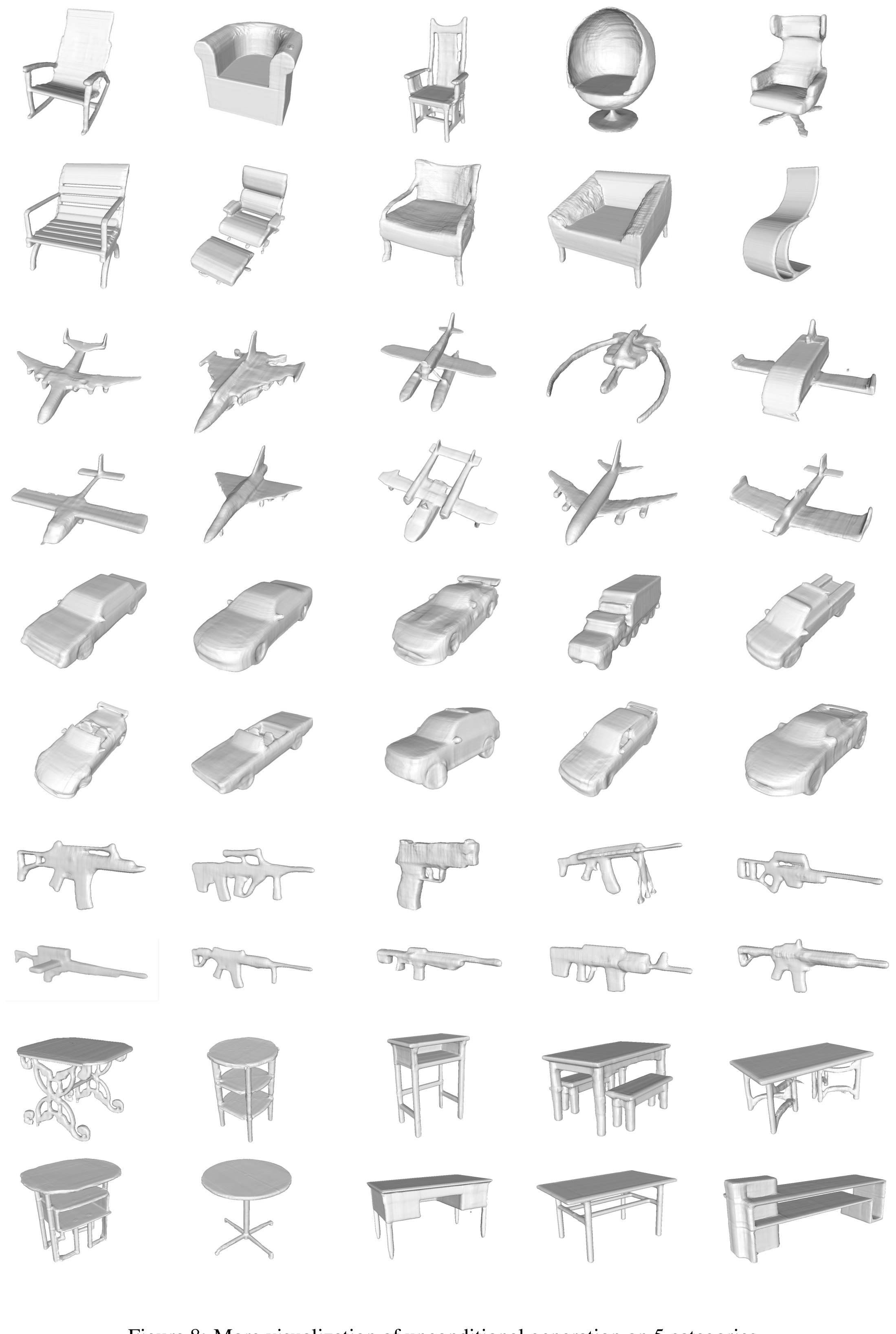


Figure 8: More visualization of unconditional generation on 5 categories.

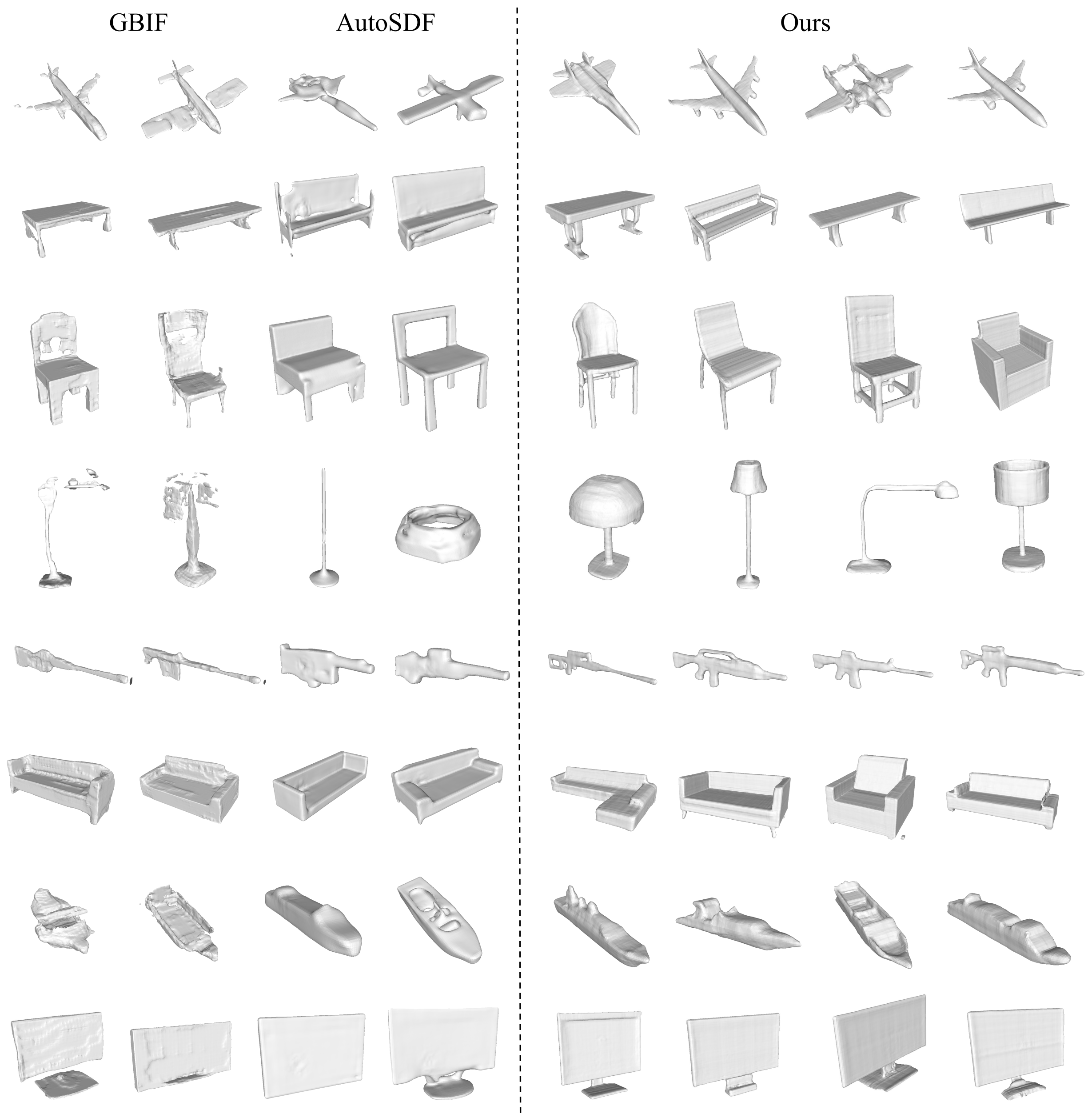


Figure 9: More qualitative comparisons of class-guide generation.

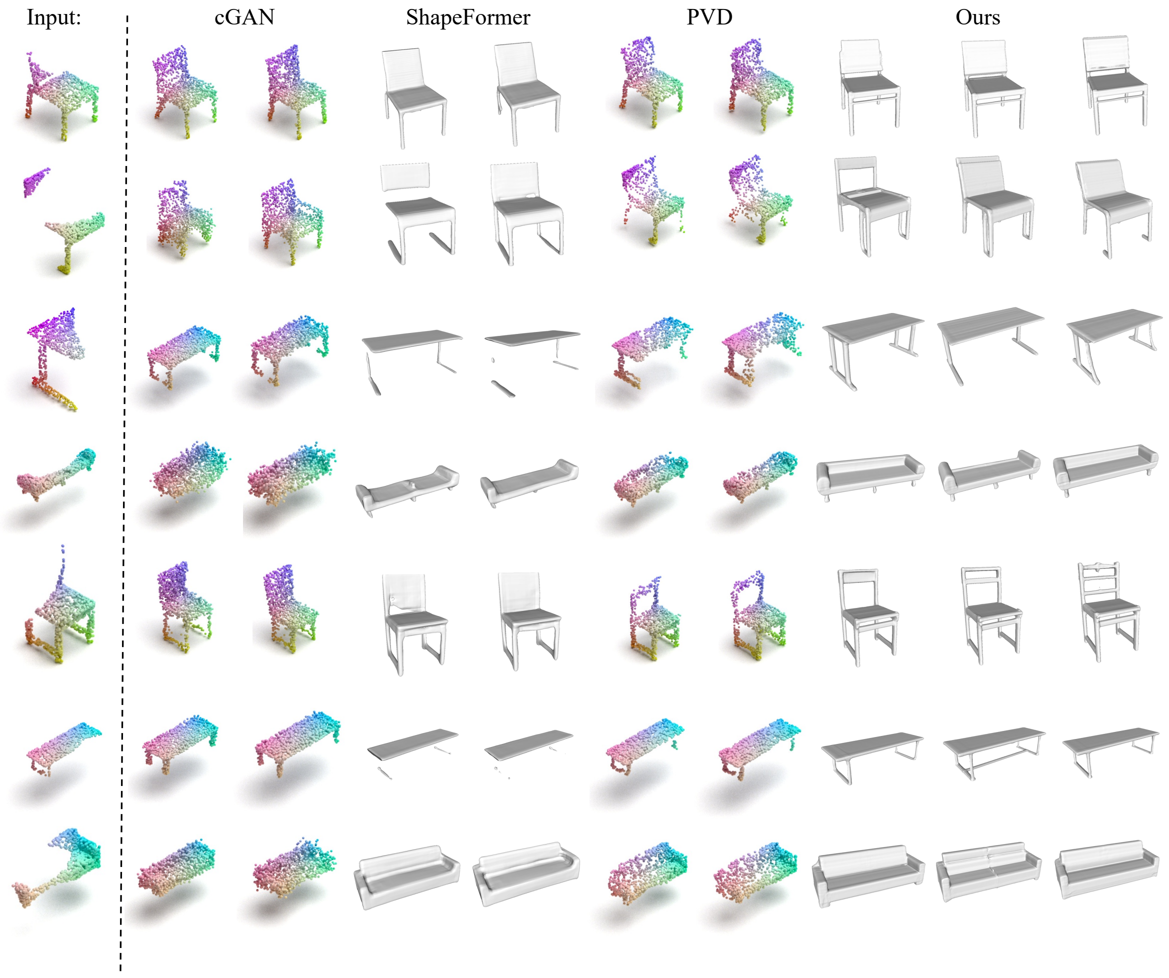


Figure 10: More qualitative comparisons of multi-modal partial point completion on 3 categories.

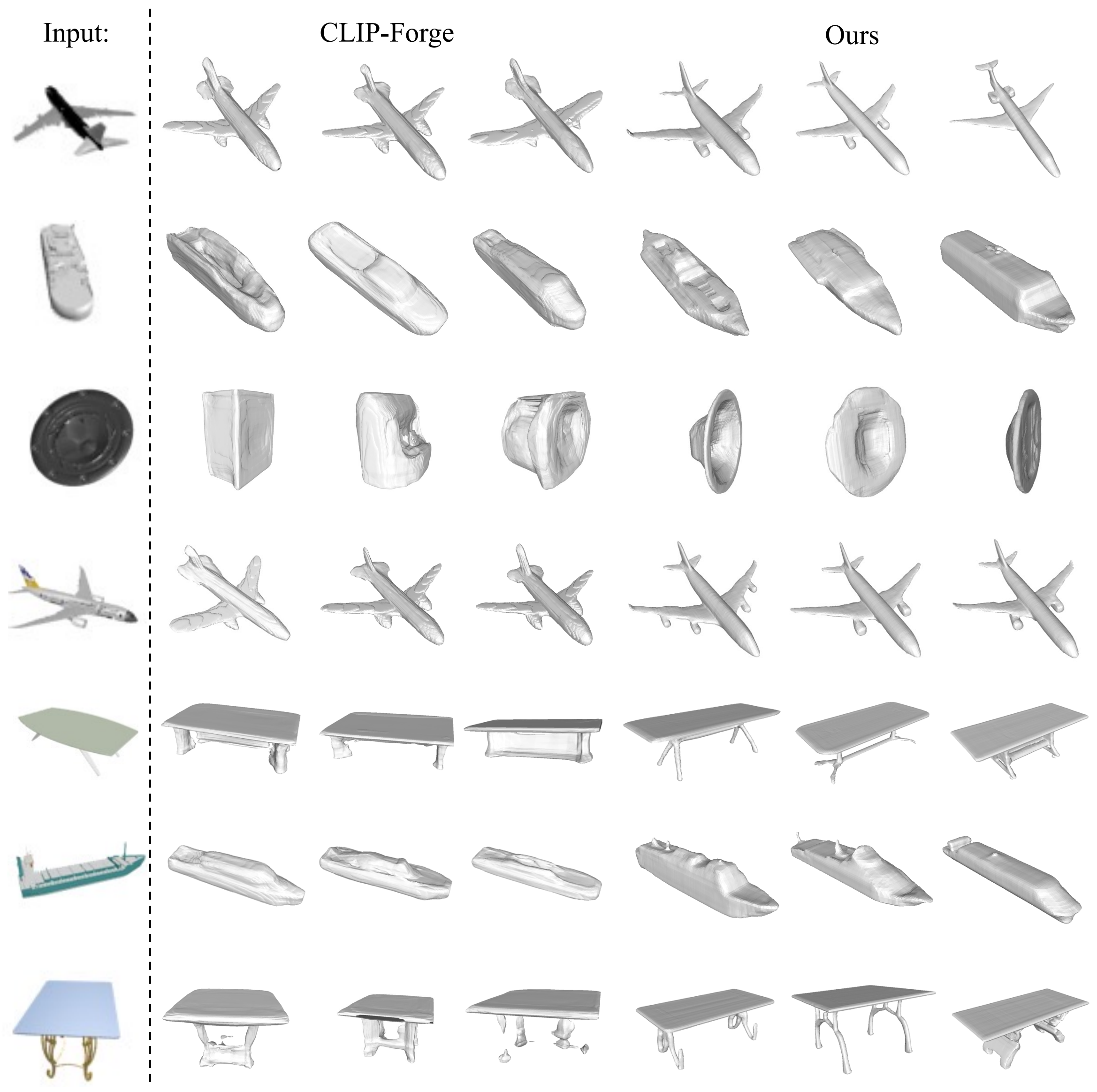


Figure 11: More qualitative comparisons of image-guide generation.

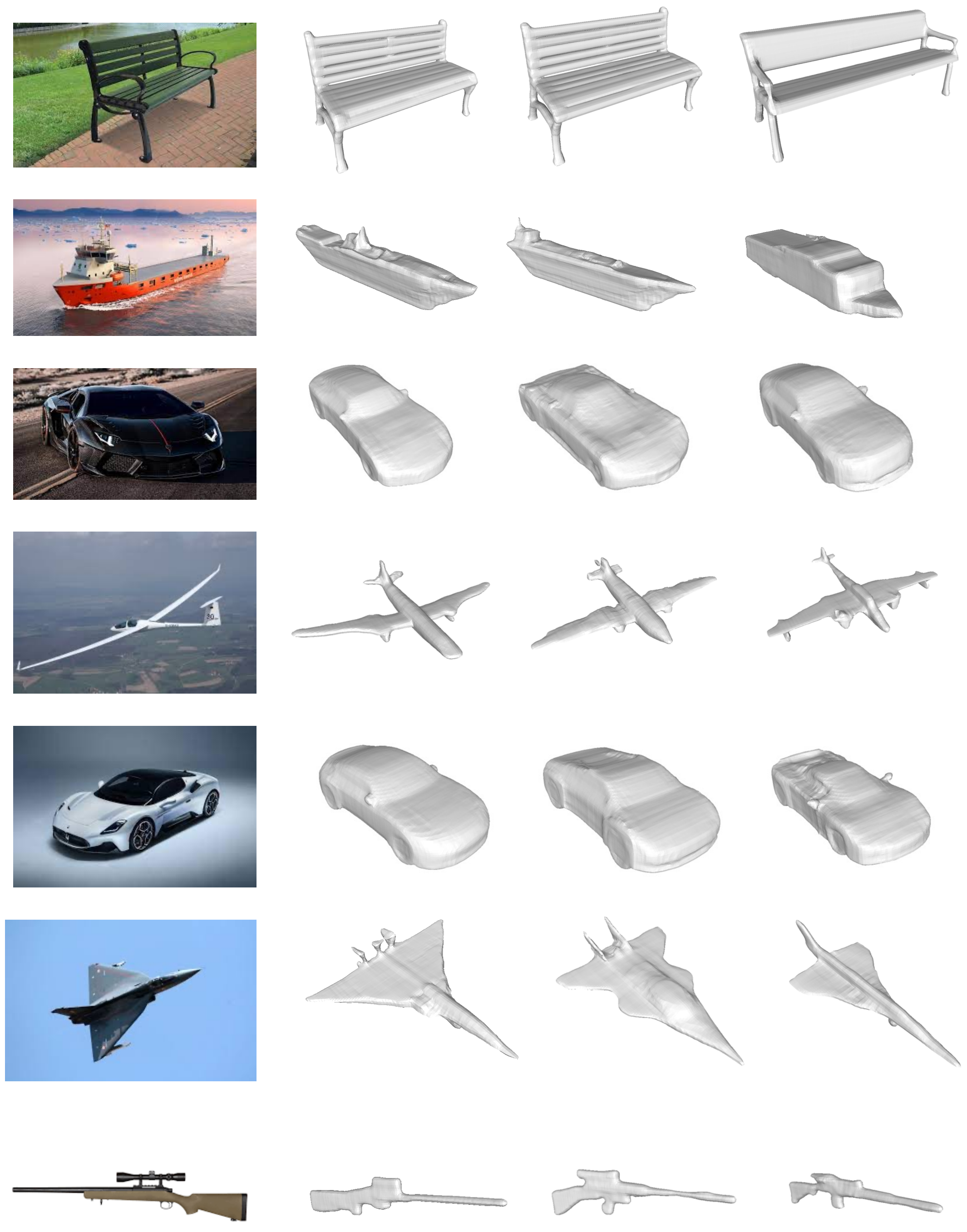


Figure 12: More qualitative results of image-guide generation from real-world data.

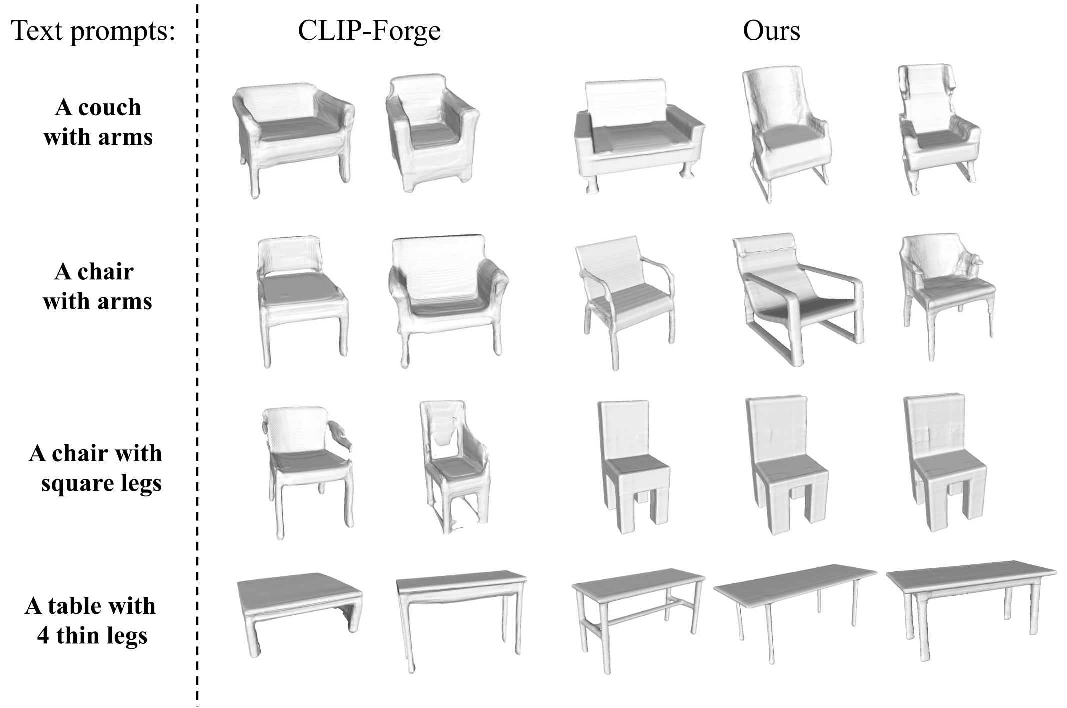


Figure 13: More qualitative comparisons of text-guide generation. Models are trained on Text2Shape dataset which only includes two categories, *i.e.*, chair and table.

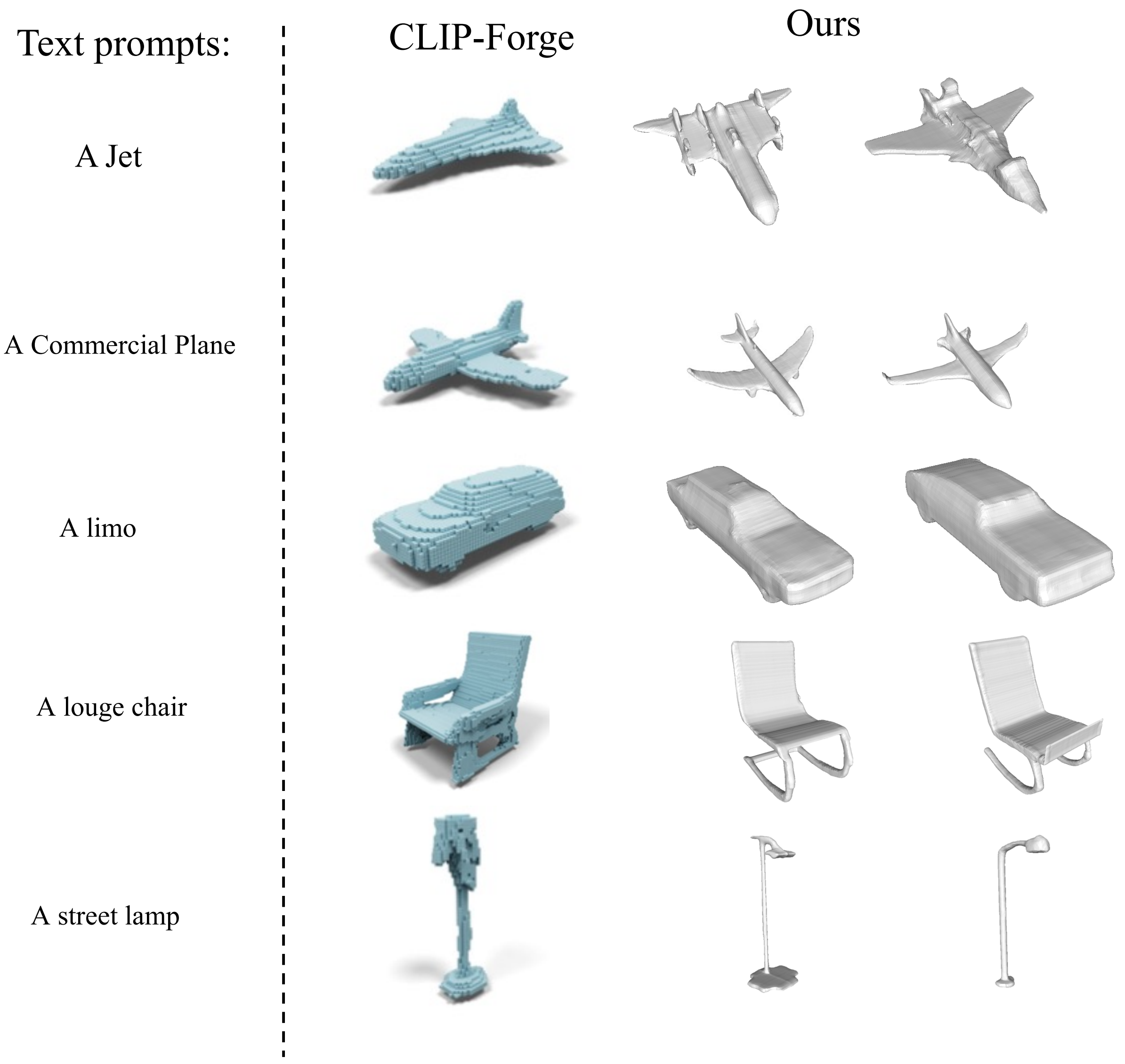


Figure 14: More qualitative results of zero-shot text-to-shape generation. Results of CLIP-Froge are reported in their paper.