JoVox: Joint Hierarchical Flow Matching for Lightweight Voxel Generation

Xiangjun Tang Jianqi Chen Peter Wonka
King Abdullah University of Science and Technology (KAUST)
xiangjun.tang@outlook.com windvchen@gmail.com pwonka@gmail.com

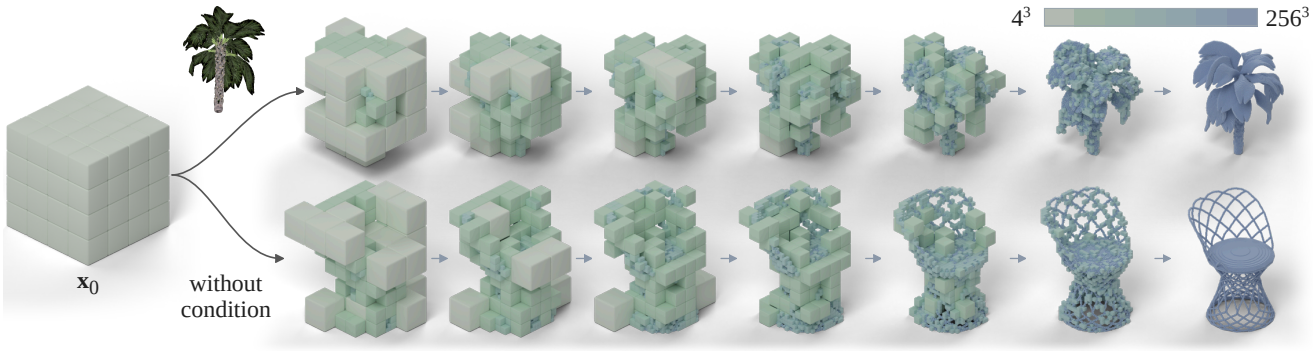


Figure 1. From the same 4^3 grid $\mathbf{x}_0$, we generate voxels conditioned on the input image (top) and without a condition (bottom). We learn the *joint distribution* of the voxels at all resolutions, instead of a separate one for each resolution. Finer resolutions thus start before coarser ones are complete, and every intermediate state mixes several resolutions, each shown in its own color.

Abstract

Generating high-fidelity voxels while keeping their lightweight nature remains an open problem in computer vision. Current generators are either high-fidelity but computationally heavy, or lightweight but of lower fidelity. Hierarchical methods keep the lightweight nature of voxels. However, their coarse levels, which decide where all finer cells can appear, are generated with the least information about the shape and are never revisited, so they are often plausible but wrong. Our key insight is to let finer cells take part in deciding coarse cells. To this end, we propose JoVox, a discrete flow matching framework that learns the joint distribution of all levels of the voxel hierarchy, instead of a separate distribution for each level, so that coarse cells can be predicted with finer cells already generated nearby. Within this framework, we further introduce a two-level prediction, which outputs the cells two levels finer at once instead of one level finer, halving the number of levels that the network processes and yielding voxels of higher fidelity. We also introduce an unmasking strategy that revisits early decisions once finer cells are generated and corrects inconsistencies in masked diffusion models. Our framework outperforms the baselines on both unconditional and image-conditioned voxel generation. Compared with the current state-of-the-art TRELLIS.2, we achieve $8\times$ higher training throughput, $1/15$ of the FLOPs in a forward pass, train on 4 vs. 32 GPUs, and obtain higher quality with an 11% lower Chamfer distance.

1. Introduction

Voxel generation is an important problem in computer vision. Voxels record where a 3D shape lies in space, giving a lightweight and widely used 3D representation. For example, existing models can turn voxels into fine-grained 3D shapes [30, 31], keeping the structure that the voxels define. Yet generating high-fidelity voxels while keeping this lightweight nature remains an open problem.

Existing methods address this problem in two ways. Latent methods [30, 31] achieve high fidelity by compressing the voxel grid with an autoencoder and generating the latent grid with a generator. The autoencoder is an extra model to train, and the generator still processes every cell of the latent grid, including those in empty space, so the computational cost remains high. Hierarchical methods [9, 27, 33] instead generate a coarse grid first and refine its occupied cells into finer grids, level by level, which keeps the lightweight nature of voxels. However, the coarse levels, which decide where all finer cells can appear, are generated with the least information about the shape and are never revisited. They are thus often plausible but wrong, and the finer levels inherit these errors.

Our key insight is to let finer cells take part in decid-

ing coarse cells. To this end, we propose JoVox, a discrete flow matching framework that learns the joint distribution of all levels of the hierarchy, instead of a separate distribution for each level as in hierarchical methods [9, 27, 33]. Such a distribution is hard to model, since the structure of the hierarchy varies across shapes, with finer cells only inside occupied cells. We model it with masked discrete flow matching [7, 11], which unmasks a few cells at each step, at any level of the hierarchy, and grows the hierarchy as cells are unmasked. A coarse cell that is still masked can then be predicted with the finer cells already unmasked nearby. Our framework needs no autoencoder and processes only the cells inside occupied cells, which keeps generation lightweight.

Within this framework, we further introduce two-level prediction and an unmasking strategy. Two-level prediction outputs the cells two levels finer at once, instead of one level finer as in hierarchical methods [6, 9, 27], and thus halves the number of levels that the network processes. We also find that two-level prediction yields voxels of higher fidelity (Section 5.4). Our unmasking strategy remasks the oldest unmasked cells, which were decided with the fewest finer cells around them, so that they are revisited once finer cells are generated. This also corrects the inconsistencies of cells unmasked in the same step [3, 12].

We evaluate our method on both unconditional and image-conditioned voxel generation, where it outperforms current SOTA. In particular, our framework makes training far cheaper than that of TRELLIS.2 [31]. With the same backbone, our framework needs significantly less GPU memory and has $8\times$ higher training throughput (Figure 5), while its voxels remain of higher fidelity, with an 11% lower Chamfer distance.

In summary, our contributions are as follows.

- JoVox, a discrete flow matching framework that learns the joint hierarchical distribution of voxels while keeping generation lightweight.
- Two-level prediction, which outputs the cells two levels finer instead of one level finer, halving the number of levels that the network processes and yielding voxels of higher fidelity, and an unmasking strategy that revisits early decisions once finer cells are generated and corrects inconsistencies in masked diffusion models [3, 12].
- Compared with the current state-of-the-art TRELLIS.2, we achieve $8\times$ higher training throughput, $1/15$ of the FLOPs in a forward pass, train on 4 vs. 32 GPUs, and obtain higher quality with an 11% lower Chamfer distance.

2. Related work

Voxel generation. Voxels are generated as the first stage of many 3D generators [29–32], or obtained by voxelizing the meshes of 3D generators that produce meshes directly [10, 24]. Methods that generate voxels fall into two groups, dense methods and hierarchical methods.

Table 1. Comparison of voxel generation methods.

Method	Hierarchy	No autoencoder	Parallel	Joint levels
TRELLIS, TRELLIS.2	✗	✗	✓	–
LAS-Diffusion	✗	✓	✓	–
DVD	✗	✓	✓	–
XCube	✓	✗	✓	✗
Nexus, OctFusion	✓	✓	✓	✗
Octree Transformer, OAT	✓	✓	✗	✗
OctGPT	✓	✓	✓	✗
Ours	✓	✓	✓	✓

Dense methods generate every cell of a full grid. TRELLIS [30, 31] compresses the voxels into a compact latent grid with an autoencoder and generates this latent grid, which reaches high fidelity at the cost of training an additional autoencoder. LAS-Diffusion [37] and DVD [32] instead generate the occupancy grid directly. LAS-Diffusion generates the occupancy with continuous diffusion and thresholds the result. DVD treats the occupancy as discrete data and generates it with discrete diffusion. Since all these methods generate every cell, including those in empty space, their cost grows with the cube of the resolution.

Hierarchical methods generate the voxels level by level and refine only occupied cells, which skips empty space. XCube [19] generates a sparse voxel hierarchy, with its own latent diffusion model and autoencoder at each level. Nexus [26] and OctFusion [33] generate the octree with continuous diffusion and threshold the result to decide which cells to refine. Octree Transformer [9] and OAT [6] generate the octree autoregressively, one cell at a time, so the number of forward passes grows with the number of cells. OctGPT [27] generates many cells of a level in parallel by masking. However, it decides the finer cells inside a cell independently and needs many steps at every level. In all these methods, a coarse level is decided from the coarser levels alone. Such a level is only roughly plausible.

We instead learn the joint distribution of all levels, so that finer cells take part in deciding coarse cells. Each prediction also outputs the cells two levels finer, instead of one level finer. Like the hierarchical methods above, we refine only occupied cells. We generate the occupancy itself, without an autoencoder. Table 1 summarizes these differences.

Discrete diffusion and flow matching. Masked discrete diffusion models [2, 5, 14, 21] and discrete flow matching [7, 11] generate discrete data by unmasking a masked sequence over several steps, and have been applied to text and image tokens. These data have a fixed size, whereas a voxel hierarchy grows as its cells are unmasked. We extend masked discrete flow matching to such a hierarchy.

When several tokens are unmasked in one step, these models sample them independently, which can make them inconsistent with each other. Some works model the dependence among the tokens unmasked together [8, 12, 34].

Other works choose which tokens to unmask together [3, 15, 20, 28], or remask unmasked tokens so that they can be revised [25]. Our unmasking strategy also remasks, but chooses the oldest unmasked cells, which were decided with the fewest finer cells around them.

3. Discrete flow matching

Discrete flow matching [7, 11] transports samples of a source distribution into samples of a data distribution, along a path indexed by $t \in [0, 1]$. A data sample is a tensor $\mathbf{x} = (x^1, \dots, x^n)$ with n entries from a finite vocabulary $\mathcal{T}$. The i-th entry of $\mathbf{x}$ is x^i. Throughout, $\mathbf{x}_1$ denotes a sample of the data distribution, and $\mathbf{x}_0$ a sample of the source distribution, which is a modeling choice. $\mathbf{x}_t$ denotes the tensor at t on the path between $\mathbf{x}_0$ and $\mathbf{x}_1$.

A common modeling choice of the source distribution is the masked source [2, 21]. Instead of sampling from a probability distribution, the masked source is a fully masked tensor $\mathbf{x}_0 = (\varnothing, \dots, \varnothing)$, where $\varnothing \notin \mathcal{T}$ is a mask symbol.

Masking. The path from the data to the source is thereby a masking process. Specifically, given $\mathbf{x}_1$, each entry independently keeps its value with probability t and is masked otherwise. The distribution of $\mathbf{x}_t$ given $\mathbf{x}_1$ is

$$p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1) = \prod_{i=1}^{n} \Big[t\, \delta(x_t^i, x_1^i) + (1-t)\, \delta(x_t^i, \varnothing) \Big], \quad (1)$$

with $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise. An entry whose value is in $\mathcal{T}$ is called unmasked.

Unmasking. A flow matching model generates data by unmasking. One forward pass of the model, a network with parameters θ, returns probabilities over $\mathcal{T}$ for every entry,

$$p_\theta(x_1^i \mid \mathbf{x}_t, t), \qquad i = 1, \dots, n, \quad (2)$$

each approximating the distribution of x_1^i given $\mathbf{x}_t$. For the currently masked entries, these probabilities are predicted considering all already unmasked entries of $\mathbf{x}_t$. At each step of generation, a random subset of the masked entries is unmasked, each with a value sampled from its predicted probabilities.

Loss. The model is fitted by minimizing the loss of discrete flow matching [7, 11]. This loss uses the predicted probabilities of both masked and unmasked entries. For the masked path, the loss of entry i is

$$\ell^i = \begin{cases} -\log p_\theta(x_1^i \mid \mathbf{x}_t, t) & \text{if } x_t^i = \varnothing, \\ 1 - p_\theta(x_1^i \mid \mathbf{x}_t, t) & \text{otherwise.} \end{cases} \quad (3)$$

The first case is the cross-entropy on a masked entry. The second penalizes a low probability of the unmasked value. The model minimizes

$$\mathcal{L} = \int_0^1 \frac{1}{1-t}\, \mathbb{E}\Big[\textstyle\sum_i \ell^i \Big]\, \mathrm{d}t. \quad (4)$$

4. Method

Given an optional condition c, such as an image, our task is to generate the occupancy of a shape. We build upon masked discrete flow matching (Section 3) to model the joint distribution of all levels of the voxel hierarchy. Section 4.1 defines the hierarchy and models its levels jointly, and Section 4.2 introduces two-level prediction. Sections 4.3 and 4.4 describe the network and the training, and Section 4.5 introduces the unmasking strategy for inference.

4.1. Joint hierarchical modeling

A shape is represented by its occupied cells at the levels $l = 0, \dots, L$. Level l is a grid of $(4 \cdot 2^l)^3$ cells, and a cell is occupied if the shape surface passes through it. Each cell at level $l < L$ contains eight cells at level $l + 1$, which we call its children. As in an octree, a cell is identified by its level l and its position in the grid of level l. We denote by O_l the set of occupied cells at level l and sample $O_0, \dots, O_L$ from $p(O_0, \dots, O_L \mid c)$.

Level-by-level generation factorizes this distribution into $p(O_0 \mid c) \prod_{l=1}^{L} p(O_l \mid O_{l-1}, c)$, so each level is decided given only the coarser levels. We instead learn the joint distribution of all levels with the masked discrete flow matching of Section 3, so that a coarse cell can be predicted given finer cells that are already generated.

Occupancies. Without loss of generality, we start with a straightforward choice of the entries x^i of Section 3 for hierarchical modeling. We define the entry $x^i \in \mathbf{x}_1$ as the occupancy of a cell. The occupancy is 1 if the cell is occupied and 0 otherwise, so that $x^i \in \mathcal{T} = \{0, 1\}$. During discrete flow matching, an occupancy can also be masked, and a masked occupancy is the mask symbol $\varnothing$ of Section 3.

Masking. The data sample $\mathbf{x}_1$ contains the occupancies of the cells at level 0 and of the children of occupied cells, and it determines $O_0, \dots, O_L$. As in Section 3, we obtain an intermediate state by masking some occupancies of $\mathbf{x}_1$. Masking the occupancy of a cell also removes all cells inside it, since their occupancies would reveal the masked occupancy. We mask the levels $J = \{0, \dots, L\}$ in order, from the coarsest to the finest. At level l, each occupancy keeps its value with probability t_l and is masked otherwise. The resulting intermediate state $\mathbf{x_t}$, with $\mathbf{t} = (t_l)_{l \in J}$, thus contains the occupancies of $\mathbf{x}_1$ whose cells are not removed, and each of these occupancies keeps its value or is masked. $\mathbf{x_t}$ follows

$$p_{\mathbf{t}|1}(\mathbf{x_t} \mid \mathbf{x}_1) = \prod_{l \in J} \prod_{i \in \mathcal{C}_l(\mathbf{x_t})} \Big[t_l\, \delta(x_{t_l}^i, x_1^i) + (1-t_l)\, \delta(x_{t_l}^i, \varnothing) \Big], \quad (5)$$

where $\mathcal{C}_l(\mathbf{x_t})$ is the set of occupancies at level l in $\mathbf{x_t}$. The number of occupancies thus depends on which cells are known to be occupied. It differs between shapes, and even for the same shape it changes as the shape is built during

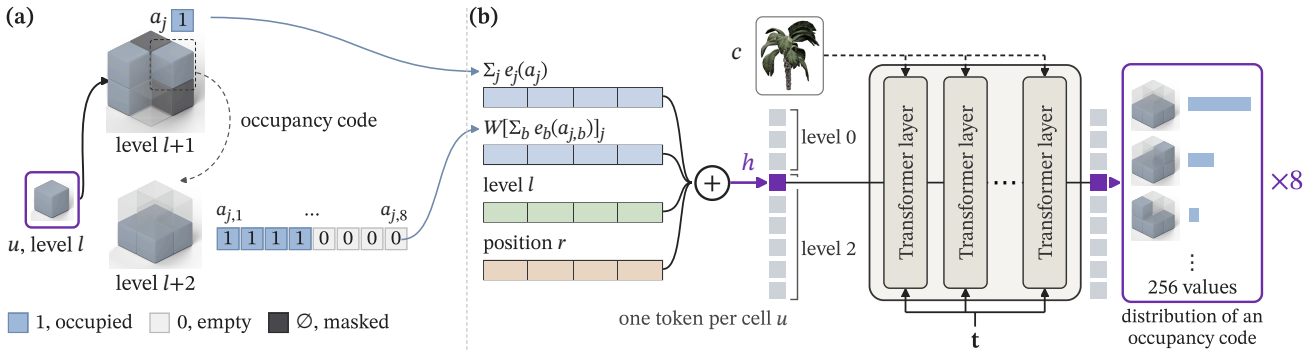


Figure 2. (a) Two-level prediction. For each cell u at level l, the network predicts the occupancy codes of the eight children of u at level $l+1$. The occupancy code of the j-th child consists of the values $a_{j,1}, \ldots, a_{j,8}$ of its eight children at level $l+2$, where a value is 1 for an occupied cell, 0 for an empty one and ∅ for a cell whose occupancy is unknown. When this occupancy code is masked, the value a_j of the j-th child and $a_{j,1}, \ldots, a_{j,8}$ are all ∅. (b) The network. Each cell u at level 0 and each occupied cell u at even levels has one token. The token h embeds the values a_j and $a_{j,b}$, the level l and the position r. A transformer processes the tokens of all levels together, conditioned on c and on the fractions $\mathbf{t}$ of unmasked occupancy codes, and outputs for u eight distributions over the 256 values of an occupancy code.

discrete flow matching.

Unmasking. Given $\mathbf{x_t}$, $\mathbf{t}$ and c, the model processes each cell whose occupancy is in $\mathbf{x_t}$ and returns probabilities over $\mathcal{T}$ for its occupancy,

$$p_\theta(x_1^i \mid \mathbf{x_t}, \mathbf{t}, c), \qquad i \in \mathcal{C}_l(\mathbf{x_t}),\ l \in J. \tag{6}$$

Generation starts from $\mathbf{x}_0$. At each step, we choose masked occupancies at any level at random (Section 4.5) and unmask them by sampling their values from the probabilities of Equation (6). Once a cell at level $l < L$ is known to be occupied, we add masked occupancies of its children to $\mathbf{x_t}$, so the hierarchy grows as occupancies are unmasked. Since occupancies at any level can be unmasked, some occupancies at a coarse level can still be masked when occupancies at finer levels in nearby cells are already unmasked.

4.2. Two-level prediction

Hierarchical methods such as Octree Transformer [9] define the *occupancy code* of a cell as the occupancy of its eight children, which has 8 bits. They process a cell to predict its occupancy code, one level finer, instead of its own occupancy as in Section 4.1. We call this single-level prediction. In our framework, single-level prediction uses the formulation of Section 4.1 with occupancy codes in place of occupancies. The entries $x^i \in \mathcal{T} = \{0,1\}^8$ are the occupancy codes at the levels $J = \{0, \ldots, L-1\}$. The data sample $\mathbf{x}_1$ contains the occupancy codes of the cells at level 0 and of the occupied cells at the levels $1, \ldots, L-1$. We mask the occupancy codes as in Section 4.1 to obtain $\mathbf{x_t}$, where $\mathcal{C}_l(\mathbf{x_t})$ is now the set of occupancy codes at level l in $\mathbf{x_t}$. Once a cell at the levels $1, \ldots, L-1$ is known to be occupied, we add its masked occupancy code to $\mathbf{x_t}$.

In two-level prediction, the model instead processes a cell u at level $l \le L-2$ to predict the occupancy of its $4 \times 4 \times 4$ cells two levels finer, at level $l+2$ (Figure 2 (a)). One possible way is to use the occupancy of these cells as one entry, but such an entry would take 2^{64} values, which is prohibitively large. We therefore split the occupancy of these cells into eight 8-bit occupancy codes, one for each child of u, each giving the occupancy of the eight cells at level $l+2$ inside that child. For a cell u, the model thus takes these eight occupancy codes, masked or unmasked, as input and outputs eight distributions of Equation (6), each over the 2^8 values of an occupancy code, instead of one distribution over 2^{64} values.

Specifically, we take L to be even and let the cells u lie at the even levels $0, 2, \ldots, L-2$, so that the entries are the occupancy codes at the odd levels $J = \{1, 3, \ldots, L-1\}$. The data sample $\mathbf{x}_1$ contains the 8-bit occupancy codes of the eight children of every cell u at level 0 and of every occupied cell u at the levels $2, \ldots, L-2$. We mask the occupancy codes as in single-level prediction to obtain $\mathbf{x_t}$. Once a cell u at the levels $2, \ldots, L-2$ is known to be occupied, we add the masked occupancy codes of its eight children to $\mathbf{x_t}$. The model thus processes each cell u at level 0 and each cell u known to be occupied. Two-level prediction thus processes cells at only half of the levels of single-level prediction, so training is faster (Section 5.4).

4.3. Network

The network is a transformer that takes one token for each cell u of Section 4.2, as illustrated in Figure 2 (b).

As in Figure 2 (a), we number the children of a cell $1, \ldots, 8$ in a fixed order. We write a_j for the value of the j-th child of u and $a_{j,b}$ for the value of the b-th child of this child. A value is 1 if the cell is occupied, 0 if it is empty, and ∅ if the occupancy code of the j-th child is masked. The token h of u embeds these values, the level l and the

position r of u as

$$h = \sum_{j=1}^{8} e_j(a_j) + W\Big[\sum_{b=1}^{8} e_b(a_{j,b})\Big]_{j=1}^{8} + \mathrm{emb}_{\mathrm{level}}(l) + \mathrm{emb}_{\mathrm{pos}}(r), \tag{7}$$

where e_m is a learned embedding of the values $\{0, 1, \varnothing\}$ for the m-th child of a cell, used both for the children of u, as e_j, and for the cells inside each child, as e_b. The first term embeds the eight children. The second sums the embeddings of the eight cells inside each child. The eight sums are concatenated in the order of the children, written $[\cdot]_{j=1}^{8}$, and a learned linear map W maps them to one vector. The level embedding is one learned vector per level. The point $r \in [0, 1)^3$ is the center of u in a frame shared by all levels. $\mathrm{emb}_{\mathrm{pos}}(r)$ is the point embedding of 3DShape2VecSet [36].

The transformer has the same architecture as the voxel stage of TRELLIS.2 [31]. It processes the tokens of all levels in one sequence, with full self-attention and 3D rotary position embeddings of r. When c is an image, we feed its DINOv3 features [22] to every block through cross-attention. We embed each t_l with its own time embedding emb_t^l, and condition all tokens on the sum $\sum_{l \in J} \mathrm{emb}_t^l(t_l)$ through adaptive layer normalization in every block.

4.4. Training

During training, we first draw t_l for each level $l \in J$ and then draw $\mathbf{x_t}$ from Equation (5). We draw the t_l of different levels independently, so the network is trained on any combination of fractions across levels. The intermediate states $\mathbf{x_t}$ produced by any unmasking strategy therefore lie within the training distribution. To train the network more where its predictions are least certain, we draw small t_l more often, following the distribution given in Section B in the appendix.

Finer levels have far more occupancy codes than coarser ones, so a loss summed over all occupancy codes would be dominated by the finer levels. We therefore first average the loss over the occupancy codes of each level, and then average over the levels. With $J(\mathbf{x_t})$ the set of levels that have occupancy codes in $\mathbf{x_t}$, the loss is

$$\mathcal{L}_{\mathrm{ours}} = \mathbb{E}\Big[\frac{1}{|J(\mathbf{x_t})|} \sum_{l \in J(\mathbf{x_t})} \frac{1}{(1 - t_l)\,|\mathcal{C}_l(\mathbf{x_t})|} \sum_{i \in \mathcal{C}_l(\mathbf{x_t})} \ell^i\Big], \tag{8}$$

where ℓ^i is the loss of Equation (3), and each level is weighted by $1/(1 - t_l)$ as in Equation (4).

4.5. Unmasking strategy

Occupancy codes unmasked in the same step are sampled independently, each from its predicted probabilities, so they may be inconsistent with each other [3, 8, 12, 34]. Some samplers for masked generation correct such inconsistencies by remasking unmasked entries, so that these entries are predicted again [25]. Our strategy also remasks, and chooses the occupancy codes to remask by when they were unmasked. Occupancy codes unmasked early are sampled given few unmasked occupancy codes. This holds in particular for coarse occupancy codes, which are unmasked before most finer occupancy codes exist. Our strategy therefore remasks the oldest unmasked occupancy codes at each step, so that they are predicted again given the occupancy codes unmasked after them, including finer occupancy codes in nearby cells.

Generation starts from $\mathbf{x}_0$ and runs for N steps $k = 0, \ldots, N - 1$. At step k, the target fraction of unmasked occupancy codes at level $l \in J$ is given by the schedule

$$\kappa_l(k) = \min(k/N_l, 1)^{\rho}, \tag{9}$$

which reaches 1 at $k = N_l \le N$. Finer occupancy codes are added as coarser ones are unmasked, so we let the schedules of coarser levels reach 1 earlier, $N_1 \le N_3 \le \cdots \le N_{L-1} = N$. Early in generation, with few occupancy codes unmasked, the predictions are least certain, so occupancy codes unmasked in the same step are most likely to be inconsistent with each other. We therefore use $\rho > 1$, so that κ_l grows slowly at first and quickly later.

Each step k does three things in order. First, for $k \ge 1$, each level remasks its oldest unmasked occupancy codes, so that its fraction of unmasked occupancy codes goes back to about $\kappa_l(k - 1)$. When an occupancy code is remasked, the cells that it marked as occupied are no longer known to be occupied. As in Section 4.1, the occupancy codes of all cells inside them are then removed from $\mathbf{x_t}$. Second, the network predicts all masked occupancy codes in $\mathbf{x_t}$, with each t_l set to the fraction of unmasked occupancy codes at level l. Third, each level unmasks masked occupancy codes chosen at random, so that its fraction rises to about $\kappa_l(k + 1)$. To unmask an occupancy code, we sample its value from the probabilities predicted in the second step, at temperature τ. At $\tau = 0$, we take the most probable value. Under the schedule, a very large number of occupancy codes at level $L - 1$ would be unmasked in step $N - 1$, and these would likely be inconsistent with each other. To alleviate this, we add a few extra steps after step $N - 1$ and unmask these occupancy codes evenly over step $N - 1$ and the extra steps. The extra steps do not remask. See Section B in the appendix for details.

5. Experiments

We evaluate our framework on image-conditioned generation (Section 5.1) and unconditional generation (Section 5.2), measure its efficiency (Section 5.3), and ablate its designs (Section 5.4). All shapes are represented by their 64^3 surface occupancy, the finest level of a hierarchy with $L = 4$. We also show shapes generated at a higher

resolution of 256^3 by our framework with $L = 6$ in Section A of the appendix. We run all baselines in Sections 5.1 and 5.2 with their official weights and settings, and evaluate all methods on the same test data with the same metrics in each task. Details of the data and metrics are provided in the appendix.

5.1. Image-conditioned generation

Data. We train on 174,528 assets of TRELLIS-500K [30] and test on 1,000 objects of Toys4k [23], which are unseen during training. The condition is a rendered view of each asset.

Baselines. We compare with TRELLIS.2 [31], the state of the art, and with TRELLIS [30], DVD [32], Direct3D-S2 [29], TripoSG [10] and TripoSR [24]. Voxels are often used to generate meshes, so we evaluate both the voxels and the meshes of each method. TRELLIS, DVD, TRELLIS.2, our framework and Direct3D-S2 generate 64^3 voxels. For the first four, we obtain the meshes by passing the voxels through the mesh stage of TRELLIS.2, which generates a mesh from 64^3 voxels. Direct3D-S2 refines its voxels into a mesh with its own second stage, and we evaluate this mesh. TripoSG and TripoSR generate meshes directly, and their voxel metrics use these meshes voxelized at 64^3.

Metrics. We first align each generated shape to its ground truth in orientation and scale, and then compare it with the ground truth. On the voxels, we report the voxel Chamfer distance (CD_V) and the voxel IoU, which measure geometric accuracy. A few misplaced voxels barely change these two metrics but produce visible artifacts, so we also report the LPIPS (LPIPS_V) and DINOv2 [17] similarity (DINO_V) between renderings of the generated and ground-truth voxels, which measure perceptual similarity. On the meshes, we report (i) CD_shell, the Chamfer distance between the outer surfaces, which measures geometric accuracy, (ii) LPIPS_N and CLIP_N [18] on rendered normal maps, which measure perceptual similarity, and (iii) the Uni3D [38] similarity of point clouds sampled on the meshes, which measures semantic similarity.

Results. As shown in Table 2, our framework achieves the best value on every voxel metric. Compared with TRELLIS.2, its voxels have an 11% lower Chamfer distance and an IoU of 0.416 against 0.402, and their renderings have the highest perceptual similarity to the ground truth. Since TRELLIS, DVD, TRELLIS.2 and our framework share one mesh stage, their meshes differ only in the voxels they start from. Our voxels thus lead to a 22% lower CD_shell than those of TRELLIS.2 and to the lowest LPIPS_N (Figure 3).

5.2. Unconditional generation

Data. We use the chair, airplane, table and rifle categories of ShapeNet [4], with the training and test splits of LAS-Diffusion [37]. The test split of each category, voxelized at 64^3, serves as its reference set.

Baselines. We compare with LAS-Diffusion, OctFusion [33], OctGPT [27] and XCube [19]. XCube releases models only for chair and airplane, so we compare with it on these two categories. LAS-Diffusion generates 64^3 occupancy, and XCube generates a sparse 512^3 occupancy, which we max-pool to 64^3. OctFusion and OctGPT output a 32^3 occupancy and a mesh. We evaluate both outputs of these two methods, voxelizing the meshes at 64^3 in the same way as the reference sets. Rows that use the 32^3 occupancy are marked in the tables.

Metrics. Unconditional samples have no corresponding ground truth, so we compare the generated set of each category with its reference set. We report the 1-NNA, the coverage (COV) and the minimum matching distance (MMD) [1, 35], which measure distribution similarity, diversity and fidelity, respectively, all computed with the Chamfer distance between voxel centers. We also report FID_V on PointNet++ features of Point-E [16], and FID_VD and KID_VD on DINOv2 features of voxel renderings, which measure the distance between the generated and reference distributions in a 3D feature space and an image feature space, respectively.

Results. As shown in Table 3, our framework achieves the best value on all six metrics, averaged over the four categories. Its FID_V is 80% lower and its FID_VD is 64% lower than the best baseline values. OctFusion, OctGPT and XCube are hierarchical methods, whose coarse levels are generated with the least information about the shape and are never revisited. Our framework learns the joint distribution of all levels and outperforms OctFusion and OctGPT on the four categories and XCube on chair and airplane. Table 7 in the appendix gives the results of each category. Figure 4 shows chairs generated by our framework, with fine and complex structures.

5.3. Efficiency

We compare one training step (a forward pass, a backward pass and an optimizer update) of our network and of the voxel stage of TRELLIS.2 with the same backbone. Both run on one A100 GPU with 80 GB, with gradient checkpointing, fp32 master weights and bf16 mixed precision. As shown in Figure 5, at a batch size of 32 our framework needs 29.0 GB of GPU memory instead of 47.3 GB and has $8\times$ higher training throughput, and it can be trained with batch sizes at which TRELLIS.2 runs out of memory. Since the two networks share the backbone, these differences in memory and throughput come from the number of cells they process. TRELLIS.2 processes all 4,096 cells of its 16^3 latent grid for every shape, including those in empty space, while our network processes only the cells that have a token. In a training state $\mathbf{x_t}$, which is partly masked, 273 cells have a token on average. A complete shape, in which

Table 2. Image-conditioned generation on Toys4k (* voxels from voxelized meshes; ‡ evaluated on their own meshes). The best and second-best values of each column are in bold and underlined, respectively.

	Voxels				Meshes			
Method	$CD_V \downarrow$	IoU ↑	$LPIPS_V \downarrow$	$DINO_V \uparrow$	CD_{shell} ($\times 10^{-3}$) ↓	$LPIPS_N \downarrow$	$CLIP_N \uparrow$	Uni3D ↑
TripoSR*‡	0.01663	0.3199	0.1837	0.7813	0.781	0.2291	0.7431	0.3414
TripoSG*‡	0.01325	0.3722	0.1626	0.8523	0.525	0.1698	0.7592	0.3849
Direct3D-S2‡	<u>0.01246</u>	0.3694	0.1709	<u>0.8550</u>	0.539	0.1939	0.7600	0.3824
DVD	0.01973	0.2786	0.2119	0.7914	1.070	0.2412	0.7538	0.3718
TRELLIS	0.01328	0.3595	0.1641	0.8385	<u>0.524</u>	0.1899	0.7583	0.3799
TRELLIS.2	0.01250	<u>0.4017</u>	<u>0.1524</u>	0.8513	<u>0.524</u>	<u>0.1666</u>	<u>0.7613</u>	<u>0.3865</u>
Ours	**0.01107**	**0.4158**	**0.1395**	**0.8727**	**0.407**	**0.1621**	**0.7615**	**0.3868**

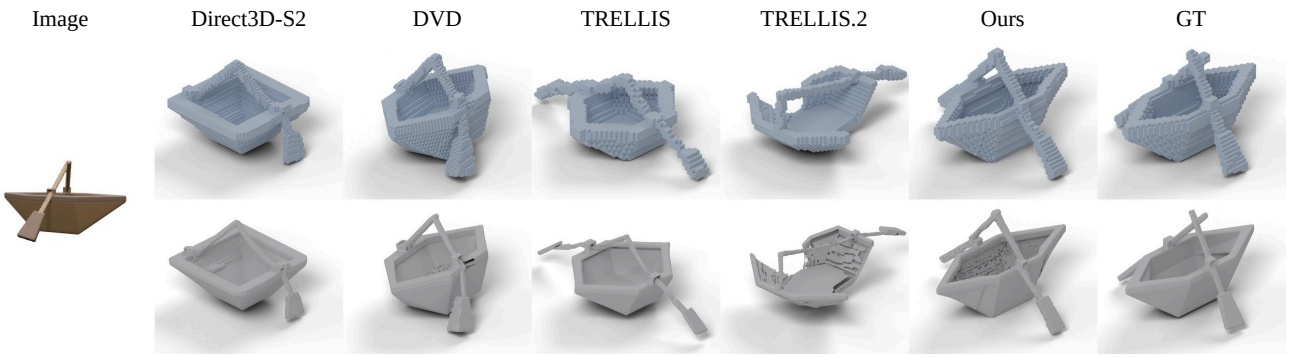


Figure 3. An example of image-conditioned generation on Toys4k. The top row shows the generated 64^3 voxels and the bottom row the meshes obtained from them. Each shape is aligned to its ground truth (GT) for display. More examples are in the appendix.

Table 3. Unconditional generation on ShapeNet, averaged over the four categories, and over chair and airplane for the last two rows. For 1-NNA, closer to 0.5 is better. MMD is in units of 10^{-3} and KID_{VD} in units of 10^{-2} († 32^3 occupancy output; "mesh" meshes voxelized at 64^3). The best and second-best values within each block are in bold and underlined, respectively.

Method	1-NNA	COV ↑	MMD ↓	$FID_V \downarrow$	$FID_{VD} \downarrow$	$KID_{VD} \downarrow$
Four categories						
LAS-Diffusion	0.782	0.365	<u>2.08</u>	63.3	82.5	16.7
OctFusion†	0.805	0.290	2.23	46.2	184.5	41.5
OctFusion (mesh)	0.779	0.310	2.23	11.1	<u>43.0</u>	<u>4.99</u>
OctGPT†	0.773	0.388	2.22	43.2	186.6	42.5
OctGPT (mesh)	<u>0.736</u>	<u>0.412</u>	2.25	<u>10.1</u>	61.0	10.4
Ours	**0.574**	**0.501**	**1.91**	**2.06**	**15.3**	**0.56**
Chair and airplane						
XCube	0.615	0.485	**1.89**	4.74	20.8	1.91
Ours	**0.576**	**0.502**	2.05	**1.79**	**15.3**	**0.54**

no occupancy code is masked as at the end of generation, has 579 on average over the Toys4k test set. One forward pass of our network therefore takes on average about $1/15$ of the FLOPs of the voxel stage of TRELLIS.2 (0.82 vs. 12.2 TFLOPs).

To measure the cost of full training, we train our framework on four H100 GPUs with a batch size of 256. TRELLIS and TRELLIS.2 are trained with the same batch size on 64 A100 GPUs and 32 H100 GPUs, respectively, 16 and 8 times as many GPUs as we use. After about two days of training (80K steps), our framework surpasses TRELLIS, which is trained for 400K steps, on every metric of Section 5.1. After about two and a half days, it matches TRELLIS.2 (Table 9 in the appendix), whose paper does not report the number of training steps. Our framework also generates the voxels faster than the voxel stage of TRELLIS.2 (Table 6 in the appendix).

Table 4. Ablation of two-level prediction and joint hierarchical modeling on Toys4k. The first two variants generate the hierarchy level by level. CD_{shell} is in units of 10^{-3}. The best value of each column is in bold.

Variant	$CD_V \downarrow$	IoU ↑	$CD_{shell} \downarrow$	$LPIPS_N \downarrow$	Uni3D ↑
Single-level	0.02150	0.238	1.280	0.2566	0.3508
Two-level	0.01941	0.264	1.096	0.2428	0.3575
Ours	**0.01506**	**0.334**	**0.744**	**0.2019**	**0.3820**

5.4. Ablation studies

We conduct ablation studies on image-conditioned generation on Toys4k.

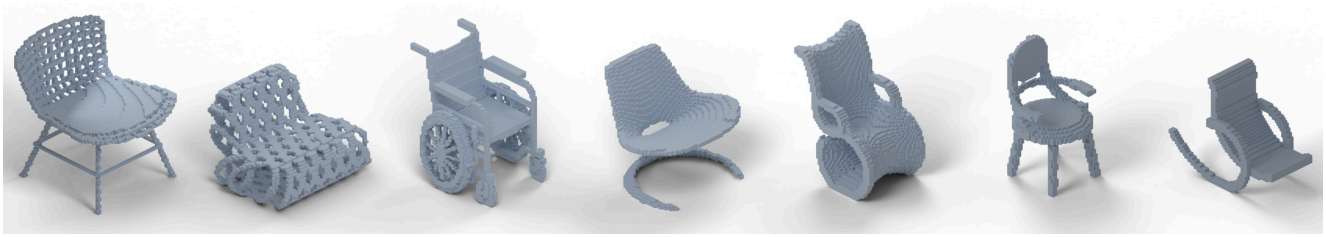


Figure 4. Unconditional generation of chairs by our framework on ShapeNet.

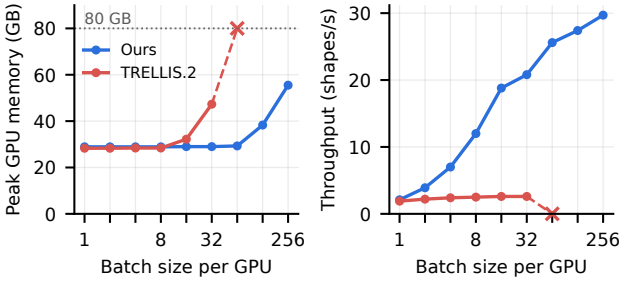


Figure 5. Peak GPU memory and training throughput of our network and of the voxel stage of TRELLIS.2, with the same backbone, at each batch size on one A100 GPU with 80 GB. A cross marks the batch size at which TRELLIS.2 runs out of memory.

Table 5. Ablation of the unmasking strategy on Toys4k. The best and second-best values of each column are in bold and underlined, respectively.

Sampler	$CD_V \downarrow$	IoU ↑
MDLM [21]	0.01162	0.4035
ReMDM [25]	<u>0.01115</u>	0.4085
ReMDM-conf [25]	0.01116	<u>0.4139</u>
MaskGIT [5]	0.01243	0.3987
Ours	**0.01107**	**0.4158**

Two-level prediction and joint hierarchical modeling. We compare our framework with two variants in Table 4. The single-level variant uses single-level prediction (Section 4.2) and the two-level variant uses two-level prediction, and both generate the hierarchy level by level. By contrast, ours extends the two-level variant with *joint* hierarchical modeling. All three share the network architecture and the training data, and are trained for the same number of steps, fewer than those used for our model in Section 5.1. As shown in Table 4, the two-level variant improves every metric over the single-level variant. Compared with the two-level variant, our framework lowers the voxel Chamfer distance by 22% and CD_{shell} by 32%, and improves every metric. In addition, two-level prediction makes training faster, since the network processes fewer tokens. With both variants on four A100 GPUs at a batch size of 256, one training step of the two-level variant takes 4.03 s, compared with 8.96 s for the single-level variant, a 2.2× speedup.

Unmasking strategy. We compare our unmasking strategy with other samplers, all applied to our model of Section 5.1. MDLM [21] does not remask. ReMDM [25] remasks unmasked occupancy codes chosen at random, and its confidence-based variant ReMDM-conf [25] remasks the less confident ones with a higher probability. MaskGIT [5] unmasks the most probable occupancy codes first and does not remask. Unlike these samplers, our strategy chooses the occupancy codes to remask by when they were unmasked. An occupancy code unmasked earlier is predicted given fewer finer occupancy codes in nearby cells, so its prediction is less certain. Our strategy therefore remasks the oldest unmasked occupancy codes, so that they are revisited once finer occupancy codes are unmasked. As shown in Table 5, our strategy gives the lowest voxel Chamfer distance and the highest IoU among all samplers.

6. Limitations and future work

A limitation of our model is the size of its training set. Due to limited training resources, we train it on about 175K assets, less than a quarter of the roughly 800K assets used to train TRELLIS.2 [31]. While our model, with the data augmentation described in the appendix, still outperforms TRELLIS.2 in generalization to the unseen objects of Toys4k, we expect more training data to further improve it. Besides, like other masked discrete flow matching models, our model unmasks multiple occupancy codes independently in one step, which can make them inconsistent with each other [3, 8, 12, 34]. Our unmasking strategy is designed to reduce such inconsistencies, and exploring better unmasking strategies may further improve the quality of the generated voxels.

In future work, we plan to apply our framework to generating large scenes with multiple objects and long sequences of 3D shapes. Both contain far more voxels than a single object, and our framework keeps their training lightweight by processing only the cells inside occupied cells.

7. Conclusion

We presented JoVox, a discrete flow matching framework that learns the joint distribution of all levels of the voxel hierarchy, so that finer cells take part in deciding coarse cells. Within this framework, two-level prediction halves the number of levels that the network processes and yields voxels of higher fidelity, and our unmasking strategy

revisits early decisions once finer cells are generated. Our framework outperforms the baselines on unconditional and image-conditioned generation, while its training is far cheaper than that of TRELLIS.2. **Our results thus show that voxels can be generated with both clearly improved fidelity and drastically improved efficiency.**

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] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021. 2, 3

[3] Victor Besnier, Mickael Chen, David Hurych, Eduardo Valle, and Matthieu Cord. Halton scheduler for masked generative image transformer. In *International Conference on Learning Representations*, pages 83027–83047, 2025. 2, 3, 5, 8

[4] 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. 6

[5] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11305–11315. IEEE, 2022. 2, 8

[6] Kangle Deng, Hsueh-Ti Derek Liu, Yiheng Zhu, Xiaoxia Sun, Chong Shang, Kiran S Bhat, Deva Ramanan, Jun-Yan Zhu, Maneesh Agrawala, and Tinghui Zhou. Efficient autoregressive shape generation via octree-based adaptive tokenization. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11685–11696. IEEE, 2025. 2

[7] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37:133345–133385, 2024. 2, 3

[8] Satoshi Hayakawa, Yuhta Takida, Masaaki Imaizumi, Hiromi Wakaki, and Yuki Mitsufuji. Distillation of discrete diffusion through dimensional correlations. *arXiv preprint arXiv:2410.08709*, 2024. 2, 5, 8

[9] Moritz Ibing, Gregor Kobsik, and Leif Kobbelt. Octree transformer: Autoregressive 3D shape generation on hierarchically structured sequences. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2698–2707. IEEE, 2023. 1, 2, 4

[10] Yangguang Li, Zi-Xin Zou, Zexiang Liu, Dehu Wang, Yuan Liang, Zhipeng Yu, Xingchao Liu, Yuan-Chen Guo, Ding Liang, Wanli Ouyang, et al. TripoSG: High-fidelity 3D shape synthesis using large-scale rectified flow models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025. 2, 6

[11] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024. 2, 3

[12] Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. In *International Conference on Learning Representations*, pages 88953–88979, 2025. 2, 5, 8

[13] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 11

[14] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023. 2

[15] Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed: Dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025. 3

[16] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-E: a system for generating 3D point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022. 6

[17] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision, 2023. 6

[18] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 6

[19] Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. XCube: Large-scale 3D generative modeling using sparse voxel hierarchies. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4209–4219. IEEE, 2024. 2, 6

[20] Liran Ringel, Ameen Ali, and Yaniv Romano. Dependency-guided parallel decoding in discrete diffusion language models. *arXiv preprint arXiv:2604.02560*, 2026. 3

[21] Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024. 2, 3, 8

[22] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. DINOv3. *arXiv preprint arXiv:2508.10104*, 2025. 5

[23] Stefan Stojanov, Anh Thai, and James M Rehg. Using shape to categorize: Low-shot learning with an explicit shape bias.

In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1798–1808. IEEE, 2021. 6

[24] Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. TripoSR: Fast 3D object reconstruction from a single image. *arXiv preprint arXiv:2403.02151*, 2024. 2, 6

[25] Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *Advances in Neural Information Processing Systems*, 38:147282–147339, 2025. 3, 5, 8

[26] Hanxiao Wang, Ying-Tian Liu, Yuan-Chen Guo, Qi-Yuan Feng, Zi-Xin Zou, Ding Liang, Biao Zhang, and Yan-Pei Cao. Nexus: Native mesh generation with diffusion. *ACM Transactions on Graphics (TOG)*, 45(4):1–14, 2026. 2

[27] Si-Tong Wei, Rui-Huan Wang, Chuan-Zhi Zhou, Baoquan Chen, and Peng-Shuai Wang. OctGPT: Octree-based multiscale autoregressive models for 3D shape generation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pages 1–11, 2025. 1, 2, 6

[28] Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *International Conference on Learning Representations*, pages 57027–57051, 2026. 3

[29] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, Yao Yao, et al. Direct3d-s2: Gigascale 3d generation made easy with spatial sparse attention. *Advances in Neural Information Processing Systems*, 38:170778–170804, 2026. 2, 6

[30] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D latents for scalable and versatile 3D generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21469–21480. IEEE, 2025. 1, 2, 6, 11

[31] Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, et al. Native and compact structured latents for 3D generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14419–14429, 2026. 1, 2, 5, 6, 8

[32] Zhengrui Xiang, Jiaqi Wu, Fupeng Sun, Heliang Zheng, and Yingzhen Li. DVD: Discrete voxel diffusion for 3D generation and editing. *arXiv preprint arXiv:2605.07971*, 2026. 2, 6

[33] 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. In *Computer Graphics Forum*, page e70198. Wiley Online Library, 2025. 1, 2, 6

[34] Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. In *International Conference on Learning Representations*, pages 33769–33789, 2025. 2, 5, 8

[35] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. PointFlow: 3D point cloud generation with continuous normalizing flows. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4540–4549. IEEE, 2019. 6

[36] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. *ACM Trans. Graph.*, 42(4), 2023. 5

[37] Xin-Yang Zheng, Hao Pan, Peng-Shuai Wang, Xin Tong, Yang Liu, and Heung-Yeung Shum. Locally attentional SDF diffusion for controllable 3D shape generation. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4), 2023. 2, 6

[38] Junsheng Zhou, Jinsheng Wang, Baorui Ma, Yu-Shen Liu, Tiejun Huang, and Xinlong Wang. Uni3D: Exploring unified 3D representation at scale. In *International Conference on Learning Representations*, pages 46766–46782, 2024. 6

A. Generation at 256^3

Figure 6 shows shapes generated at 256^3 from input images by our framework with $L = 6$ (Section B).

B. Implementation details

The experiments in Section 5 use $L = 4$, so the finest level is a grid of 64^3 cells. The tokens come from the cells at level 0 and the occupied cells at level 2, with grids of 4^3 and 16^3 cells, and the occupancy codes are at levels 1 and 3. The 256^3 shapes in Section A and on the project webpage are generated by a model with $L = 6$, where the tokens come from the cells at level 0 and the occupied cells at levels 2 and 4, and the occupancy codes are at levels 1, 3 and 5. For this model, we add the level embedding $\mathrm{emb}_{\mathrm{level}}(4)$ and the embedding emb_t^5 of t_5, initialize all other weights from the model with $L = 4$, and train it for 32K further steps on four GPUs.

The transformer has 1.54B parameters in 30 blocks of width 1536 with 12 heads. The image encoder is a frozen DINOv3 ViT-L/16 at a resolution of 512×512. The condition is dropped with probability 0.1 during training, and classifier-free guidance at scale 1.5 is applied to the logits during sampling.

We train with AdamW [13] at a learning rate of 10^{-4} and a batch size of 256 on four GPUs, and keep an exponential moving average of the weights. For data augmentation, the condition image is randomly perturbed in brightness, saturation, contrast, hue and gamma, and the occupancy is rotated by a random multiple of 90° about the vertical axis. We draw $t_l = v_l^2$ with $v_l \sim \mathcal{U}(0, 1)$.

In the unmasking strategy of Section 4.5, we remask at the levels one after another at each step, from the coarsest to the finest, so that we count the unmasked occupancy codes at a finer level only after remasking at the coarser levels has removed some of them. At step $k \geq 1$, at each level l with $\kappa_l(k) < 1$, we remask up to $\mathrm{round}\big((1-\kappa_l(k-1)/\kappa_l(k))\, n\big)$ occupancy codes, rounded half to even, where n is the number of all unmasked occupancy codes at level l. We do not remask an occupancy code unmasked at step $k-1$, since no prediction has yet been made given it, nor an occupancy code unmasked in a step in which no other occupancy code, at any level, was unmasked. Among the other unmasked occupancy codes, we remask those last unmasked earliest, with ties broken at random, or all of them if there are fewer. For the prediction, we set t_l to 0 at a level that has no occupancy codes. After the prediction, we unmask only occupancy codes predicted at this step, and occupancy codes added after an unmasking are first predicted at the next step and are not counted below. At a level l with $\kappa_l(k-1) = 1$, we unmask all of its masked occupancy codes. At any other level l with masked occupancy codes, we unmask the proportion $\big(\kappa_l(k+1) - \kappa_l(k-1)\big)/\big(1 - \kappa_l(k-1)\big)$ of them, rounded up and at least one, chosen at random, where $\kappa_l(-1) = 0$. From step $N-1$ on, we instead unmask the proportion $1/(N + N_{\mathrm{f}} - k)$ of the masked occupancy codes at level $L-1$ at step k, also rounded up and at least one, so that they are unmasked evenly over step $N-1$ and N_{f} extra steps, and we do not remask in the extra steps.

For image-conditioned generation, the sampler uses $N = 72$, $N_{\mathrm{f}} = 8$, $N_1 = 0.8N$, $\rho = 2$ and $\tau = 0$, so a generation takes 80 steps. For unconditional generation, it uses $\tau = 1$, with $N = 288$ for chair and table and $N = 72$ for airplane and rifle.

Table 6 reports the time and the peak GPU memory of generating the voxels. Our framework generates the voxels faster than the voxel stage of TRELLIS.2 at every batch size.

Table 6. Time per shape and peak GPU memory of generating the voxels for image-conditioned generation on one A100 GPU with 80 GB, for our framework and the voxel stage of TRELLIS.2 (OOM, out of memory). The time includes image encoding, with 80 sampling steps for our framework and the default 12 for TRELLIS.2.

	Ours		TRELLIS.2	
Batch	Time (s)	Memory (GB)	Time (s)	Memory (GB)
8	2.02	6.1	2.30	5.7
16	1.77	7.9	2.24	11.2
32	1.80	11.3	2.21	22.2
64	1.67	17.8	2.20	44.2
128	1.62	28.9	OOM	

C. Data and evaluation details

Data. The 174,528 training assets come from the Sketchfab, ABO, 3D-FUTURE and HSSD parts of TRELLIS-500K [30], about 35% of the full dataset. The condition of a training asset is a rendered view at an elevation between $-10°$ and 30°, and each Toys4k test object has one condition view at a similar elevation. On ShapeNet, each category is evaluated separately on its test split, and its reference set has as many shapes as the generated set.

Alignment. Methods and datasets differ in their orientation and scale conventions. We therefore align each generated shape to its ground truth by picking the one of the 24 axis-aligned rotations that minimizes the voxel Chamfer distance and refining it with a scale-aware ICP.

Voxel metrics. For $\mathrm{LPIPS_V}$ and $\mathrm{DINO_V}$, we render the aligned voxels and the ground-truth voxels from six views and average the LPIPS and the cosine similarity of DINOv2 features over the views. For Direct3D-S2, the voxel metrics use the 64^3 voxels of its dense first stage, and for TripoSG



Figure 6. Image-conditioned generation at 256^3. Each input image is shown in the box at the top left of the generated 256^3 voxels, which are colored by the texture stage of TRELLIS.2.

Table 7. Unconditional generation on each ShapeNet category (test splits). For 1-NNA, closer to 0.5 is better. MMD is in units of 10^{-3} and KID_{VD} in units of 10^{-2}. Rows marked † use the 32^3 occupancy the method outputs, and rows marked "mesh" use the method's meshes voxelized at 64^3. XCube releases models only for chair and airplane. The best and second-best values of each column within each category are in bold and underlined, respectively.

	Chair						Airplane					
Method	1-NNA	COV ↑	MMD ↓	FID_{V} ↓	FID_{VD} ↓	KID_{VD} ↓	1-NNA	COV ↑	MMD ↓	FID_{V} ↓	FID_{VD} ↓	KID_{VD} ↓
LAS-Diffusion	0.697	0.432	3.40	30.67	57.6	8.77	0.881	0.357	1.06	73.07	130.3	28.54
OctFusion†	0.689	0.406	3.59	22.49	106.8	15.63	0.902	0.261	1.17	51.04	305.2	67.86
OctFusion (mesh)	0.671	0.412	3.68	5.01	29.5	2.70	0.863	0.268	1.00	12.64	51.2	6.41
OctGPT†	0.715	0.397	3.74	25.38	139.6	26.30	0.872	0.365	1.09	47.83	312.3	68.88
OctGPT (mesh)	0.697	0.401	3.90	9.36	77.8	14.74	0.821	0.361	0.93	12.78	27.9	2.21
XCube	**0.541**	0.503	**3.03**	2.85	24.9	2.74	0.689	0.467	**0.75**	6.62	16.6	1.07
Ours	0.597	**0.504**	3.31	**2.01**	**19.1**	**0.88**	**0.554**	**0.499**	0.78	**1.57**	**11.5**	**0.20**
	Table						Rifle					
LAS-Diffusion	0.727	0.401	2.92	58.25	80.5	14.43	0.823	0.269	0.95	91.18	61.4	15.20
OctFusion†	0.788	0.312	3.13	34.45	86.5	10.35	0.839	0.181	1.02	76.75	239.4	72.30
OctFusion (mesh)	0.769	0.327	3.19	12.39	64.3	7.24	0.813	0.232	1.06	14.20	27.1	3.61
OctGPT†	0.724	0.451	3.15	35.21	90.1	12.58	0.780	0.339	0.88	64.31	204.3	62.07
OctGPT (mesh)	0.684	0.480	3.24	10.65	118.1	21.86	0.742	0.406	0.92	7.67	20.2	2.60
Ours	**0.562**	**0.520**	**2.73**	**1.73**	**18.4**	**0.83**	**0.581**	**0.482**	**0.83**	**2.92**	**12.0**	**0.32**

and TripoSR, they use their voxelized meshes.

Mesh metrics. The meshes of Direct3D-S2, TripoSG and TripoSR are aligned to the ground truth in the same way. CD_{shell} is the Chamfer distance between the outer surfaces of the two meshes. LPIPS_{N} and CLIP_{N} compare normal maps rendered from four views. Uni3D is the similarity between the Uni3D embeddings of point clouds sampled on the two meshes.

ShapeNet metrics. FID_{V} is computed on points of the voxels. FID_{VD} and KID_{VD} use renderings of the voxels from six views.

D. Additional ablations

This section measures two choices of Section B on the first 200 objects of Toys4k, with the unmasking strategy of Section 4.5.

Exponential moving average of the weights. For our

model, the moving average improves CD_V from 0.01215 to 0.01193 and CD_{shell} from 0.503×10^{-3} to 0.446×10^{-3}. It also improves $LPIPS_N$ from 0.1758 to 0.1714.

Augmentation. We train the same model with and without the augmentation of Section B and compare the two after the same number of training steps (Table 8). Augmentation improves CD_V, CD_{shell}, $LPIPS_N$, $CLIP_N$ and Uni3D.

Table 8. Training with and without augmentation, compared after the same number of training steps on the first 200 objects of Toys4k. CD_{shell} is in units of 10^{-3}. The better value of each column is in bold.

Aug.	$CD_V \downarrow$	IoU $\uparrow$	$CD_{shell} \downarrow$	$LPIPS_N \downarrow$	$CLIP_N \uparrow$	Uni3D $\uparrow$
	0.01280	**0.3923**	0.568	0.1858	0.7588	0.3844
✓	**0.01271**	0.3790	**0.524**	**0.1797**	**0.7608**	**0.3847**

E. Per-category results on ShapeNet

Table 7 gives the results of Table 3 in the main paper for each category.

F. Results after shorter training

Table 9 compares our framework at the two training lengths of Section 5.3 with TRELLIS and TRELLIS.2 on Toys4k.

G. Qualitative comparison

Figures 7, 8 and 9 compare the shapes generated from the same image by each method on Toys4k.

Table 9. Our framework after about two days and two and a half days of training on four H100 GPUs, compared with TRELLIS and TRELLIS.2 on Toys4k. Each of our rows is compared with the baseline above it. Steps are counted in batches of 256 shapes.

	Voxels				Meshes			
Method	$CD_V \downarrow$	IoU $\uparrow$	$LPIPS_V \downarrow$	$DINO_V \uparrow$	CD_{shell} $(\times 10^{-3}) \downarrow$	$LPIPS_N \downarrow$	$CLIP_N \uparrow$	Uni3D $\uparrow$
TRELLIS	0.01328	0.3595	0.1641	0.8385	0.524	0.1899	0.7583	0.3799
Ours, about two days (80K steps)	0.01221	0.3893	0.1485	0.8614	0.497	0.1737	0.7600	0.3851
TRELLIS.2	0.01250	0.4017	0.1524	0.8513	0.524	0.1666	0.7613	0.3865
Ours, about two and a half days (104K steps)	0.01137	0.4035	0.1424	0.8692	0.421	0.1668	0.7612	0.3859

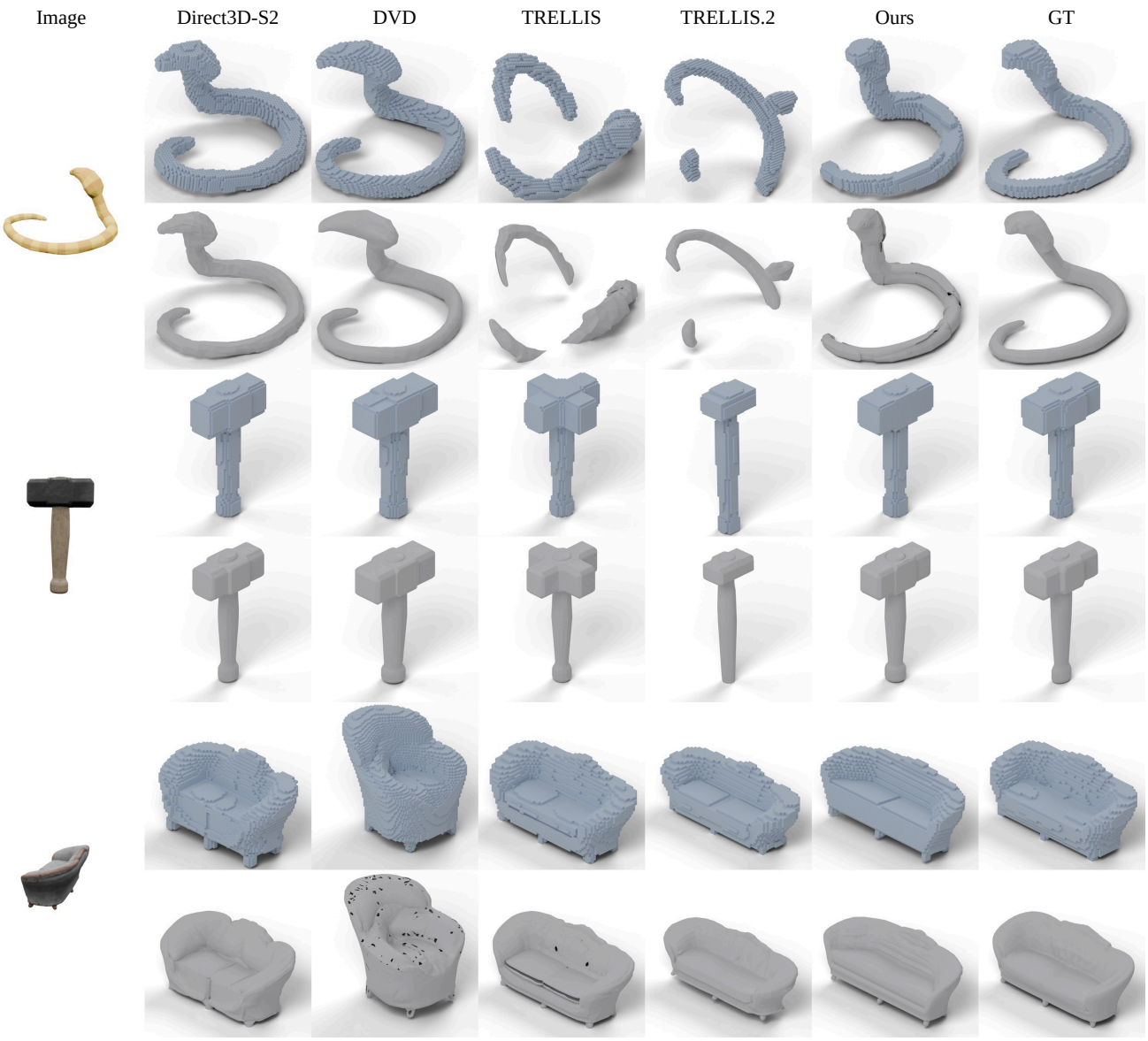


Figure 7. Image-conditioned generation on Toys4k. For each input image, the top row shows the generated 64^3 voxels and the bottom row the meshes obtained from them as in Section 5.1. The ground truth (GT) shows the voxels and the mesh of the shape. Each shape is aligned to its ground truth for display.

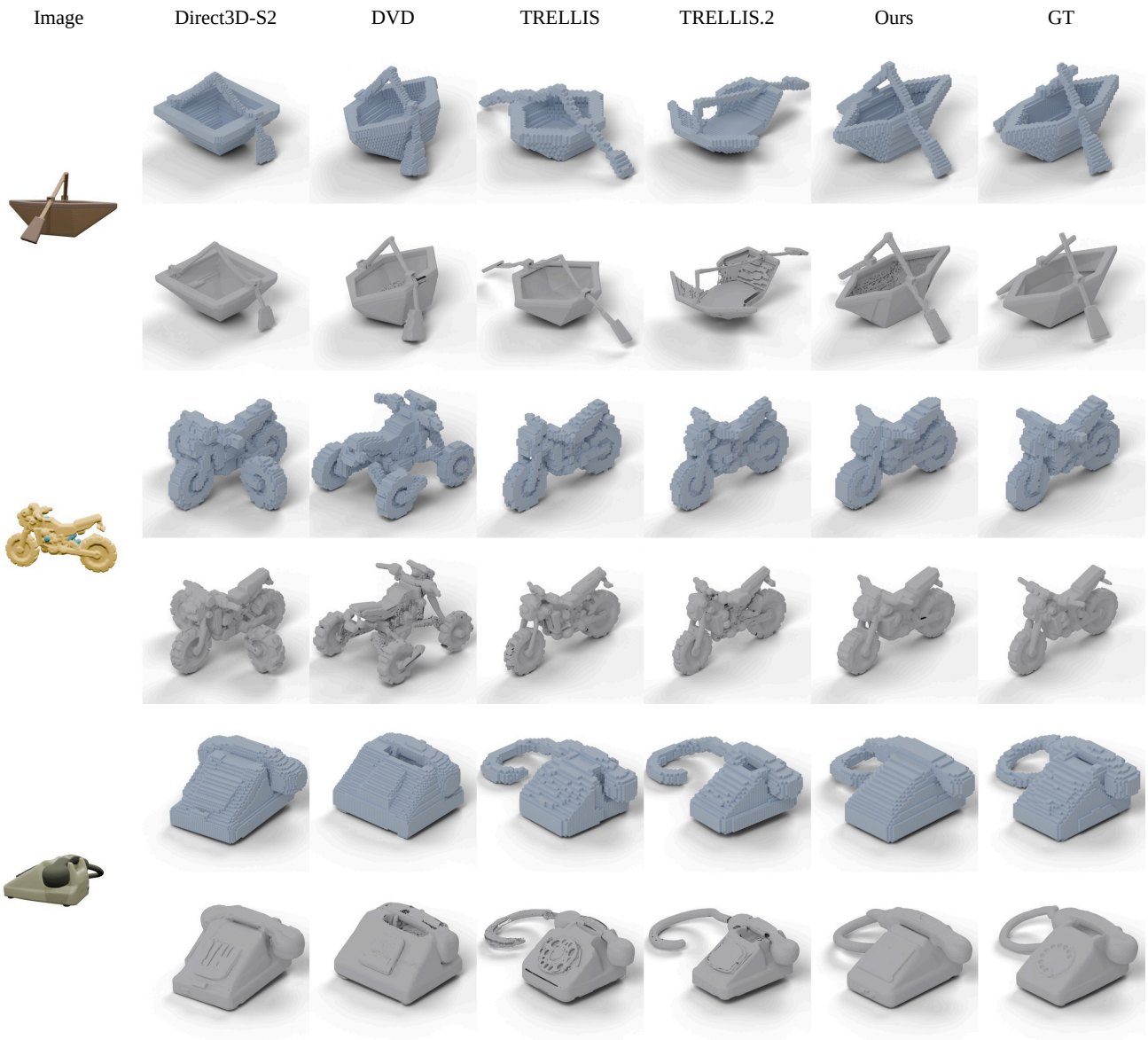


Figure 8. Image-conditioned generation on Toys4k, continued from Figure 7.

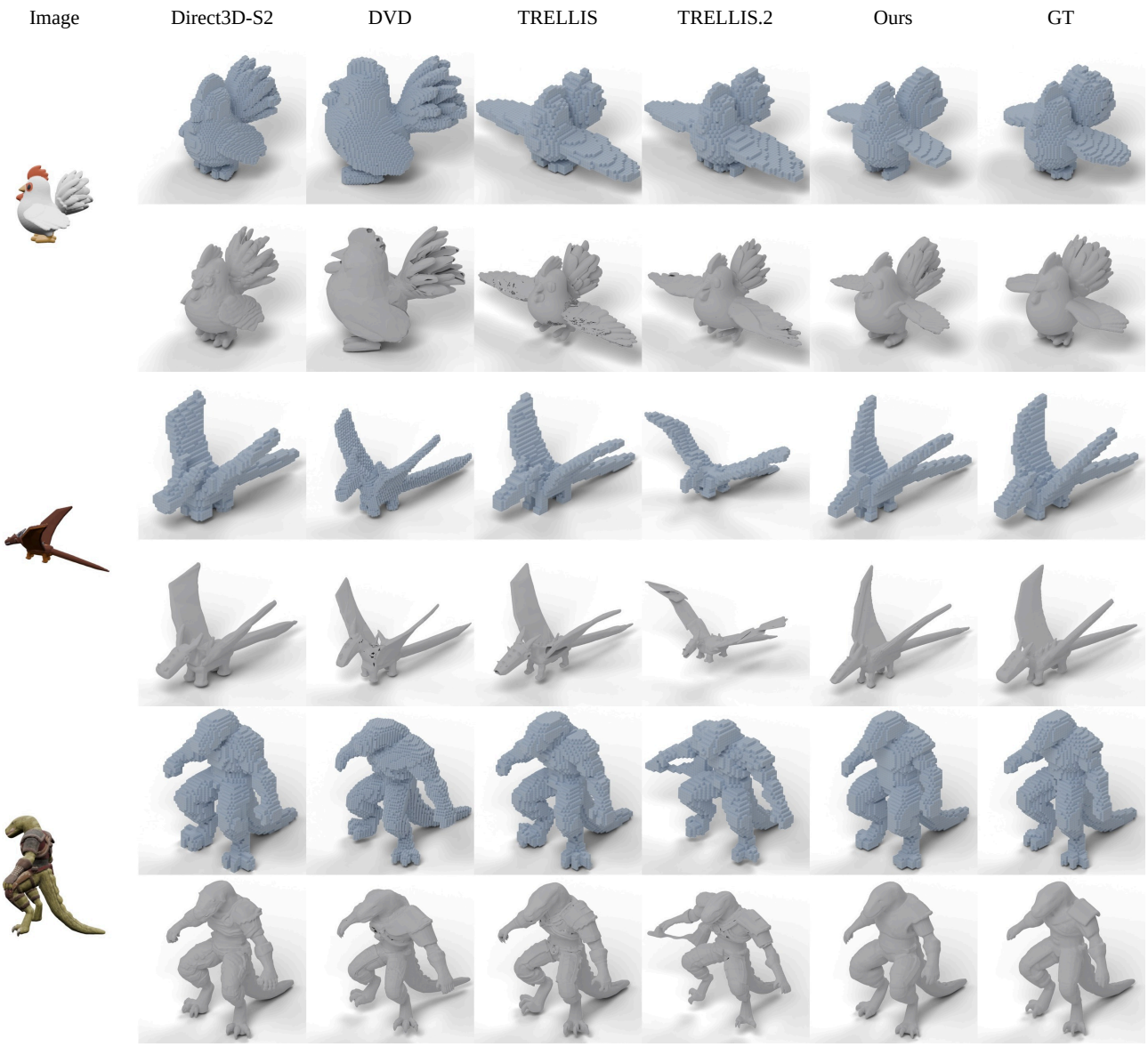


Figure 9. Image-conditioned generation on Toys4k, continued from Figure 8.