Supplementary Material for DeepMesh: Auto-Regressive Artist-mesh Creation with Reinforcement Learning

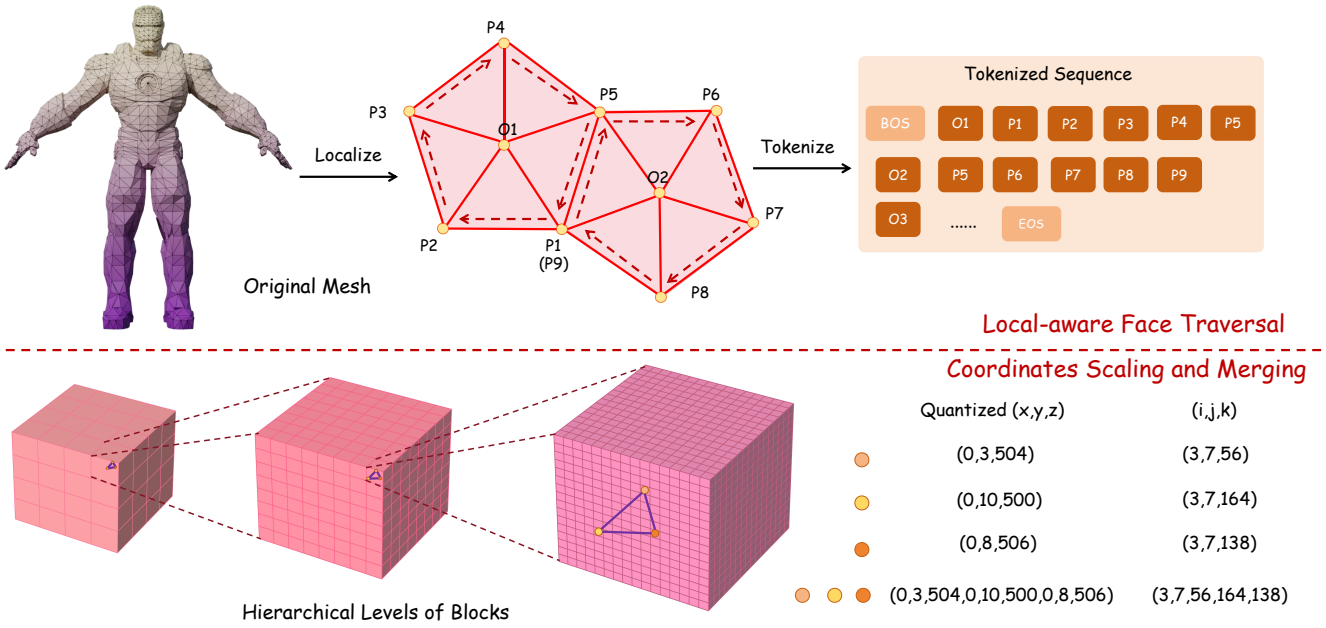


Figure 1. **Details of our tokenization algorithm.** We first traverse mesh faces by dividing them into patches according to their connectivity and quantize each vertex of faces into r bins (in our setting $r = 512$).Then we partition the whole coordinate system into three hierarchical levels of blocks and index the quantized coordinates as offsets within each block. We merge the index of neighbor vertices if they have the identical values.

A. Details of Tokenization Algorithm

In this section, we detail our improved tokenization algorithm. As illustrated in Figure 1, we first traverse mesh faces to reduce redundancy in the vanilla mesh representation. Specifically, we divide mesh faces into multiple local patches according to their connectivity similar to [6]. Each local patch is formed by grouping a central vertex O with its adjacent vertices $P_{1:n}$, which are organized based on their connectivity order:

$$L_O = (O, P_1, P_2, \cdots, P_n) \tag{1}$$

This organization helps maintain local mesh connectivity by explicitly encoding edge-sharing relationships between adjacent faces. To find each center vertex, we begin by sorting all the unvisited faces. Next, we select the first unvisited face and choose the vertex connected to the most unvisited faces as the center. Then, we iteratively traverse the neighboring vertices within the center's unvisited faces, expanding the local patch by adding adjacent vertices that connect to the current patch. Once the patch is complete, we mark all its faces as visited. We repeat the process above until every face is visited.

After the local-wise face traversal, we normalize and quantize each vertex in faces and flatten them in XYZ order. With a resolution of r, coordinates of each vertex are quantized into $[0, r-1]$ (in our setting, $r = 512$). The coordinates of all vertices are then concatenated to form a complete sequence of tokens. To further reduce the sequence length, we partition the whole coordinate system into three hierarchical levels of blocks and index the quantized coordinates as offsets within each block, as shown in Figure 1. The volume of each block is A, B and C respectively. In our setting, $A = 4$, $B = 8$ and $C = 16$. We scale quantized Cartesian coordinate (x, y, x) of each vertex into (i, j, k)

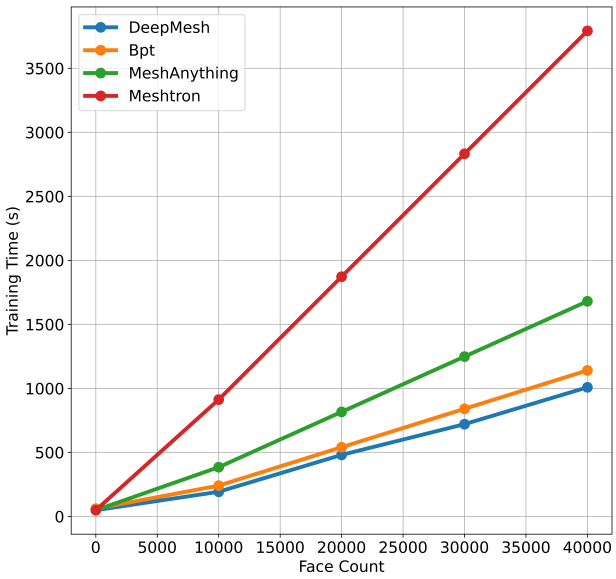


Figure 2. **Comparison with other tokenization algorithms in training efficiency.** We integrate all tokenization algorithms into our model architecture and train them on a dataset of 80 meshes for each face count category (10K, 20K, 30K, 40K). Our method achieves the fastest training time across all face count categories, demonstrating superior training efficiency.

by:

$$\begin{aligned} i =& (x \mid B \cdot C) \cdot A^2 + (y \mid B \cdot C) \cdot A + (z \mid B \cdot C) \\ j =& (x\%B \cdot C \mid C) \cdot B^2 + (y\%B \cdot C \mid C) \cdot B \\ &+ (z\%B \cdot C \mid C) \\ k =& (x\%C) \cdot C^2 + (y\%C) \cdot C + z\%C \end{aligned} \tag{2}$$

As the coordinates are sorted, it is common for neighbor vertices to share the same offset in block. Therefore, we merge the adjacent (i, j, k) if they have the identical values to save more length. Specifically, for vertices $v_i, v_2, \cdots, v_n$, the sequence of their coordinate representation can be simplified as follows:

$$\begin{aligned} (v_1, v_2, \cdots, v_n) &= (i_1, j_1, k_1, i_1, j_1, k_2, \cdots, i_1, j_2, k_{s+1}) \\ &= (i_1, j_1, k_1, k_2, \cdots, k_s, j_2, k_{s+1}, \cdots, k_n) \end{aligned} \tag{3}$$

To distinguish different patches, we extend the vocabulary size of i and j for each center vertex in patches. This design eliminates the need for special tokens to separate adjacent local patches, thereby avoiding unnecessary increases in mesh sequence length.

B. More Implementation Details

B.1. Training Data Filtering Pipeline

The data in training dataset varies significantly in quality, which may lead to three primary challenges: 1. Unstructured topology that fails to meet the artist-mesh standard. 2. Fragmented data that cannot assemble into complete surfaces. 3. Overly complex structures, such as characters with tangled or messy hair geometry.

To efficiently filter out low-quality data, we propose a multi-stage data filtering pipeline:

(1) First of all, We remove meshes with a mesh.area metric below 1 to filter out the fragmented data.

(2) Subsequently, We construct a high-quality subset of the training data and perform low-cost pretraining on it to build a baseline model. Specifically, We train a 0.5B model on Objaverse [3].

(3) Finally, the pretrained model is then used to evaluate the remaining data by computing test losses. Evaluation is mainly conducted on the private data in our training dataset. Samples with a test loss greater than 1 are discarded.

After filtering out the poor-quality samples, the resulting dataset consists of approximately 500k meshes, with an average face count of 8k.

B.2. Preference Pair Construction Pipeline

The point clouds in the preference pair construction are from both the training set and a manually curated test set to ensure diversity for learning human preferences. However, generating high-poly meshes is extremely time-consuming. For example, our full-scale model requires around 10 minutes to generate a single mesh with over 30K faces. Therefore, it is crucial to filter out overly complex meshes when constructing the DPO dataset to improve efficiency and feasibility. Moreover, to maintain representativeness, overly simple meshes must also be excluded. We follow a similar data-filtering approach in Section B.1, removing samples with fewer than 5,000 faces as well as those with extremely high or low test loss. Based on the remaining curated subset, we construct the preference pair dataset for post-training. For each point cloud, we generate two mesh outputs with a temperature of 0.5.

B.3. More Training Details

We respectively train a small-scale model and a large-scale model for DeepMesh, with architecture details provided in the Table 1. We train both of the models for 100k iteration steps to ensure convergence. Moreover, we employ FlashAttention and Zero2 to reduce GPU memory usage.

B.4. Hourglass Transformers

Inspired by [4, 5], we adopt the Hourglass Transformers architecture for efficient training. For hyper-parameters, we adopt the settings from [4]. Specifically, the shortening factor is set to 3, while both the down-sampling and up-sampling layers are used with the Linear layers.

	Small scale	Large scale
Parameter count	500 M	1.1B
Batch Size	9	5
Layers	21	20
Heads	10	14
d_{model}	1280	1792
d_{FFN}	5120	7168
Learning rate	$1e-4$	$1e-4$
LR scheduler	Cosine	Cosine
Weight decay	0.1	0.1
Gradient Clip	1.0	1.0

Table 1. Deepmesh's architectural and training details.

C. Additional Ablation Studies

C.1. Efficiency of Tokenization

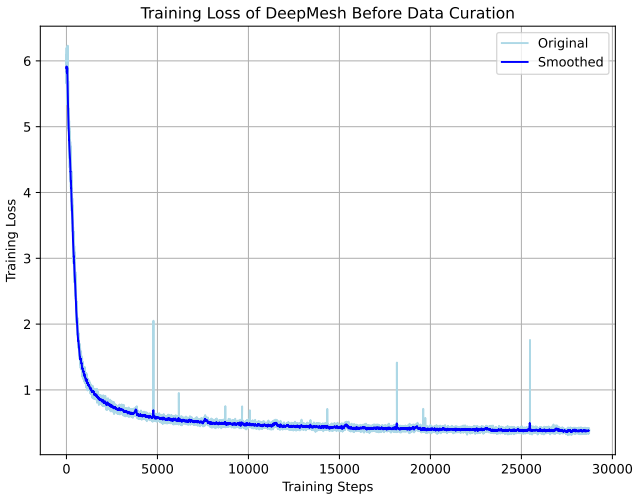
We evaluate the computational efficiency of our mesh tokenization algorithm compared to other baselines [2, 4, 6]. To ensure a fair comparison, we integrate each method's compressed mesh representation into our model while keeping all other parameters unchanged, as detailed in Table 1. For training, we use a single GPU and dynamically adjust the batch size to fully utilize available memory. We test on a dataset of 80 meshes for each face count category: 10K, 20K, 30K, and 40K faces. As shown in Figure 2, our method consistently exhibits lowest training time across all face count categories, and achieves the best training efficiency.

C.2. Data Curation

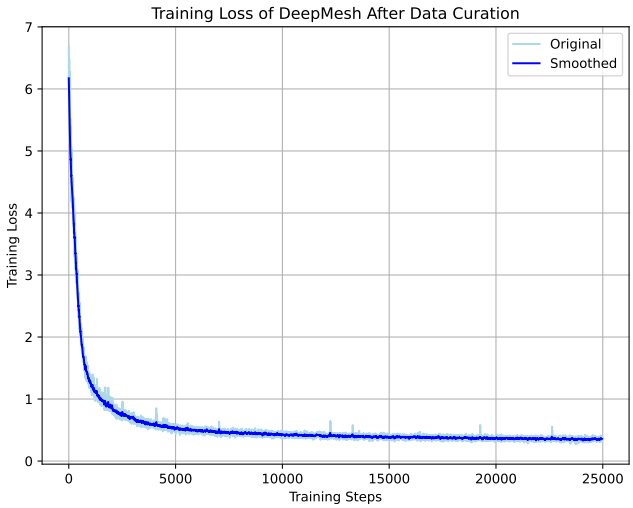
During the initial stages of training, we observe frequent spikes in the loss curve, as illustrated in Figure 3a . This suggests that certain training samples lead to irregular loss values, potentially disrupting the learning process. To address this issue, we apply the data filtering strategy outlined in Section B.1, removing low-quality samples to ensure stable training. This filtering process can mitigate the inconsistencies caused by poor mesh structures. The impact of this curation is reflected in the improved training loss curve, shown in Figure 3b.

C.3. Scaling Preference Data Pairs in DPO

We generated a total of 10,822 mesh pairs. Among them, 5,707 low-quality pairs, in which both meshes had a Chamfer Distance (CD) > 0.1, were discarded. The remaining dataset included 1,528 CD-labeled and 3,587 human-labeled pairs respectively. To evaluate the effect of data scaling, we constructed multiple subsets by sampling from the retained dataset while preserving the original ratio between CD-labeled and human-labeled pairs. The model was then post-trained on each subset containing different



(a) Before data curation



(b) After data curation

Figure 3. **Training loss before and after data curation.** Before data curation, we observe frequent loss spikes. After data curation, pre-training becomes significantly more stable.

quantities of data pairs. For an intuitive presentation, we use validation loss to evaluate the effectiveness of DPO. As shown in Figure 4, scaling up data pairs leads to a more pronounced and consistent reduction in validation loss, indicating that DPO performance improves with larger-scale datasets.

C.4. Effect of Human-annotated Preferences

Although annotation based solely on the Chamfer Distance (CD) metric offers automation and efficiency, we still incorporate human selection to more accurately capture human preferences. To evaluate the impact of human annotation, we additionally annotate the retained human-labeled mesh pairs with CD and conduct separate experiments using the dataset labeled by CD and human. As illustrated in

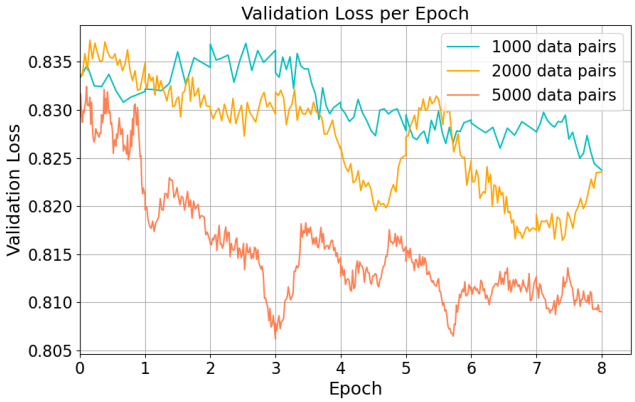


Figure 4. **Validation loss at post-training across different scale of DPO dataset.** Scaling up data pairs leads to greater reductions in validation loss, which indicates a better DPO generalization performance.

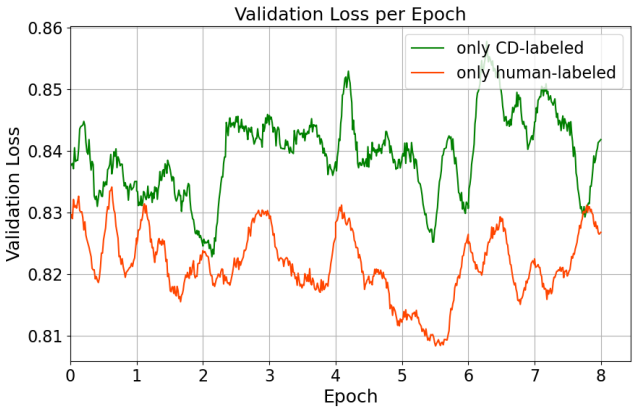


Figure 5. **Comparison of validation loss using CD-labeled and human-labeled data pairs.** The lower validation loss achieved through post-training on human-labeled data highlights the importance of human selection and insufficiency of CD-only annotation in capturing complex preferences.

Figure 5, post-training on human-labeled data yields lower validation loss and improved DPO performance. This highlights the necessity of human-annotation and limitations of relying solely on geometric metrics such as CD in capturing nuanced human preferences.

D. Details of User Study

We conducted a user study to quantitatively evaluate our point cloud-conditioned results with baselines. The study contained 100 questions, with each question displaying two rendered views of the input point cloud and the output meshes generated by the compared methods. To ensure an objective evaluation, all options were anonymously presented in a randomized order. Each participant was randomly assigned 30 questions and asked to select the best result for each. In total, we collected 900 valid responses from 30 volunteers with diverse backgrounds.

E. Limitations and Future Work

Although DeepMesh demonstrates impressive mesh generation capabilities, there are several limitations to address in future work. First, The generation quality of DeepMesh is constrained by the low-level features of point cloud conditioning. As a result, it struggles to recover fine-grained details present in the original meshes. To address this, future improvements could focus on enhancing the point cloud encoder or integrating salient point sampling techniques, such as those proposed in [1]. Also, DeepMesh is trained on a limited number of 3D data. We believe incorporating more datasets could further enrich the generated results. Additionally, we use only a 1B model due to limited computational resources. We believe that using a larger scale model would further improve the generation quality.

F. More Results

We provide more visualization results respectively in Figure 6 and Figure 7. Additionally, we select specific cases and present their high-resolution renderings in Figure 8,9 and 10 to see their finer details.

References

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

[2] Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. Meshanything v2: Artist-created mesh generation with adjacent mesh tokenization. *arXiv preprint arXiv:2408.02555*, 2024. 3

[3] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023. 2

[4] Zekun Hao, David W Romero, Tsung-Yi Lin, and Ming-Yu Liu. Meshtron: High-fidelity, artist-like 3d mesh generation at scale. *arXiv preprint arXiv:2412.09548*, 2024. 2, 3

[5] Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. *arXiv preprint arXiv:2110.13711*, 2021. 2

[6] Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, et al. Scaling mesh generation via compressive tokenization. *arXiv preprint arXiv:2411.07025*, 2024. 1, 3

Figure 6. **More results of DeepMesh.** We present more high-fidelity results generated by our method.

Figure 7. **More results of DeepMesh.** We present more high-fidelity results generated by our method.

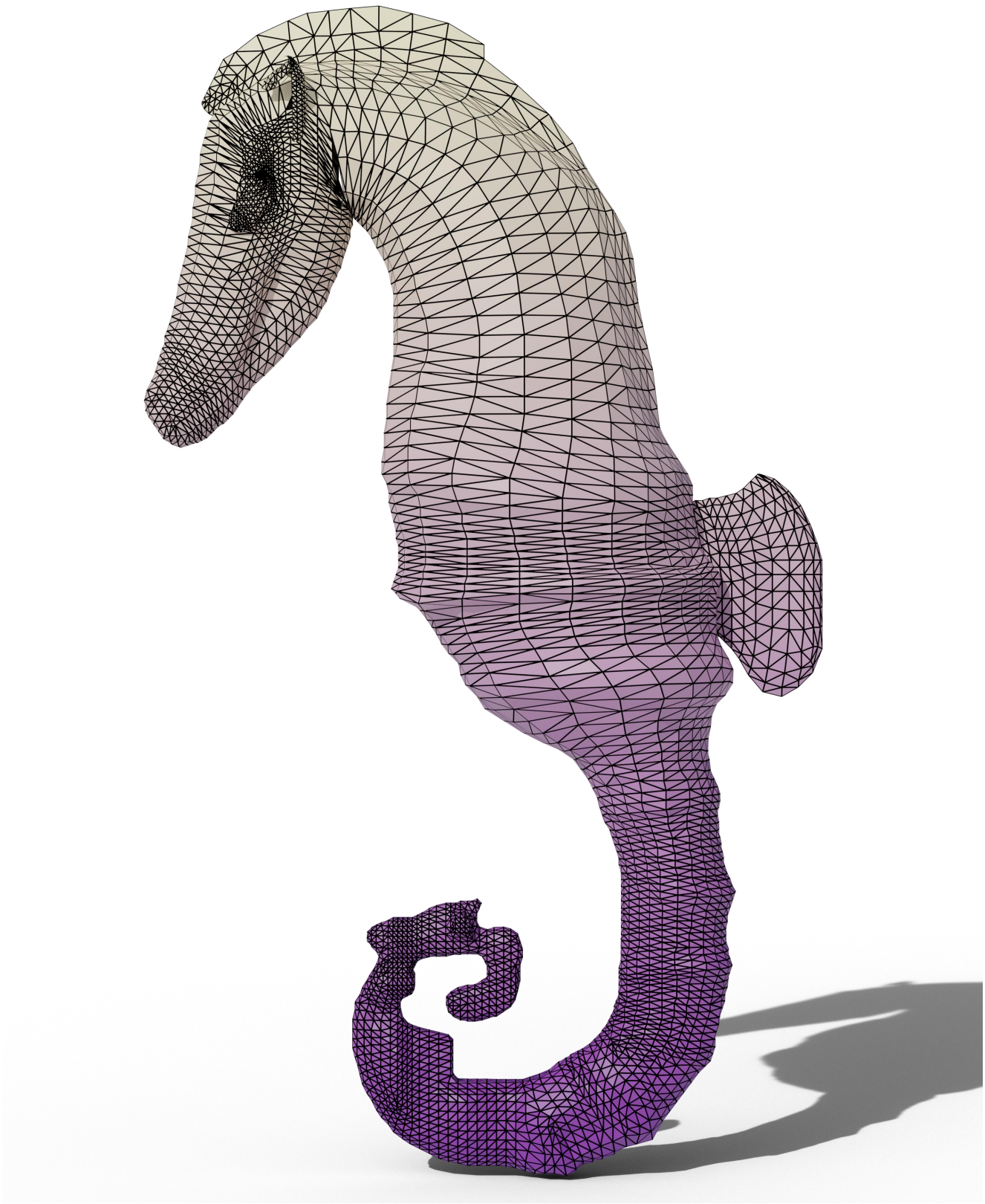


Figure 8. **High resolution results of our generated meshes.**

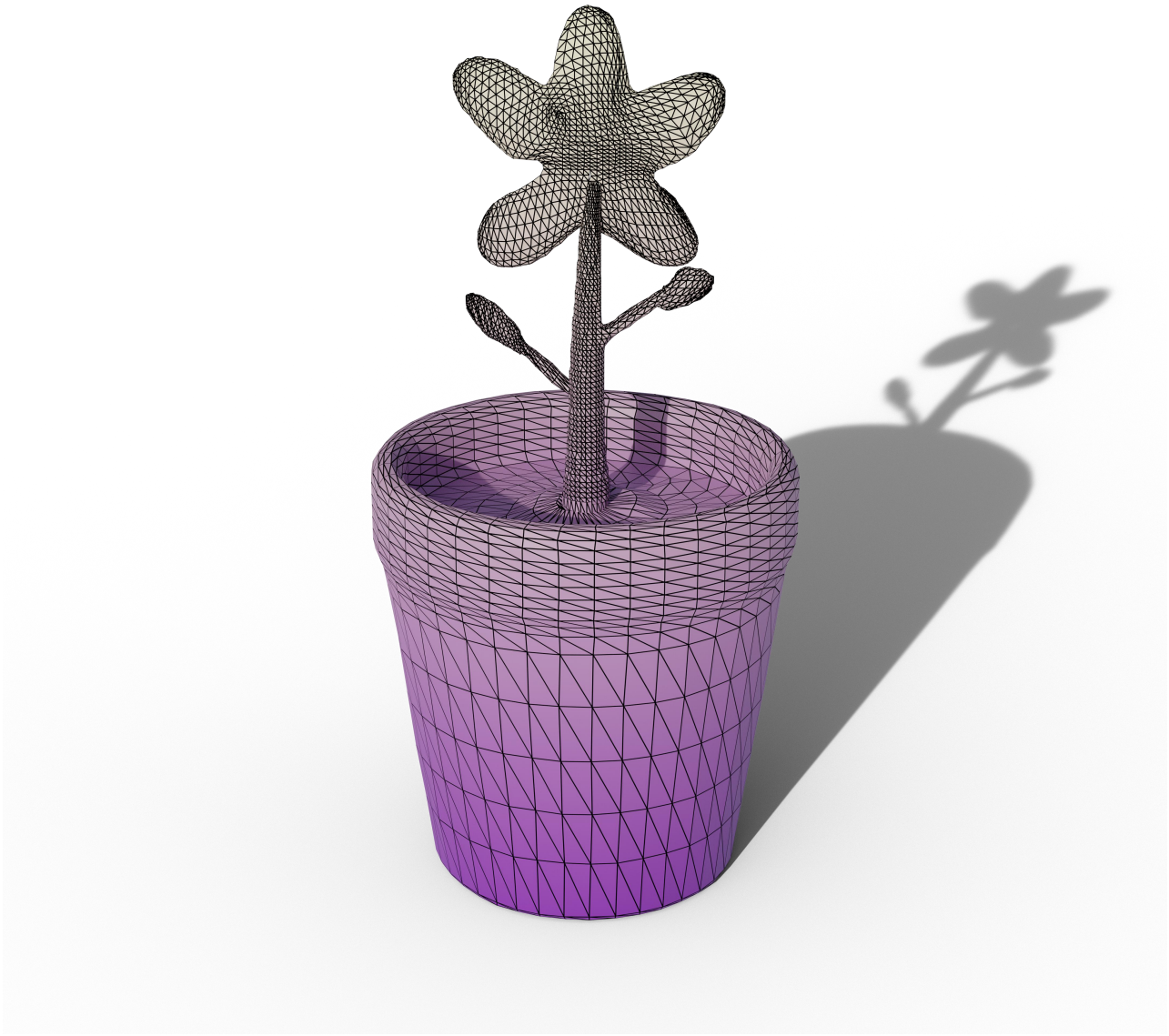


Figure 9. **High resolution results of our generated meshes.**

Figure 10. **High resolution results of our generated meshes.**