MeshFlow: Efficient Artistic Mesh Generation via MeshVAE and Flow-based Diffusion Transformer

Supplementary Material

A. Additional Implementation Details

In this section, we describe the implementation in more detail, including the data preparation and training specifics.

We utilize a proprietary dataset of approximately $600k$ high-quality 3D models from professional artists. All data are normalized to the $[-1, 1]$ unit space and pre-processed by merging duplicate and outlier vertices *without* any quantization. We excluded meshes with a maximum vertex degree exceeding 50 from the dataset for all experiments. Notably, the most complex model in our unfiltered dataset contains up to $15k$ faces. We uniformly sampled $32,768$ points for shape conditioning, which were then fed into a similarly pre-trained shape encoder with a structure similar to Shape2VecSet [7] to obtain 2048 tokens. The resulting tokens and the vertex count serve as conditional inputs for our Diffusion Transformer.

The mesh vertices are first sorted in z-y-x lexicographic order and then padded using points uniformly sampled from the mesh to the maximum vertex count before being used as input to our MeshVAE. The MeshVAE is built upon an 8-layer encoder and an 8-layer decoder Transformer architecture, with a hidden size of 1024, resulting in a model with a total of 233 million parameters. We use 16 Fourier encoding frequency bands for all features. The hyperparameters are set as follows: the threshold is $\tau = 0.6$, the negative sampling weight is $\lambda_{\text{neg}} = 0.01$, and the KL divergence weight is $\lambda_{\text{kl}} = 10^{-3}$ by default. For training, we employed the AdamW optimizer with a learning rate of 5×10^{-4}, complemented by a warmup strategy and a cosine annealing schedule. The VAE was trained with a batch size of 64 per GPU for 3 days using 32 H100 GPUs. The DiT uses the same standard transformer with 18 blocks and a 1024 hidden size, amounting to 427M parameters. Rotary Positional Embeddings (RoPE) [5] are utilized in every Self-Attention module. The RoPE can be applied to either the center coordinates or the indices of the encoded point set. We observe that using the center coordinates of the point set yields better performance during training. However, at inference time, the point center coordinates are unavailable, leading to a gap between the training and inference performance. The results presented in the paper therefore employ indices for RoPE. For training, we used a learning rate of 1×10^{-4} and trained the model for 10 days on 64 H100 GPUs with a batch size of 32 per GPU.

We also leverage Flash Attention [1] and BF16 mixed precision to accelerate training, complemented by Exponential Moving Average (EMA) to improve stability and generalization.

Algorithm 1 Robust Mesh Extraction

Input: Mask logits $\boldsymbol{M} \in \mathbb{R}^N$, vertex coordinates $\in \mathbb{R}^{N\times 3}$, vertex normals $\boldsymbol{N} \in \mathbb{R}^{N\times 3}$, edge embeddings $\boldsymbol{H} \in \mathbb{R}^{N\times D}$ and edge distance threshold τ
Output: Mesh $= (V, F)$
// Exact Edges and Adjacency Matrix
$\mathbf{I} \leftarrow \boldsymbol{M} > 0.5$
$\leftarrow [\mathbf{I}]$, $\boldsymbol{N} \leftarrow \text{CLAMP}(\boldsymbol{N}[\mathbf{I}], -1, 1)$
$D \leftarrow$ Compute Pairwise Distances$(\boldsymbol{H}[\mathbf{I}])$
$E \leftarrow \text{NonZero}((D > \tau) \wedge \text{UpperTriMask}(D))$
$\mathcal{A} \leftarrow$ Build Adjacency Matrix(E)
// Start Build All Faces from the Edges Soup
$F_{\text{temp}} \leftarrow \emptyset$
for $(v_1, v_2) \in \text{Unique}(E)$ **do**
    $\mathcal{N}_c \leftarrow (\mathcal{A}[v_1] \cap \mathcal{A}[v_2]) \setminus \{v_1, v_2\}$
    $F_{\text{temp}} \leftarrow F_{\text{temp}} \cup \{\text{Sorted}(v_1, v_2, v_3) \mid v_3 \in \mathcal{N}_c\}$
$F \leftarrow \text{Unique}(F_{\text{temp}})$
// Fix k-gon Ring in Topology
$E_{\text{boundary}} \leftarrow$ Detect Boundary Edges(E)
$\mathcal{C}_{\text{loops}} \leftarrow$ Extract K-Loops(DFS(E_{boundary}))
$F \leftarrow$ Triangulate$(\mathcal{C}_{\text{loops}})$
// Recover Cyclic Ordering
$N_{\text{face}} \leftarrow \text{Normalize}(\text{Cross}(_{f,1} -_{f,0}, _{f,2} -_{f,0}))$
$N_{\text{avg}} \leftarrow \text{Normalize}(\text{Mean}(_F, \text{dim} = 1))$
$F[N_{\text{face}} \cdot N_{\text{avg}} < 0, 1] \leftrightarrow F[N_{\text{face}} \cdot N_{\text{avg}} < 0, 2]$
Return Mesh(V, F)

B. Details of the Mesh Extraction Algorithm

As shown in Algorithm 1, we propose a simple and highly efficient mesh extraction algorithm to recover the local cyclic ordering of the vertices for each face and extract the final mesh from the decoded feature sequence $\hat{\mathbf{X}}$. In sharp contrast to the method proposed by SpaceMesh [4], which uses continuous permutation features to determine the local cyclic ordering of incident edges and infer the *half-edge* connectivity, we leverage the additionally predicted vertex normal information to recover the edge ordering. Specifically, we first obtain a set of valid edges based on the edge embeddings. Then, if an edge's two endpoints share a third vertex, the resulting triplet is considered to form a triangular face. The cyclic ordering of the vertices within this face is determined by calculating the mean of the three vertex normals to serve as the face normal. Furthermore, to repair de-

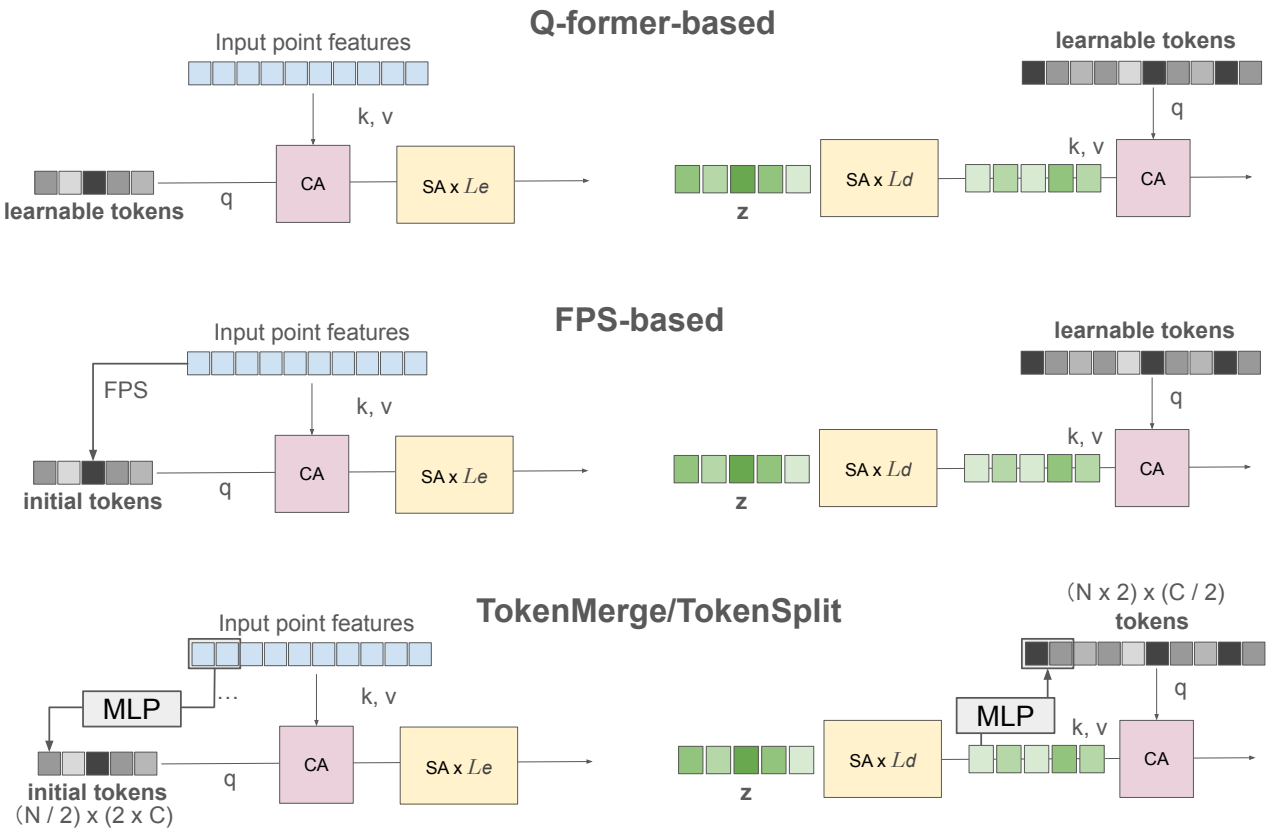

Figure 1. Detailed structure of different downsample and upsample strategies.

fects caused by inaccurate diffusion predictions, we implement a boundary repair post-processing step. A boundary edge is defined as an edge that belongs to only one triangular face. We therefore detect boundary edges and then determine whether these boundary edges form a k-gon ring. By default, when $k < 5$, we perform triangulation to convert the loop into multiple triangular faces, thereby enhancing the robustness of the generated results.

C. Different Downsample/Upsample Strategies

We provide a detailed description of our Token Merge and Token Split operators for downsampling and upsampling, which are analogous to PixelShuffle. We also detail the specific variations examined in the ablation studies.

- A **Q-Former-based approach** that utilizes randomly initialized learnable tokens as queries for both downsampling and upsampling.
- A **FPS-based approach** that employs Farthest Point Sampling (FPS) on encoded vertex features during downsampling like Shape2Vecset [7], and uses learnable tokens for upsampling.
- Our **TokenMerge/Split approach** that groups the encoded vertex features for downsampling and splits the latents by channel for upsampling like PixelShuffle.

D. Inference Time of A Single Object

As noted in Table 2 in the paper, to ensure consistency with FastMesh [2], we initially adopted the same calculation method for "Inf. Time", which uses the average inference time of a batch of multiple objects. However, this metric may not accurately reflect the true speed of single-object inference. For a fair comparison, we present the time required for different methods to infer a single mesh. The results in Tab. 1 demonstrate the superior inference efficiency of our proposed diffusion-based method compared to autoregressive methods.

E. Latent Space Learned by MeshVAE

We compress the discrete mesh data structure into a continuous and compact latent space. Here, we illustrate the distribution of the learned latent space of our MeshVAE, along with the latent distribution of a randomly sampled channel. As shown in Fig. 3, the latent distribution closely approximates a standard Gaussian distribution.

F. Additional Results

We present additional mesh reconstruction results of the MeshVAE in Fig. 2. The results collectively demonstrate

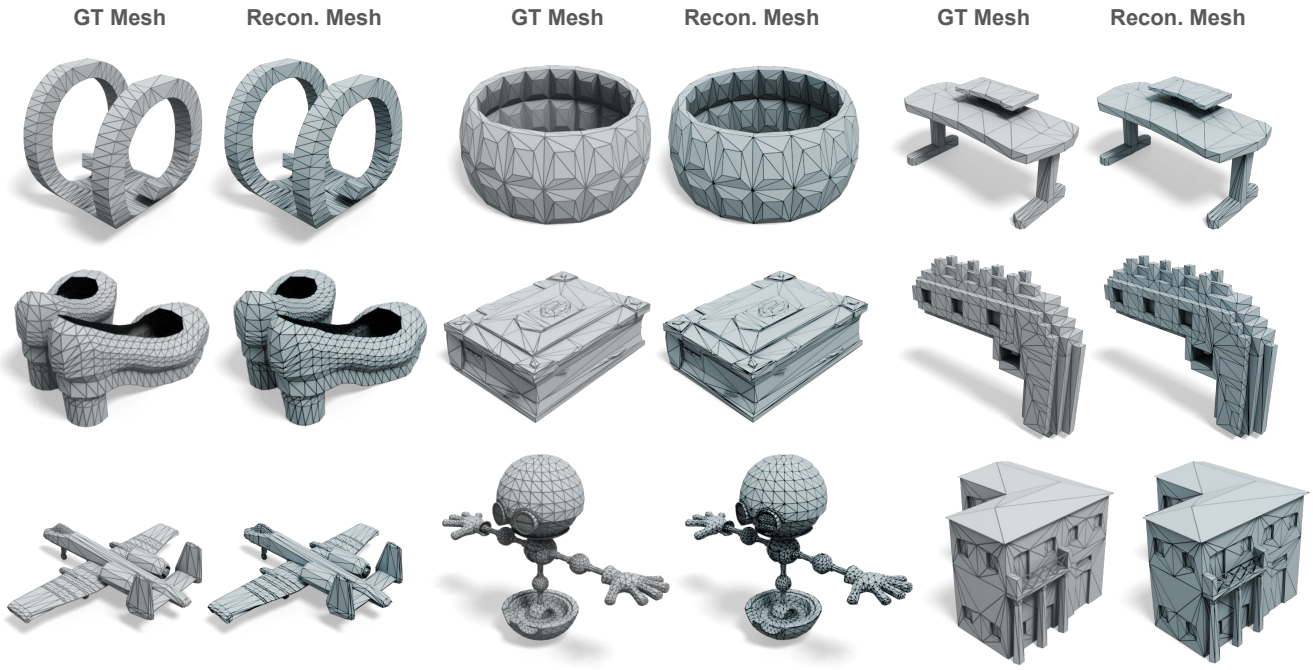

Figure 2. More mesh reconstruction results of our MeshVAE.

Table 1. Inference time of a single mesh generation. For autoregressive-based methods, the time varies with the complexity of the input.

Method	**BPT [6]**	**TreeMeshGPT [3]**	**DeepMesh [8]**	**FastMesh-V1K [2]**	**FastMesh-V4K [2]**	**Ours**
Inference Time	∼8 min	∼4 min	∼50 min	∼21 s	∼50 s	∼1.2 s

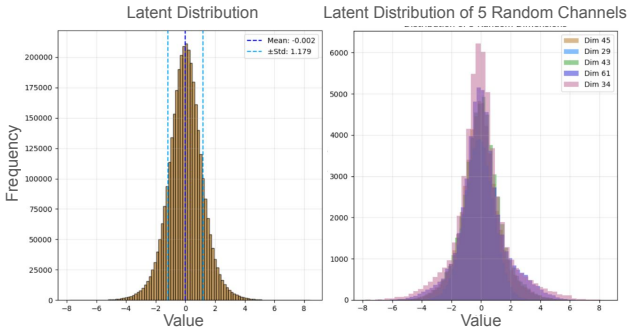

Figure 3. Latent distribution of the learned latent space.

that our method is capable of high-fidelity reconstruction of the input mesh from the learned latent space, preserving critical geometric details. Furthermore, we show more results of point cloud conditioned mesh generation i Fig. 5. These results highlight the ability of our method to generate a diverse set of meshes that are faithful to the input point cloud guidance.

G. Failure Cases

Our method is also subject to some failure cases when the model has difficulty generating accurate latents during the denoising process, which in turn leads to noisy vertex positions and edge embeddings. This ultimately causes the extracted meshes to exhibit holes, as shown in Fig. 4. We believe that this could be further mitigated by carefully refining the diffusion process or adopting a more powerful DiT model for generation.

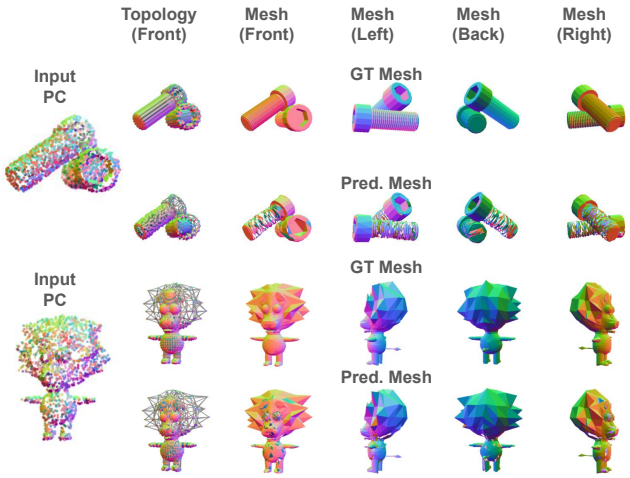

Figure 4. Failure cases of our generated meshes.

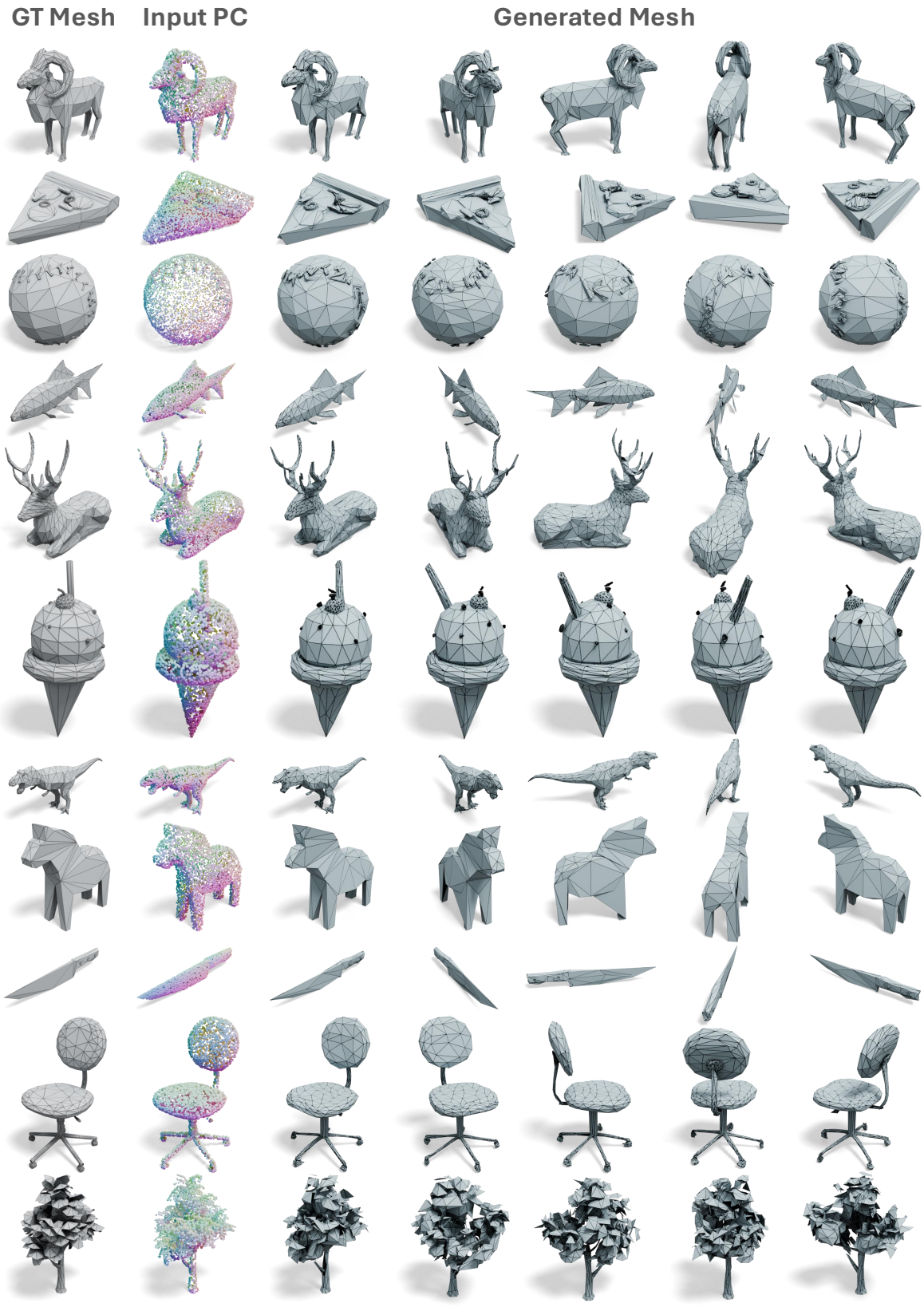

Figure 5. Additional point cloud conditioned mesh generation results for our method.

References

[1] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. 1

[2] Jeonghwan Kim, Yushi Lan, Armando Fortes, Yongwei Chen, and Xingang Pan. Fastmesh: Efficient artistic mesh generation via component decoupling. *arXiv preprint arXiv:2508.19188*, 2025. 2, 3

[3] Stefan Lionar, Jiabin Liang, and Gim Hee Lee. Treemeshgpt: Artistic mesh generation with autoregressive tree sequencing. *arXiv preprint arXiv:2503.11629*, 2025. 3

[4] Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. Spacemesh: A continuous representation for learning manifold surface meshes. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24)*, page 11, New York, NY, USA, 2024. ACM. 1

[5] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. 1

[6] Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, Tong Zhang, Shenghua Gao, and C. L. Philip Chen. Scaling mesh generation via compressive tokenization. *arXiv preprint arXiv:2411.07025*, 2024. 3

[7] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM TOG*, 42(4), 2023. 1, 2

[8] Ruowen Zhao, Junliang Ye, Zhengyi Wang, Guangce Liu, Yiwen Chen, Yikai Wang, and Jun Zhu. Deepmesh: Auto-regressive artist-mesh creation with reinforcement learning. *arXiv preprint arXiv:2503.15265*, 2025. 3