

TreeMeshGPT: Artistic Mesh Generation with Autoregressive Tree Sequencing

Stefan Lionar^{1,2,3}Jiabin Liang^{1,2,3}Gim Hee Lee³¹Sea AI Lab²Garena³National University of Singapore

Figure 1. **Artistic meshes generated by TreeMeshGPT.** Our method offers a novel sequencing approach for artistic mesh generation using autoregressive Transformer decoder by retrieving the next token from a dynamically growing tree structure. In our experiment with 7-bit discretization, TreeMeshGPT supports meshes with up to 5,500 triangular faces under strong point cloud conditioning.¹²

Abstract

We introduce *TreeMeshGPT*, an autoregressive Transformer designed to generate high-quality artistic meshes aligned with input point clouds. Instead of the conventional next-token prediction in autoregressive Transformer, we propose a novel *Autoregressive Tree Sequencing* where the next input token is retrieved from a dynamically growing tree structure that is built upon the triangle adjacency of faces within the mesh. Our sequencing enables the mesh to extend locally from the last generated triangular face at each step, and therefore reduces training difficulty and improves mesh quality. Our approach represents each triangular face with two tokens, achieving a compression rate of approximately 22% compared to the naive face tokenization. This efficient tokenization enables our model to generate highly detailed artistic meshes with strong point cloud conditioning, surpassing previous methods in both capacity and fi-

delity. Furthermore, our method generates mesh with strong normal orientation constraints, minimizing flipped normals commonly encountered in previous methods. Our experiments show that *TreeMeshGPT* enhances the mesh generation quality with refined details and normal orientation consistency.

1. Introduction

In recent advancements in 3D generation, representations such as voxels, point clouds, and implicit functions are often utilized [17, 35]. After the generation process, these representations are converted into meshes using techniques like Marching Cubes [19], which result in dense, over-tessellated triangular meshes. However, these dense meshes are unsuitable for applications that require real-time rendering, such as gaming and virtual reality. Although many mesh down-sampling algorithms can reduce the number of triangles, they also degrade mesh quality and produce messy, unstructured wireframes. In contrast, skilled artists can create highly compact meshes with minimal triangles

¹TreeMeshGPT supports a higher face count with finer discretization as it preserves the required manifold connectivity condition.

²Our code is available at: <https://github.com/sail-sg/TreeMeshGPT>.

while preserving the object’s sharp details. Additionally, the wireframes created by artists are more regular, aesthetically pleasing, and better aligned with the object’s feature or semantic boundaries, which facilitates further human interactions, such as editing and animation. However, the process of manually creating artist-quality meshes is highly time-consuming and labor-intensive.

This laborous process highlights the need for automated methods that can replicate the quality of artist-created meshes without requiring extensive manual effort. MeshAnything [3] seeks to bridge the gap between advancements in 3D generation and artist-quality mesh creation by adding point cloud condition to the artistic mesh generation Transformer initially proposed by MeshGPT [28]. Point clouds are chosen as the condition because they are either the direct output or can be obtained conveniently from the generated Marching Cubes meshes of the advanced 3D generation techniques.

MeshAnything [3] represents each triangular face with 9 latent tokens, leading to long sequences and limiting artistic mesh generation to 800 faces due to the Transformer’s quadratic complexity. This constraint poses challenges for real-world applications, which often require meshes with significantly higher face counts to accurately represent complex objects and environments. In the subsequent works, MeshAnything V2 [4] and EdgeRunner [29] leverage triangle adjacency to create shorter sequences to represent the same meshes. Consequently, they are able to generate meshes with up to 1,600 and 4,000 faces, respectively. However, many real-world applications demand meshes with higher face count to accurately represent detailed surface topology. Additionally, challenges remain in generating high-quality meshes free from artifacts such as gaps, missing components, and flipped normals.

To further improve tokenization efficiency and mesh quality, we introduce TreeMeshGPT. Unlike previous methods that rely on conventional next-token prediction in autoregressive Transformers, TreeMeshGPT introduces a novel Autoregressive Tree Sequencing approach. Instead of sequentially predicting next tokens, our method retrieves the next token from a dynamically growing tree structure built upon triangle adjacency within the mesh. This strategy allows the mesh to expand locally from the last generated triangular face at each step, and thus reducing training difficulty and enhancing mesh quality. Our approach represents each triangular face with two tokens, achieving a compression rate of 22% compared to the naive face tokenization of 9 tokens per face. This efficient tokenization technique pushes the boundary of artistic mesh generation. With 7-bit discretization, it enables the generation of meshes with up to 5,500 triangles under a strong point cloud condition of 2,048 tokens. Furthermore, our method generates meshes with strong normal orientation constraints,

minimizing flipped normals commonly encountered in MeshAnything [3] and MeshAnythingV2 [4].

In summary, our contributions are as follows:

- We propose a novel Autoregressive Tree Sequencing technique that efficiently represents two tokens per triangular face.
- Our proposed tokenization enables the training of a 7-bit discretization artistic mesh generative model with strong point cloud condition, capable of generating high-quality meshes with up to 5,500 faces.
- Extensive experiments show that our model can generate higher quality meshes and can generalize to real-world 3D scans.

2. Related Work

2.1. Mesh Extraction

Constructing a mesh from other 3D representations has been a research focus for decades. Among many successful methods [1, 15, 19], Marching Cubes [19] is the most widely used. It divides a scalar field into cubes and extracts triangles to approximate the isosurface. It is simple yet robust, producing watertight and 2-manifold results. Many improvements such as Dual Contouring [12] and Dual Marching Cubes [24] have been developed to enhance its capabilities. Another well-known method is Poisson reconstruction [13], which uses point clouds and normals as boundary conditions to solve a scalar field defined in 3D space, then applies Marching Cubes to extract the mesh. However, these approaches often focus on representing shapes with dense, over-tessellated meshes, resulting in messy, unstructured wireframes. This makes them unsuitable for downstream workflows that require efficient and structured meshes, such as real-time rendering, animation rigging, and editing. The dense, over-tessellated meshes not only increase computational load but also lack the regularity and semantic alignment necessary for the downstream processes.

2.2. 3D Generation

After the great success of 2D image generation [26], 3D generation has become a promising research direction. This field focuses on generating 3D assets for industries such as gaming, film, and AR/VR. Due to the limited availability of 3D data, early methods [11, 21, 25, 27] relied on optimizing underlying representations to mimic conditioning from 2D images or multi-view 2D images.

With the introduction of large-scale datasets [6, 7], feed-forward 3D generation techniques, such as those in [10, 18, 36], have become feasible. These techniques significantly improve generation speed compared to optimization-based methods. However, the resulting meshes often suffer from lower quality and lack diversity.

Inspired by the success of 2D diffusion models in image generation, many researchers have attempted to apply diffusion techniques directly to 3D data [5, 23, 32, 38]. For instance, CLAY [38], a transformer-based 3D latent diffusion model, achieves state-of-the-art results in high-quality shape generation. Despite these advances, these generation methods often require post-conversion for downstream applications, which remains a non-trivial challenge.

2.3. Autoregressive Mesh Generation

To address these limitations, recent approaches have leveraged autoregressive models for direct mesh generation [2, 22, 28, 33]. MeshGPT [28] was the first to tokenize a mesh through face sorting and compress it using a VQ-VAE [16, 30], followed by an autoregressive transformer to predict the compressed token sequence. This method enables the generation of meshes with direct supervision from artist-created topology information, which is often absent in previous approaches.

Subsequent works [2, 3, 33] explored more efficient representations and incorporated input conditioning, such as point clouds and images. However, these methods are limited to generating meshes with fewer than 800 faces due to the long sequence lengths and the quadratic computational cost of transformers. MeshAnythingV2 [4] and EdgeRunner [29] introduced more compact mesh tokenization techniques that leverage triangle adjacency, increasing the maximum face count to 1,600 and 4,000, respectively. Meshtron [9] proposes hourglass architecture and sliding window inference to scale up face count capacity while using the naive tokenization [2]. In a concurrent work, BPT [34] proposes a compact tokenization using block-wise indexing and localized patch aggregation. These methods rely on the next-token prediction commonly used in large language models (LLMs) and autoregressive image generation. In contrast, we offer a novel sequencing strategy based on a dynamically growing tree structure, aiming to increase the maximum face count and improve the overall quality of the generated meshes.

3. Method

3.1. Autoregressive Tree Sequencing

Tokenization plays a crucial role in autoregressive models, particularly in complex tasks like 3D mesh generation, where both the quality and efficiency of tokenization significantly affect model performance and scalability. In the context of mesh generation, tokenization involves encoding vertices, edges, or faces into sequential tokens that the model can process step-by-step. Drawing insights from LLMs, previous autoregressive mesh generation methods follow next-token prediction strategy that explicitly uses each output as the input for the subsequent step.

Our approach differs from those methods by using a tree-based traversal scheme to grow the mesh during the generation process. Specifically, in TreeMeshGPT, the input to the Transformer decoder are directed mesh edges, represented as $\mathbf{I} = \{(v_1^n, v_2^n)\}_{n=1}^N \in \mathbb{R}^{N \times 6}$, where each (v_1^n, v_2^n) denotes a pair of dequantized vertices for the generation at step n . At each step, the Transformer decoder makes a localized prediction to either add a new vertex v_3^n to expand the mesh by connecting to the initial pair of vertices (v_1^n, v_2^n) or predict [STOP] label indicating that no further expansion should occur from the input edge. TreeMeshGPT leverages a tree traversal process to construct the sequential input-output pairs for the autoregressive generation, described as follows (Note: the sequence order description below omits auxiliary tokens for simplicity).

Sequence order: We utilize a half-edge data structure with depth-first-search (DFS) traversal to construct the sequential input-output pairs, $(\mathbf{I}, \mathbf{O}) = \{(I_n, o_n)\}_{n=1}^N$, accompanied by a dynamic stack \mathbf{S} to manage the traversal process.

The traversal process starts from a directed edge in a mesh, in which we determine if this edge has an opposite vertex that forms a triangle, is a boundary, or if the triangle has already been visited. When a new triangle is formed, the output o_n is defined as the opposite vertex v_3^n . Two new edges are created by connecting v_3^n to the initial edge’s vertices $I_n = (v_1^n, v_2^n)$, resulting in edges (v_3^n, v_2^n) and (v_1^n, v_3^n) . These edges are directed in a counter-clockwise orientation on the potential next adjacent faces, as enforced by the half-edge data structure. The newly created edges are then pushed onto the stack \mathbf{S} for continued traversal:

$$\mathbf{S} := \mathbf{S} \odot (v_1^n, v_3^n) \quad \text{and} \quad \mathbf{S} := \mathbf{S} \odot (v_3^n, v_2^n)$$

where \odot represents the operation of pushing the edge (v_i, v_j) onto the top of stack \mathbf{S} . Conversely, if the input edge is a boundary or if adding a new vertex forms a previously visited triangle, the output o_n is set to the [STOP] label. In this case, no new vertex or edge is added.

The input for the next step, $n + 1$, is obtained by popping the top edge from the stack:

$$I_{n+1} = (v_1^{n+1}, v_2^{n+1}) := \text{top}(\mathbf{S}), \quad \mathbf{S} := \mathbf{S} \setminus \text{top}(\mathbf{S})$$

where $\text{top}(\mathbf{S})$ retrieves the current top edge of the stack \mathbf{S} , and $\mathbf{S} := \mathbf{S} \setminus \text{top}(\mathbf{S})$ updates \mathbf{S} by removing this top edge. We initialize the stack with a directed edge at the lowest position of the mesh and its twin. The traversal then proceeds until all triangles are visited. A simple illustration of this sequencing process is provided in Figure 2.

Note that a mesh may consist of multiple connected components. A component begins from an initial edge, expands as edges are added to a stack, and is considered fully traversed once the stack is empty. When there are multiple components in a mesh, the traversal of the first component starts from the edge with the lowest position in the mesh

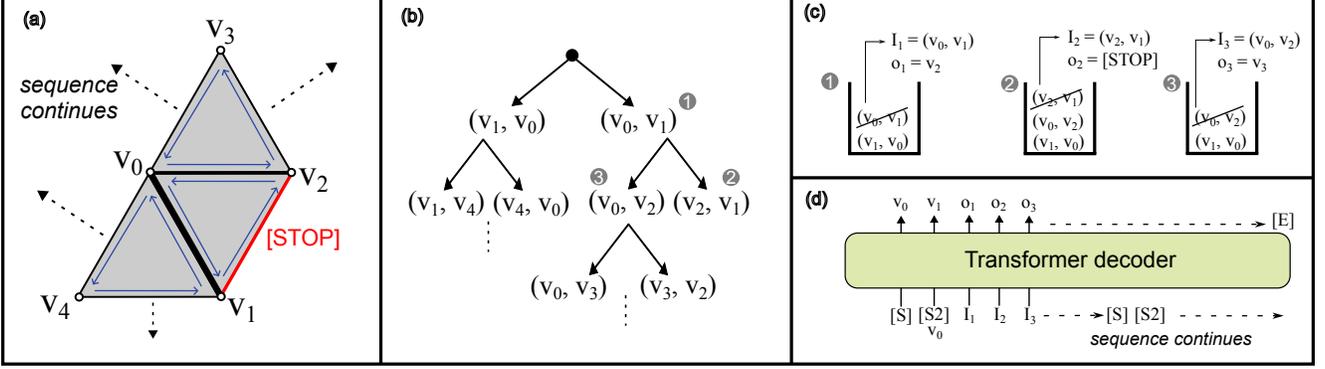


Figure 2. **Illustration of the sequence order in our Autoregressive Tree sequencing.** *a).* A small subset of a triangular mesh. [STOP] indicates boundary. *b).* An equivalent tree representation of the mesh. In this tree, each node is represented as a directed edge from a pair of vertices. The root is initialized with two child nodes: (v_0, v_1) and its twin (v_1, v_0) . A DFS traversal then proceeds to create the input-output sequence. *c).* Dynamic stack from the DFS traversal. The stack is initialized with (v_0, v_1) and its twin (v_1, v_0) . The input I_n is always obtained from the top of the stack. Thus, $I_1 = (v_0, v_1)$ at step 1. The opposite vertex of I_1 is v_2 and consequently, o_1 is set to v_2 . Two new edges are then formed by connecting the opposite vertex to the initial pair of vertices: (v_2, v_1) and (v_0, v_2) . The direction is enforced to be counter-clockwise on the potential next adjacent faces. At step 2, $I_2 = (v_2, v_1)$. Since I_2 is a boundary, o_2 is set to [STOP] label and no new edge is added to the stack. Step 3 and onwards follow the same traversal process. *d).* Transformer decoder sequence. The sequence in the Transformer decoder follows the input-output pairs from the tree traversal. The auxiliary tokens to initialize the generation of a connected component and the [EOS] are also added to the input-output sequence.

and continues until no edge remains in the stack. Each subsequent component begins from the next available edge positioned at the lowest among the remaining unvisited faces.

Generation process: To initialize the generation of a mesh component, an auxiliary token $[SOS] \in \mathbb{R}^D$ is used as the input to predict the first vertex v_1 . For the next step, a second auxiliary token $[SOS2] \in \mathbb{R}^C$, concatenated with the embedded representation of v_1 , serves as input to predict the second vertex v_2 . Once these initial vertices are predicted, the mesh generation proceeds through our Autoregressive Tree Sequencing with a stack initialized by (v_1, v_2) and its twin (v_2, v_1) . When the stack is empty—indicating the current component is complete—a new component is initialized by reintroducing the [SOS] and [SOS2] tokens. After all components have been generated, the sequence is terminated with an [EOS] label.

The final mesh is constructed by gathering the faces formed from the initial input vertex pairs and their predicted opposite vertices: $\mathcal{M} = \bigcup_{n=1}^N (v_1^n, v_2^n, v_3^n)$ for the generation steps where $I_n \notin \{[SOS], [SOS2]\}$ and $o_n \notin \{[STOP], [EOS]\}$.

Input embedding: We employ a positional embedding function from [37] to encode each vertex into a high-dimensional space, capturing its positional information across multiple frequency bases. This embedding function $\text{PosEmbed}(\cdot) : \mathbb{R}^3 \rightarrow \mathbb{R}^C$ maps 3D coordinates to a C -dimensional embedding. For each edge, the embeddings of its vertex pair are concatenated, creating a representation in \mathbb{R}^{2C} , which is subsequently passed through an MLP to map it to the Transformer’s hidden dimension, $\mathbb{R}^{2C} \rightarrow \mathbb{R}^D$.

Vertex prediction: In prior works on mesh generation, each vertex is represented as a sequence of three tokens corresponding to its quantized x -, y -, and z -coordinates. Specifically, to predict a single vertex’s position, those models generate each coordinate independently as separate tokens in sequence. This approach leads to longer sequences, as each vertex requires three distinct tokens. In contrast, our method predicts the vertex’ quantized x -, y -, and z -coordinates in one sequence length by using hierarchical MLP heads. This hierarchical approach maintains the sequential nature in predicting the x -, y -, and z -coordinates. Further details can be found in the supplementary material. As shown in the ablation study (Section 4.5), our hierarchical MLP heads result in easier coordinate sampling compared to prediction heads that predict the x -, y -, and z -coordinates simultaneously.

Advantages: Our Autoregressive Tree Sequencing approach adds only two sequence steps per triangular face as each face introduces two new nodes during the tree traversal process. Additionally, since most meshes consist of only a few connected components, our method requires minimal auxiliary tokens. This efficient sequencing achieves a compression rate of approximately 22% for most meshes compared to naive tokenization, which uses 9 tokens per face. Our compression rate is thus approximately double that of methods like MeshAnythingV2 [4] and EdgeRunner [29]. Additionally, by using a dynamic stack to manage the input sequence, our method allows the Transformer to focus solely on making localized predictions at each step, hence improving training efficiency. Furthermore, our method

generates mesh with strong normal orientation constraints, minimizing flipped normals commonly encountered in MeshAnything [3] and MeshAnythingV2 [4].

Loss function: We aim to train the Transformer decoder θ to maximize the likelihood of generating the sequence of outputs $\{o_n\}_{n=1}^N$ given the input sequence $\{I_n\}_{n=1}^N$:

$$\prod_{n=1}^N P(o_n | I_{\leq n}; \theta). \quad (1)$$

To this end, given the ground truth and predicted values with teacher-forcing across all steps, denoted by $\mathbf{O} = \{\mathbf{O}_x, \mathbf{O}_y, \mathbf{O}_z\}$ and $\hat{\mathbf{O}} = \{\hat{\mathbf{O}}_x, \hat{\mathbf{O}}_y, \hat{\mathbf{O}}_z\}$, respectively, where each \mathbf{O} represents the discretized vertex coordinate along a specific axis, we use a loss function defined as the sum of cross-entropy losses for each coordinate:

$$\mathcal{L} = \mathcal{L}_{\text{CE}}(\mathbf{O}_x, \hat{\mathbf{O}}_x) + \mathcal{L}_{\text{CE}}(\mathbf{O}_y, \hat{\mathbf{O}}_y) + \mathcal{L}_{\text{CE}}(\mathbf{O}_z, \hat{\mathbf{O}}_z). \quad (2)$$

To incorporate stopping conditions, we add the [STOP] and [EOS] labels to the class selection on the height axis, extending it with two additional classes beyond the discretized coordinate classes.

4. Experiments

4.1. Dataset

We use Objaverse [7] meshes as our training dataset. To ensure high-quality meshes, we select meshes that meet the half-edge traversal requirement, i.e., they are manifold and have no flipped normal. All meshes are preprocessed by centering and normalizing them within a cube spanning $[-0.5, 0.5]$. We apply 7-bit discretization, remove any duplicate triangles, and choose meshes with less than 5,500 faces. Additionally, we perform orthographic projections and exclude meshes where one of the projections has extremely small area or contains more than one cluster. After filtering, we retain a total of 75,000 meshes, of which 1,000 are reserved for validation, with the remainder used for training. To increase data diversity, we apply the following augmentations:

- **Scaling:** Each axis is scaled independently by a factor randomly chosen from the range $[0.75, 0.95]$.
- **Rotation:** We first apply a 90° or -90° rotation along the x - or y -axis with a probability of 0.3. Afterward, we perform a rotation around the z -axis with an angle uniformly sampled from $[-180^\circ, 180^\circ]$.

4.2. Implementation Details

Point cloud conditioning: Following previous approaches [3, 4, 29], we adopt point cloud conditioning to provide practical guidance for the generation process. To achieve this, we sample a point cloud from the input mesh surface and apply a lightweight encoder [3, 4, 37, 38]. Specifically, we sample 8192 points, denoted as $\mathbf{X} \in$

$\mathbb{R}^{8192 \times 3}$, from the mesh surface. A cross-attention layer is then used to encode these points into a latent code:

$$\mathbf{Z} = \text{CrossAtt}(\mathbf{Q}, \text{PosEmbed}(\mathbf{X})) \quad (3)$$

where $\mathbf{Q} \in \mathbb{R}^{2048 \times C}$ represents query embeddings with a shorter sequence length of 2048 and hidden dimension of C , $\text{PosEmbed}(\cdot)$ is the same input embedding function in our Autoregressive Tree Sequencing, and $\mathbf{Z} \in \mathbb{R}^{2048 \times L}$ is the resulting latent code. The latent code \mathbf{Z} is then prepended to the initial [SOS] token in the Transformer decoder to provide point cloud-based conditioning for mesh generation.

Architecture details: Our model employs a Transformer decoder with 24 layers, 16 attention heads, and a hidden dimension of 1024 and adds the sinusoidal positional encoding [31] to encode the token position. We apply a full self-attention for the latent code condition and a causal self-attention mask for the autoregressive decoder. PyTorch’s FlexAttention is used to implement this attention mask efficiently. Additionally, we adopt fp16 mixed-precision to optimize computational speed and memory efficiency. We set the hidden dimension C of the positional embedding (PosEmbed) to 512 and use 7-bit quantization to discretize the coordinate output into 128 classes.

Training details: We use AdamW [14, 20] with a learning rate of 10^{-4} , $\beta_1 = 0.9$ and $\beta_2 = 0.99$ as the optimizer. Our model is trained with $8 \times$ A100-80GB GPUs for 5 days with an effective batch size of 128.

Sampling strategy: We use a multinomial sampling strategy with a top- k of 5 during the generation process. Empirically, we find that adjusting the temperature at different stages achieves an optimal balance between diversity and generation quality. Specifically, we set the temperature to 1 when the stack length is below 10, reduce it to 0.4 when the stack length is below 30, and further decrease it to 0.2 beyond that.

Inference adjustment: We find that a few adjustments during inference can improve the generation performance. First, we keep track of the generated faces for each step. If our model predicts a vertex that would form a triangle that is a duplicate to the previously generated faces, we adjust the prediction to [STOP] and retrieve the next input from the top of the stack. This checking operation is fast since the faces are of discrete tensor.

Additionally, we observe that our model often struggles to predict the [EOS] label in longer sequences. In these cases, while the [EOS] label consistently appears among the top 5 logits, it is rarely sampled. To address this, we apply an addition factor to the logit of [EOS], incrementally increasing this factor each time [EOS] appears in the top 5 logits. To avoid early [EOS] prediction after this adjustment, we bypass multinomial top- k sampling for [EOS] and select it only when it becomes the highest logit.

4.3. Results on Objaverse Dataset

4.3.1. Qualitative Results

We present the qualitative results of mesh generation conditioned on input point cloud in Figure 3, comparing our method with MeshAnything [3] and MeshAnythingV2 [4] as the baselines. As shown, our method demonstrates a notable improvement to generate meshes with higher face counts and refined details.

4.3.2. Quantitative Results

We use the following metrics to assess the quality of the generated meshes:

- **Chamfer Distance (CD).** It provides an indication of geometric accuracy by measuring the average closest-point distance between points sampled from the source and reference meshes, computed bidirectionally. Lower values of CD indicate better geometric accuracy.
- **Normal Consistency (NC).** It evaluates the surface orientation alignment between the source and reference meshes. Specifically, it measures the cosine similarity between the normals of each face in the source mesh and the nearest face in the reference mesh, computed bidirectionally. Higher NC values signify better normal alignment. We also report absolute Normal Consistency ($|\text{NC}|$), which disregards the sign, focusing solely on the magnitude of similarity. The mathematical details are provided in the supplementary materials.

Objaverse evaluation set: We ran the evaluation on 200 samples in our validation set with 1 generation seed for each model. The quantitative results presented in Table 1, indicate that our model generates meshes more faithful to the ground truth meshes compared to the baselines. The NC values of MeshAnything [3] and MeshAnythingV2 [4] are relatively low due to flipped normals, which leads to inconsistencies in the sign of the normals. In contrast, our method generates meshes with consistent normal orientation.

Model	CD↓	NC↑	$ \text{NC} $ ↑
MeshAnything [3]	0.0115	0.223	0.853
MeshAnythingV2 [4]	0.0102	0.167	0.843
Ours	0.0070	0.798	0.880

Table 1. Quantitative results on Objaverse evaluation set.

Tokenization effectiveness: The previous trained models and ours differ in dataset composition, point cloud conditioning, and training settings. To accurately evaluate the effectiveness of different tokenization methods, we conduct a controlled experiment on the subset of our filtered dataset of 24k samples with ≤ 500 faces. All factors except for the tokenizers are kept the same: 1). 16-layer Transformers with 768 hidden dimension and positional encoding. 2). Point

cloud condition of 2048 tokens. 3). Training until 40k steps with an effective batch size of 128 and data augmentation.

Results on a test set of 200 samples, shown in Table 2, strongly indicate our method provides highly effective inductive bias. We use PivotMesh’s [33] pretrained VQ-VAE since MeshAnything does not release their VQ-VAE encoder and noise resistant decoder fine-tuning code.

Tokenizer	CD↓	NC↑	$ \text{NC} $ ↑	Sequence Length↓
Naive [2]	0.0376	0.639	0.822	$9N_f$
VQ-VAE [33]	0.0352	0.673	0.815	$6N_f$
AMT [4]	0.0327	-0.069	0.768	$\pm 4N_f$
Ours	0.0100	0.734	0.874	$2N_f + 2N_c$

Table 2. Quantitative results on our controlled experiment. N_f and N_c are the number of triangular faces and connected components, respectively.

4.4. Results on GSO Dataset

We further conduct a quantitative evaluation on GSO dataset [8], a dataset of real-world 3D scans to test the generalization capabilities of each model. In this experiment, we observe that the baseline models and ours are sensitive to the input point cloud, with results varying large on different sampling seeds. To mitigate this variability, we generate five samples for each model and select the mesh with the lowest Chamfer Distance for evaluation.

For our model, we sample point clouds directly from the meshes provided in the GSO dataset. However, direct sampling from these high-resolution meshes often results in reconstructions with extremely small faces. To address this, we decimate the meshes to five target faces ranging from 1000 to 2500 faces and then perform uniform sampling on these decimated versions. For MeshAnything and MeshAnythingV2, we sample the input point clouds from Marching Cubes meshes derived from both the original high-resolution and decimated (1500 faces) meshes using an 8-level octree.

As shown in Table 3, our method achieves the best results across the evaluated metrics. Figure 4 presents qualitative comparisons, demonstrating that our model generates meshes more consistent with the input than the baselines. Additionally, Figure 5 compares our output with the decimated mesh, highlighting that our model can generate meshes with the topology of those created by 3D artists.

Model	CD↓	NC↑	$ \text{NC} $ ↑
MeshAnything [3]	0.0105	0.453	0.869
MeshAnythingV2 [4]	0.0116	0.3269	0.865
Ours	0.0077	0.842	0.897

Table 3. Quantitative results on GSO dataset.

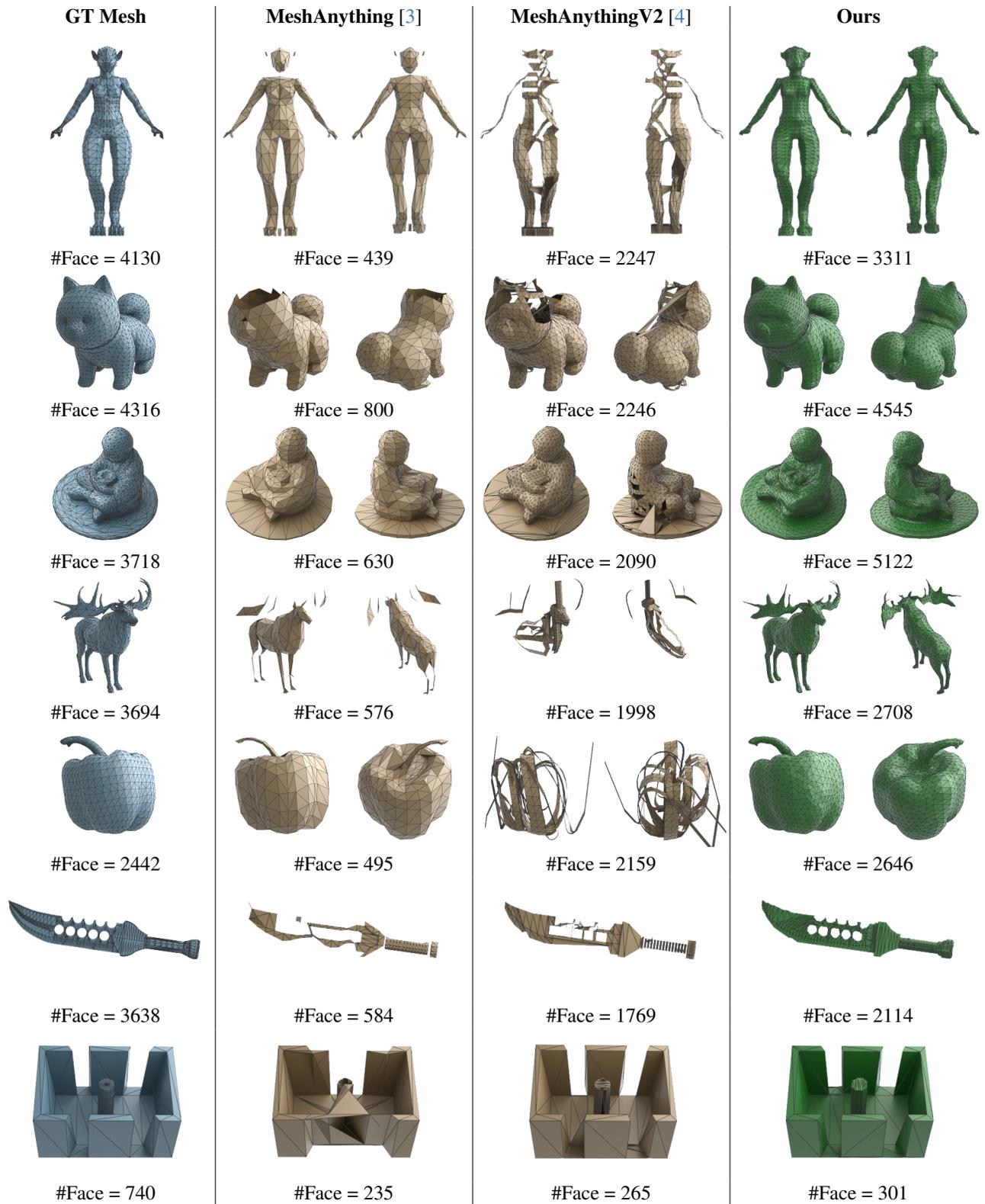


Figure 3. **Qualitative comparison on Objaverse dataset [7].** Our model is able to generate meshes with higher face counts and refined details compared to the baselines. Results from the baselines use point clouds sampled from marching cube meshes with 8-level octree.

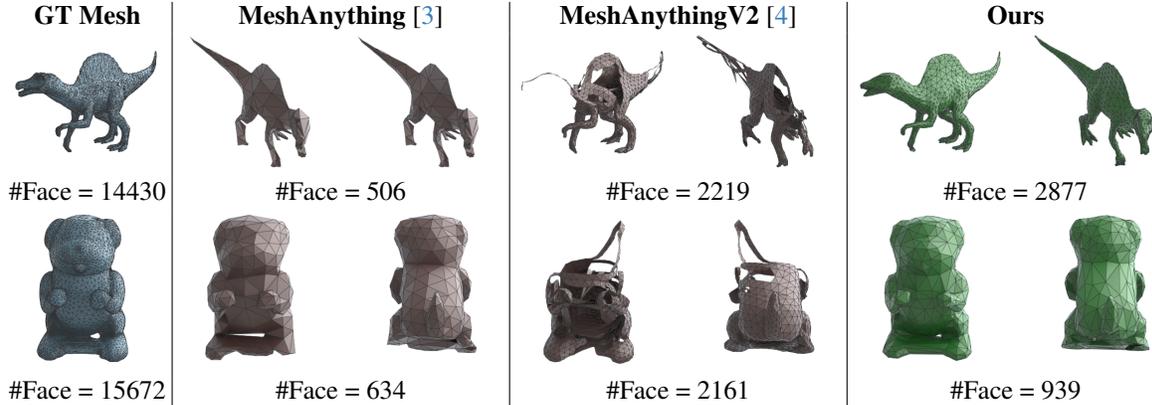


Figure 4. Qualitative comparison on GSO dataset [8].

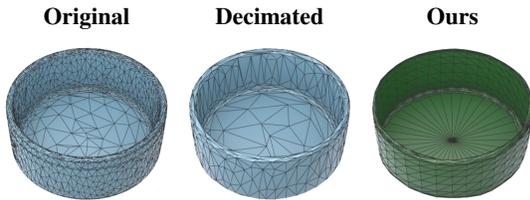


Figure 5. Comparison between the decimated mesh and our output. Our model is capable of generating meshes with the topology of those created by 3D artists.

4.5. Ablation Study

Decoder MLP head: We compare TreeMeshGPT trained with MLP heads that predict the x - y - z coordinates simultaneously and our proposed hierarchical MLP. Predicting the coordinate simultaneously leads to challenging sampling that results in noisy and more incomplete meshes, as shown in Figure 6. Running evaluation on the 200 validation samples used in Table 1 with this head yields $CD = 0.0114$, $NC = 0.724$, $|NC| = 0.847$.

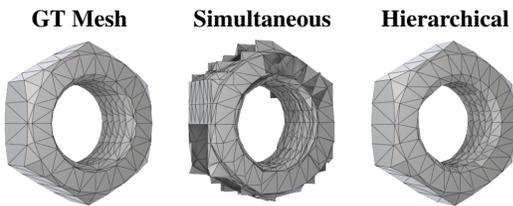


Figure 6. MLP head ablation. Our hierarchical MLP maintains the sequential nature of the x - y - z coordinates prediction that results in easier sampling compared to simultaneous prediction.

Tree traversal: We conduct an ablation study on smaller Transformer architecture comparing DFS and breadth-first-search (BFS) traversals in forming input-output sequences for our Autoregressive Tree Sequencing. As shown in the training perplexity plot in Figure 7, DFS traversal enables more efficient Transformer training. This improvement likely stems from the stronger local dependencies introduced by DFS, where each step is more predictable based on its immediate predecessors. In contrast, BFS traversal often

introduces dependencies between steps that are spatially or structurally distant, and thus complicating the learning process.

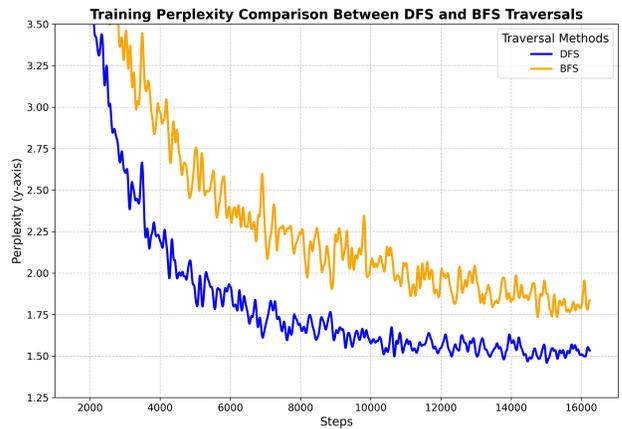


Figure 7. Training perplexity comparison Between BFS and DFS traversals. DFS traversal shows a better training perplexity compared to BFS. Shown in the plot is the perplexity for y -axis vertex coordinate.

5. Conclusion

We present TreeMeshGPT, an autoregressive Transformer designed to generate high-quality artistic meshes aligned with input point clouds. TreeMeshGPT incorporates a novel Autoregressive Tree Sequencing technique instead of the conventional next-token prediction. Our Autoregressive Tree Sequencing represents each face with two tokens, enabling a 7-bit discretization model that can generate up to 5,500 triangular faces with 2,048 point cloud latent tokens. Experiments show that TreeMeshGPT can generate meshes with higher quality compared to the previous methods.

Limitations: Our model has a similar failure mode to the previous methods that the success rate decreases as the sequence length increases. Additionally, while our model has an improved capacity to generate meshes with higher face counts, challenges persist in enforcing an optimal mesh topology.

Acknowledgment. This research is supported by the National Research Foundation (NRF) Singapore, under its NRF-Invigatiorship Programme (Award ID. NRF-NRFI09-0008).

References

- [1] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 1999. 2
- [2] Sijin Chen, Xin Chen, Anqi Pang, Xianfang Zeng, Wei Cheng, Yijun Fu, Fukun Yin, Yanru Wang, Zhibin Wang, Chi Zhang, et al. Meshxl: Neural coordinate field for generative 3d foundation models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 3, 6
- [3] Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, et al. Meshanything: Artist-created mesh generation with autoregressive transformers. *arXiv preprint arXiv:2406.10163*, 2024. 2, 3, 5, 6, 7, 8
- [4] 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. 2, 3, 4, 5, 6, 7, 8
- [5] Yen-Chi Cheng, Hsin-Ying Lee, Sergey Tulyakov, Alexander G Schwing, and Liang-Yan Gui. Sdfusion: Multimodal 3d shape completion, reconstruction, and generation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3
- [6] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2
- [7] 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 *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 5, 7
- [8] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2022. 6, 8
- [9] 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. 3
- [10] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. Lrm: Large reconstruction model for single image to 3d. In *International Conference on Learning Representations (ICLR)*, 2024. 2
- [11] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2
- [12] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th Annual Conference on Computer graphics and Interactive Techniques*, 2002. 2
- [13] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*, 2006. 2
- [14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 5
- [15] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 2000. 2
- [16] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3
- [17] Chenghao Li, Chaoning Zhang, Joseph Cho, Atish Wagh-wase, Lik-Hang Lee, Francois Rameau, Yang Yang, Sung-Ho Bae, and Choong Seon Hong. Generative ai meets 3d: A survey on text-to-3d in aigc era. *arXiv preprint arXiv:2305.06131*, 2023. 1
- [18] Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich, and Sai Bi. Instant3d: Fast text-to-3d with sparse-view generation and large reconstruction model. In *International Conference on Learning Representations (ICLR)*, 2024. 2
- [19] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 1987. 1, 2
- [20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. 5
- [21] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2
- [22] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning (ICML)*, 2020. 3
- [23] Alex Nichol, Heewoo Jun, Pratul Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022. 3
- [24] Gregory M Nielson. Dual marching cubes. In *IEEE Visualization*, 2004. 2
- [25] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *International Conference on Learning Representations (ICLR)*, 2023. 2
- [26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

- [27] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. Mvdream: Multi-view diffusion for 3d generation. In *International Conference on Learning Representations (ICLR)*, 2024. 2
- [28] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2, 3
- [29] Jiaxiang Tang, Zhaoshuo Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. Edgerunner: Auto-regressive auto-encoder for artistic mesh generation. *arXiv preprint arXiv:2409.18114*, 2024. 2, 3, 4, 5
- [30] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 3
- [31] 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 (NeurIPS)*, 2017. 5
- [32] Tengfei Wang, Bo Zhang, Ting Zhang, Shuyang Gu, Jianmin Bao, Tadas Baltrusaitis, Jingjing Shen, Dong Chen, Fang Wen, Qifeng Chen, et al. Rodin: A generative model for sculpting 3d digital avatars using diffusion. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3
- [33] Haohan Weng, Yikai Wang, Tong Zhang, CL Chen, and Jun Zhu. Pivotmesh: Generic 3d mesh generation via pivot vertices guidance. *arXiv preprint arXiv:2405.16890*, 2024. 3, 6
- [34] 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. 3
- [35] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, 2022. 1
- [36] Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. Instantmesh: Efficient 3d mesh generation from a single image with sparse-view large reconstruction models. *arXiv preprint arXiv:2404.07191*, 2024. 2
- [37] Biao Zhang, Jiapeng Tang, Matthias Niessner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics (TOG)*, 2023. 4, 5
- [38] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 2024. 3, 5

Supplementary Material for TreeMeshGPT: Artistic Mesh Generation with Autoregressive Tree Sequencing

In this supplementary document, we provide the implementation details of our network’s MLP heads in Section A. Then, we provide the mathematical details of the Normal Consistency metrics in Section B. We then demonstrate the capability of TreeMeshGPT to generate artistic meshes from text prompts through a multi-step process in Section C. Finally, we present our 9-bit model supporting the generation of artistic meshes with up to 11,000 faces in Section D.

A. Vertex Prediction Heads

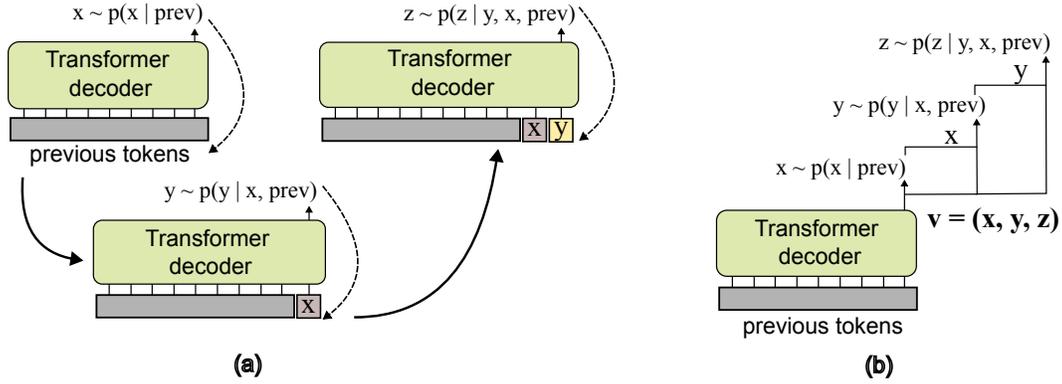


Figure 8. **Sequential vertex prediction.** a). Next-token prediction. b). Our hierarchical MLP heads.

To mimic the sequential nature in the prediction of vertex’s x -, y -, and z -coordinates in next-token prediction Transformer (Figure 1a), we adopt hierarchical MLP heads (Figure 1b). Our hierarchical MLP heads contain three stages to generate each vertex’s x -, y -, and z -coordinates, where each coordinate is predicted sequentially based on the previous ones. In the first stage of the hierarchical MLP, represented by g_{θ_1} , the initial coordinate (e.g., x -coordinate) of the vertex is predicted based on the latent code $\mathbf{c} \in \mathbb{R}^d$ from the Transformer decoder:

$$x \sim p(x | \text{prev}) = g_{\theta_1}(\mathbf{c}). \quad (4)$$

Here, "prev" denotes all previously generated tokens, and \mathbf{c} is the latent code output by the Transformer decoder, which encapsulates information from these prior tokens. Next, the y -coordinate is predicted in the second stage of the MLP, represented by g_{θ_2} :

$$y \sim p(y | x, \text{prev}) = g_{\theta_2}(E_x(x), \mathbf{c}), \quad (5)$$

where $E_* \in \mathbb{R}^d$ denotes the learnable embeddings for the discretized coordinates of an axis and $*$ can represent x , y , or z . For example, $E_x(x)$ represents the embedding of the discretized x -coordinate, and similarly, $E_y(y)$ and $E_z(z)$ denote the embeddings of the discretized y - and z -coordinates, respectively. This second stage conditions on both the latent code \mathbf{c} and the discretized coordinate embedding E_x . Finally, the z -coordinate is predicted in the third stage of the MLP, g_{θ_3} , which takes as input the latent code \mathbf{c} along with the embeddings of both previously predicted coordinates, $E_x(x)$ and $E_y(y)$:

$$z \sim p(z | y, x, \text{prev}) = g_{\theta_3}(E_y(y), E_x(x), \mathbf{c}). \quad (6)$$

In each stage, the input to the MLP g_{θ} consists of the concatenation of the latent code \mathbf{c} and the corresponding embeddings E_* .

In our experiments with the Objaverse dataset, where the z -axis represents the height axis, we predict the z -coordinate first, followed by the y -coordinate and then the x -coordinate. Additional [STOP] and [EOS] labels are included in the class selection for the z -coordinate. During training, the loss functions for the y - and x -coordinates are applied only when

the ground truth z -coordinate is not one of these additional labels. Also, teacher-forcing is employed to supervise the y - and x -coordinates by conditioning with the embeddings of the preceding ground truth coordinates.

B. Normal Consistency Metrics

This section details the calculation of our normal consistency metrics. Let \mathcal{M}_s and \mathcal{M}_r denote the source and reference meshes, respectively, where each consists of triangular faces. The centroid \mathbf{c}_i^s of the i -th face in the source mesh \mathcal{M}_s is given by:

$$\mathbf{c}_i^s = \frac{\mathbf{v}_{i1}^s + \mathbf{v}_{i2}^s + \mathbf{v}_{i3}^s}{3},$$

where $\mathbf{v}_{i1}^s, \mathbf{v}_{i2}^s, \mathbf{v}_{i3}^s$ are the vertices of the i -th triangular face of \mathcal{M}_s . For each centroid \mathbf{c}_i^s , we find the closest face j on the reference mesh \mathcal{M}_r using the shortest point-to-face distance:

$$j = \arg \min_{k \in \mathcal{M}_r} d(\mathbf{c}_i^s, F_k^r),$$

where F_k^r is the k -th face in \mathcal{M}_r and $d(\mathbf{c}_i^s, F_k^r)$ represents the shortest distance from the point \mathbf{c}_i^s to the face F_k^r . The cosine similarity between the normals of the i -th face in the source mesh (\mathbf{n}_i^s) and the closest face (\mathbf{n}_j^r) in the reference mesh is then computed as:

$$\text{Sim}_{i \rightarrow j}(\mathbf{n}^s, \mathbf{n}^r) = \frac{\mathbf{n}_i^s \cdot \mathbf{n}_j^r}{\|\mathbf{n}_i^s\| \|\mathbf{n}_j^r\|}.$$

This process is repeated bidirectionally. For the reverse direction, the centroid \mathbf{c}_k^r of the k -th face in \mathcal{M}_r is computed to find the corresponding closest face l in \mathcal{M}_s . The Normal Consistency (NC) metric is the average cosine similarity across all face pairs in both directions:

$$\text{NC} = \frac{1}{2|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} \text{Sim}_{i \rightarrow j}(\mathbf{n}^s, \mathbf{n}^r) + \frac{1}{2|\mathcal{M}_r|} \sum_{k \in \mathcal{M}_r} \text{Sim}_{k \rightarrow l}(\mathbf{n}^r, \mathbf{n}^s), \quad (7)$$

where $|\mathcal{M}_s|$ and $|\mathcal{M}_r|$ are the numbers of faces in the source and reference meshes, respectively and $l = \arg \min_{i \in \mathcal{M}_s} d(\mathbf{c}_k^r, F_i^s)$. The absolute version ($|\text{NC}|$) that omits the flipping direction is then given as:

$$|\text{NC}| = \frac{1}{2|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} |\text{Sim}_{i \rightarrow j}(\mathbf{n}^s, \mathbf{n}^r)| + \frac{1}{2|\mathcal{M}_r|} \sum_{k \in \mathcal{M}_r} |\text{Sim}_{k \rightarrow l}(\mathbf{n}^r, \mathbf{n}^s)|. \quad (8)$$

C. Generating Artistic Meshes from Text Prompts

We demonstrate the capability of our model to generate artistic meshes from text prompts through a multi-step process, shown in Figure 9. We utilize the Luma AI Genie³ text-to-3D model to generate dense meshes from text prompts. These meshes are typically over-tessellated, containing around 50,000 small triangles that make them unsuitable for downstream applications. To generate artistic meshes, we first apply decimation to the dense meshes. Next, we sample point clouds from the decimated meshes and use them as input conditions of TreeMeshGPT.

³<https://lumalabs.ai/genie>

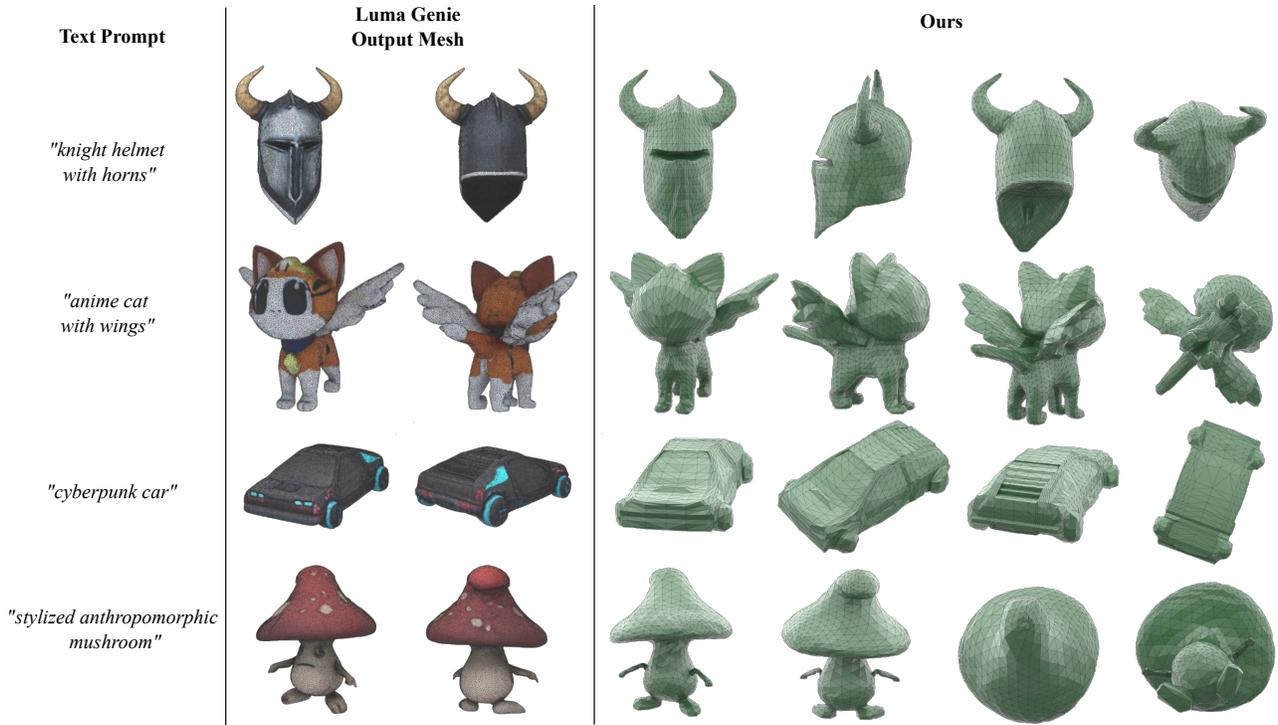


Figure 9. **Multi-step text-to-artistic mesh generation.** Given a text prompt, we first generate a dense mesh using the Luma AI Genie model. This dense mesh, typically containing around 50,000 triangles, is then decimated. A point cloud is sampled from the decimated mesh and serves as the input condition for TreeMeshGPT, which generates the final artistic mesh.

D. 9-bit Model Supporting 10K+ Faces

In our model training with 7-bit discretization, we performed the discretization to the normalized manifold Objaverse meshes, removed the duplicate triangles, and chose meshes with $\leq 5.5k$ faces as we found significant amount of these discretized meshes with $> 5.5k$ faces contain small triangles that collapse or merge, thus violating the manifold condition required for our sequencing approach.

These triangles collapses/merges occur less with finer discretization and we further train TreeMeshGPT with 9-bit discretization. Our 9-bit model supports the generation of up to 11,000 faces, taking 25 days of training with $8 \times$ A100-80GB GPUs. Some of the qualitative results are shown in Figure 10. Compared to the 7-bit model, our 9-bit model can generate artistic meshes with smoother surfaces, finer details, and higher number of faces.



Figure 10. **Qualitative results of our 9-bit model.** The generated meshes contain up to 11,000 faces, demonstrating improved surface smoothness and finer details compared to the 7-bit model. Inputs are point clouds sampled from Objaverse meshes.