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

Stefan Lionar[1,2,3] Jiabin Liang[1,2,3] Gim Hee Lee[3]

[1]Sea AI Lab [2]Garena [3]National University of Singapore

In this supplementary document, we provide the implementation details of our network's MLP heads in Section 1. Then, we provide the mathematical details of the Normal Consistency metrics in Section 2. We then demonstrate the capability of TreeMeshGPTto generate artistic meshes from text prompts through a multi-step process in Section 3. Finally, we present our 9-bit model supporting the generation of artistic meshes with up to 11,000 faces in Section 4.

1. Vertex Prediction Heads

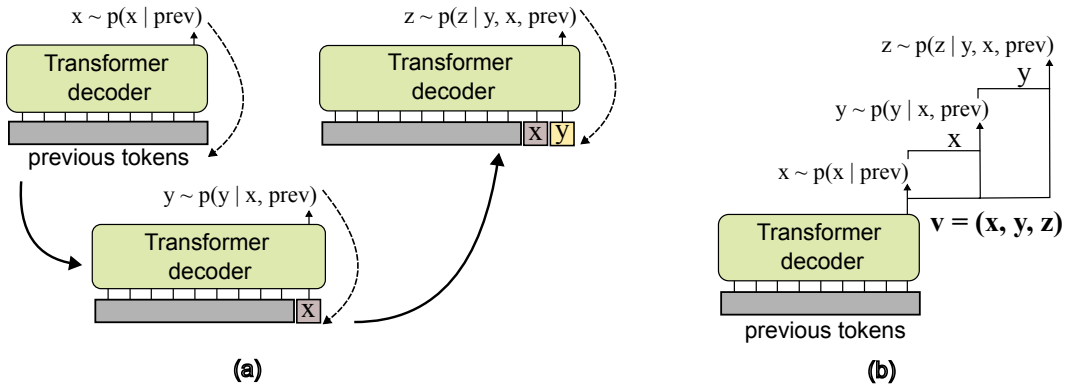

Figure 1. **Sequential vertex prediction.** a). Next-token prediction. b). Our hierarchical MLP heads.

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 the 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 $g_{\theta 1}$, 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:

$$x \sim p(x \mid \text{prev}) = g_{\theta 1}(\mathbf{c}). \tag{1}$$

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 $g_{\theta 2}$:

$$y \sim p(y \mid x, \text{prev}) = g_{\theta 2}(\text{E}_x(x), \mathbf{c}), \tag{2}$$

where $\text{E}_* \in \mathbb{R}^d$ denotes the learnable embeddings for the discretized coordinates of an axis and $*$ can represent x, y, or z. For example, $\text{E}_x(x)$ represents the embedding of the discretized x-coordinate, and similarly, $\text{E}_y(y)$ and $\text{E}_z(z)$ 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, $g_{\theta 3}$, which takes as input the latent code $\mathbf{c}$ along with the embeddings of both previously predicted coordinates, $\mathrm{E}_x(x)$ and $\mathrm{E}_y(y)$:

$$z \sim p(z \mid y, x, \text{prev}) = g_{\theta 3}(\mathrm{E}_y(y), \mathrm{E}_x(x), \mathbf{c}). \tag{3}$$

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.

2. 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:

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

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:

$$\mathrm{NC} = \frac{1}{2|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} \mathrm{Sim}_{i \to j}(\mathbf{n}^s, \mathbf{n}^r) + \frac{1}{2|\mathcal{M}_r|} \sum_{k \in \mathcal{M}_r} \mathrm{Sim}_{k \to l}(\mathbf{n}^r, \mathbf{n}^s), \tag{4}$$

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 ($|\mathrm{NC}|$) that omits the flipping direction is then given as:

$$|\mathrm{NC}| = \frac{1}{2|\mathcal{M}_s|} \sum_{i \in \mathcal{M}_s} |\mathrm{Sim}_{i \to j}(\mathbf{n}^s, \mathbf{n}^r)| + \frac{1}{2|\mathcal{M}_r|} \sum_{k \in \mathcal{M}_r} |\mathrm{Sim}_{k \to l}(\mathbf{n}^r, \mathbf{n}^s)|. \tag{5}$$

3. 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 2. We utilize the Luma AI Genie[1] 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.

Figure 2. **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.

4. 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 3. Compared to the 7-bit model, our 9-bit model can generate artistic meshes with smoother surfaces, finer details, and higher number of faces.

[1] https://lumalabs.ai/genie

Figure 3. **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.