MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers – Supplementary Document –

Yawar Siddiqui[1] Antonio Alliegro[2] Alexey Artemov[1]
Tatiana Tommasi[2] Daniele Sirigatti[3] Vladislav Rosov[3] Angela Dai[1] Matthias Nießner[1]
Technical University of Munich[1] Politecnico di Torino[2] AUDI AG[3]

In this supplementary document, we discuss additional details about our method MeshGPT. We provide implementation details of our method, loss functions, and the baselines in Section 2. Additional details about the user study are provided in Section 3. We also provide further qualitative and quantitative results (Section 5), including a shape novelty analysis (Section 4) for shapes from the main paper. Additional limitations are discussed in Section **??**. We further encourage the readers to check out the supplemental video for a summary of the method and an overview of results.

1. Data

Selection. We use the ShapeNetV2 [2] dataset for all our experiments. We first apply planar decimation to each shape using Blender [4], with the angle tolerance parameter α set within $[1, 60]$. The impact of this decimation is assessed by calculating the Hausdorff distance [1] between the decimated and original shapes. We then choose, for each original shape, the decimated version with the Hausdorff distance closest to, but below, a pre-set threshold $\delta_{\text{hausdorff}}$. Shapes with more than 800 faces are excluded, resulting in a final count of 28980 shapes across all categories. The Chair, Table, Bench, and Lamp categories are further divided into a 9:1 train-test split. All shapes from rest of the categories are used for pretraining phase, while only the training subset from specific categories is used for pretraining and finetuning. All shapes are normalized to be centered at the origin and scaled to ensure the longest side is of unit length.

Augmentation. During the training of both the encoder-decoder and the transformer, multiple augmentation techniques are applied to all train shapes. Scaling augmentation, ranging from 0.75 to 1.25, is independently applied across each axis. Post-scaling, meshes are resized to keep the longest side at unit length. Additionally, jitter-shift augmentation in the range of $[-0.1, 0.1]$ is used, adjusted to maintain the mesh within the unit bounding box around the origin. We also implement varying levels of planar decimation for training shapes, provided the distortion remains below $\delta_{\text{hausdorff}}$.

2. Method Details

2.1. Architecture

The architecture of our encoder-decoder network is elaborated in Fig. 4. The encoder comprises a series of SAGEConv [8] graph convolution layers, processing the mesh in the form of a face graph. For each graph node, input features include the positionally encoded 9 coordinates of the face triangle, its area, the angles between its edges, and the normal of the face. The decoder is essentially a 1D ResNet-34 [9] network, applied to the face features interpreted as a 1D sequence. It outputs logits corresponding to the 9 discrete coordinates of each face triangle, which are discretized within a 128^3 space. The codebook $\mathcal{C}$ has a size of 16384. The architecture of the transformer is simply a GPT-2 medium architecture, i.e. 24 multi-headed self attention layers, 16 heads, 768 as feature width, with context length of 4608.

2.2. Residual Vector Quantization

Fundamentals. For quantization, we employ residual vector quantization (RQ) [10, 11]. RQ discretizes a vector $\mathbf{z}$ with a stack of D ordered codes. Starting with the 0^{th} residual $\mathbf{r}^0 = \mathbf{z}$, RQ recursively computes t^d as the code of the residual $\mathbf{r}^{d-1}$, and the next residual $\mathbf{r}^d$ as

$$t^d = \mathcal{Q}\left(\mathbf{r}^{d-1}; \mathcal{C}\right) \tag{1}$$

$$\mathbf{r}^d = \mathbf{r}^{d-1} - \mathbf{e}(t^d) \tag{2}$$

where $\mathcal{Q}(\mathbf{z}; \mathcal{C})$ denotes vector quantization of $\mathbf{z}$ with codebook $\mathcal{C}$, and $\mathbf{e}(t^d)$ is the embedding in the codebook $\mathcal{C}$. Further, we define

$$\hat{\mathbf{z}}^{(d)} = \sum_{1}^{d} \mathbf{e}(t^d) \tag{3}$$

as the partial sum of up to d code embeddings, and $\hat{\mathbf{z}} = \hat{\mathbf{z}}^D$ is the quantized vector of $\mathbf{z}$. The recursive quantization of

Figure 1. Additional novel shapes on Chairs, Tables, Benches and Lamps generated by our method.

RQ thus approximates the vector $\mathbf{z}$ in a coarse-to-fine manner [10]. The commitment loss can now be defined between vector $\mathbf{z}$ and its quantization $\hat{\mathbf{z}}$ as

$$\mathcal{L}_{\text{commit}}(\mathbf{z}, \hat{\mathbf{z}}) = \sum_{d=1}^{D} \|\mathbf{z} - \text{sg}[\hat{\mathbf{z}}^{(d)}]\|_2^2 \tag{4}$$

where sg denotes the stop gradient operation.

Per Vertex Residual Vector Quantization. Instead of directly applying RQ, for a face feature $\mathbf{z_i}$ extracted by the graph encoder, we first split this 576 dimension face feature $\mathbf{z_i}$ into 3 features, $(\mathbf{z}_i^1, \mathbf{z}_i^2, \mathbf{z}_i^3)$, each of 192 dimensions representing the features of the face triangle's 3 vertices. The features the fall on vertices that are shared across faces are averaged. On these per vertex index feature $\mathbf{z}_i^j$, RQ quantizes them into a stack of $\frac{D}{3}$ features,

$$\text{RQ}(\mathbf{z_i}; \mathcal{C}, D) = (\text{RQ}(\mathbf{z}_i^1; \mathcal{C}, \frac{D}{3}), \ldots, \text{RQ}(\mathbf{z}_i^3; \mathcal{C}, \frac{D}{3})) \tag{5}$$

for codebook $\mathcal{C}$, with

$$\text{RQ}(\mathbf{z_i^j}; \mathcal{C}, \frac{D}{3}) = (t_i^{2j-\frac{D}{3}+1}, t_i^{2j-\frac{D}{3}+2}, \ldots, t_i^{2j}), \tag{6}$$

where t_i^d is the index to the embedding $\mathbf{e}(t_i^d)$ in the codebook $\mathcal{C}$. Taken together for each vertex, these form a stack of D features,

$$\text{RQ}(\mathbf{z_i}; \mathcal{C}, D) = (t_i^1, t_i^2, \ldots, t_i^D) = t_i. \tag{7}$$

Thus, the residual quantization for the features extracted for all the N faces of the mesh $\mathbf{Z} = (z_1, z_2, \ldots, z_N)$ is given as

$$\text{RQ}(\mathbf{Z}; \mathcal{C}, D) = \text{RQ}(\mathbf{z_1} \ldots \mathbf{z_N}; \mathcal{C}, D) \tag{8}$$

$$\text{RQ}(\mathbf{z_1} \ldots \mathbf{z_N}; \mathcal{C}, D) = (t_0, t_1, \ldots, t_N). \tag{9}$$

Fig. 6 gives an intuition on why 'per vertex' tokenization is better than 'per face' tokenization, with ablations in the main paper confirming it.

Figure 2. Shape novelty analysis on ShapeNet [2] chair and table category for shapes generated by our method shown in main paper. We show the 3 nearest neighbors in terms of Chamfer Distance (CD) for a generated shape. Shapes are scaled to a unit length along all axes to account for augmented generations before computing CD.

2.3. Loss Functions

Vocabulary Learning. Let $\mathcal{P}_{nijk}$ be the predicted probability distribution over the discrete coordinates, where n is the face index, i is the vertex index inside the face, j is the coordinate's axis index (x, y or z), and k goes over the discretized positions $\in \{1, 2, 3, \ldots, 128\}$. If V_{nij} is the target discretized position, then the reconstruction loss for the encoder-decoder network is given as

$$\mathcal{L}_{\text{recon}} = \sum_{n=1}^{N} \sum_{i=1}^{3} \sum_{j=1}^{3} \sum_{k=1}^{128} \mathrm{w}_{nijk} \log \mathcal{P}_{nijk} \tag{10}$$

with

$$\mathrm{w}_{nijk} = \text{smooth}\left(\text{one-hot}_{128}\left(V_{nij}\right)\right) \tag{11}$$

is a smoothening kernel applied across the one-hot probability distribution over the targets, encouraging physically close coordinates to be penalized less. The loss over the encoder-decoder network is the sum of $\mathcal{L}_{\text{recon}}$ and $\mathcal{L}_{\text{commit}}$ previously described.

Transformer. Given a target sequence $\mathbf{T} = (t_0, t_1, \ldots, t_N)$ with $t_i = (t_i^1, t_i^2, \ldots, t_i^D)$, and s_i^j is the corresponding predicted sequence element, then the transformer is trained with the loss

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^{N} \sum_{j=1}^{D} \sum_{k=1}^{|\mathcal{C}|} \log p(s_i^k = t_i^j). \tag{12}$$

2.4. Baselines

We utilize the official implementations for BSPNet [3], AtlasNet [7], and GET3D [6]. For Polygen [12], we re-implement it following the details in their paper. To align its architecture with our method, we employ the same GPT2-medium architecture for the vertex model in Polygen. Ad-

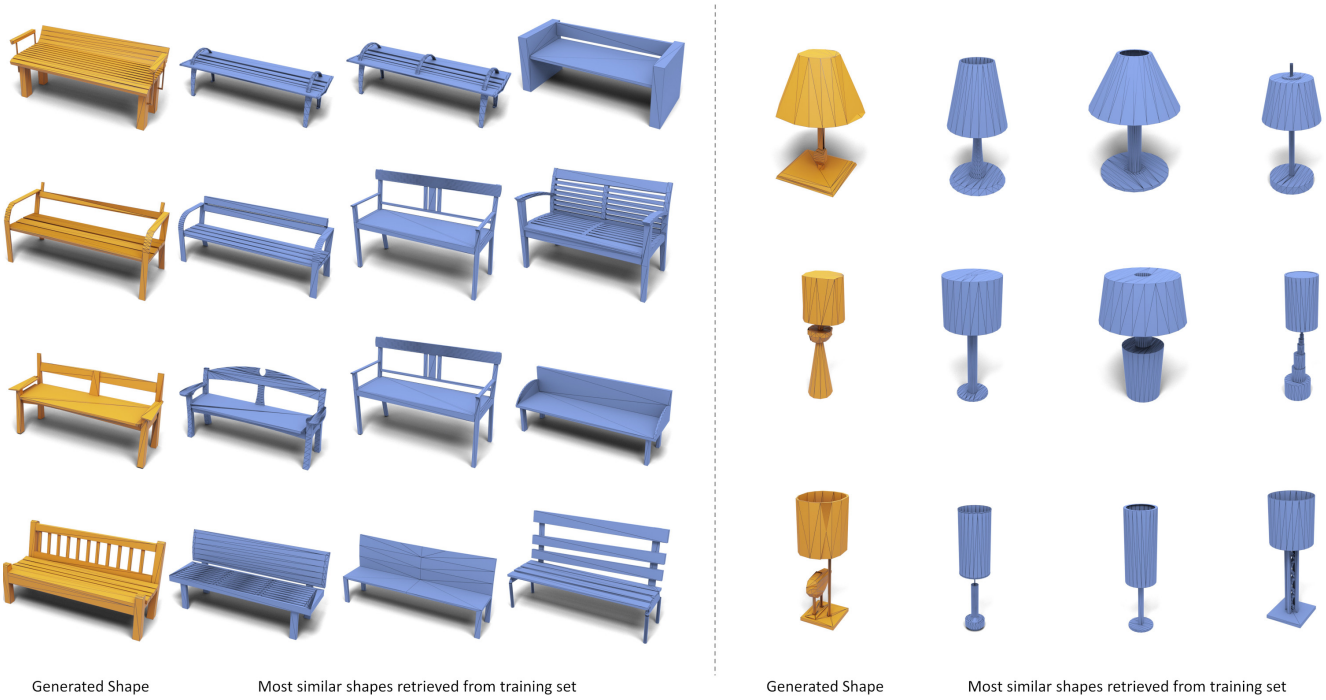

Figure 3. Shape novelty analysis on ShapeNet [2] bench and lamp category for shapes generated by our method shown in main paper. We show the 3 nearest neighbors in terms of Chamfer Distance (CD) for a generated shape. Shapes are scaled to a unit length along all axes to account for augmented generations before computing CD.

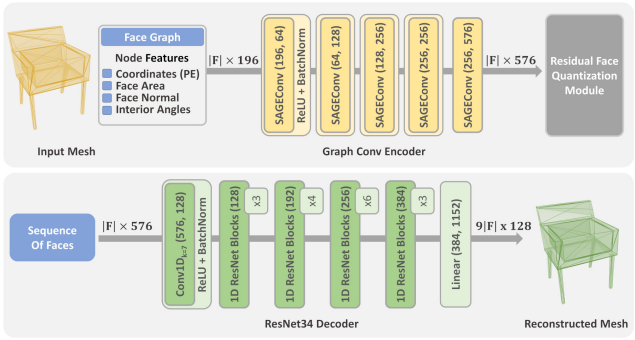

Figure 4. Our encoder-decoder network features an encoder with SAGEConv [8] layers processing mesh faces as a graph. Each node inputs positionally encoded face triangle coordinates, area, edge angles, and normal. The decoder, a 1D ResNet-34 [9], interprets face features as a sequence, outputting logits for the discretized face triangle coordinates in a 128^3 space.

ditionally, mirroring our approach, Polygen undergoes pretraining on all categories and is finetuned for each evaluated category, applying the same train-time augmentations as used in our method.

3. User Study Details

We develop a Django-based web application for the user study. In Fig. 5, we show the interface for the questionnaire. We randomly select 16 pairs of meshes from each baseline and our method across the Chair and Table categories, half of which are used for a question on preference based on shape quality, and the other half for preference based on triangulation quality. After the samples are prepared, we ask the users to pick the sample which they prefer more based on the question. To avoid biases in this user study, we shuffle the pairs so that there is no positional hint to our method. We also show a collection of ground-truth meshes to the user for them to get an idea of the real distribution. In the end, we gather 784 responses from 49 participants to calculate the preferences.

4. Shape Novelty Analysis

Fig. 2 and 3 displays the top-3 most similar shapes from the train set corresponding to all samples used in the main paper that were generated by our model. These nearest neighbor shapes are identified based on Chamfer Distance (CD). To ensure a fair distance computation, accounting for potential discrepancies due to scale or shift in the augmented generations, we normalize all generated and train shapes to be centered within $[0, 1]^3$ and scaled to the extremes of this cube.

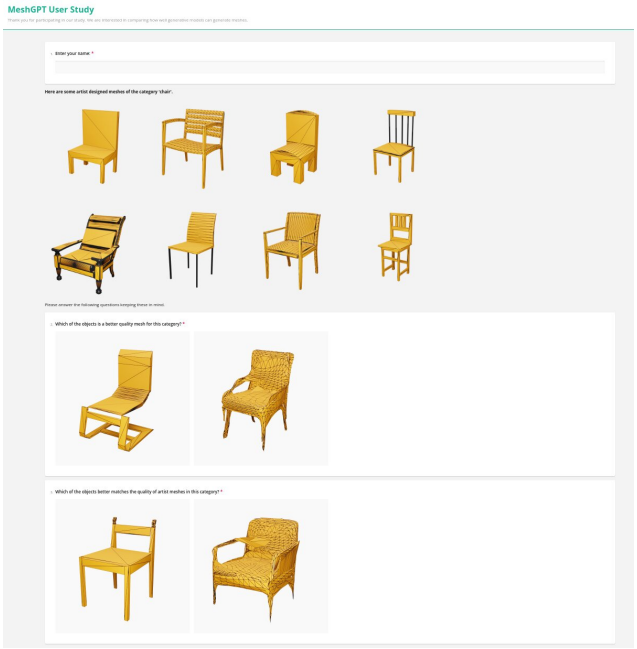

Figure 5. User study interface. We show users a set of random ground-truth shapes for a category and then ask users for shape quality and triangulation preference among meshed generated by two methods.

5. Additional Results

Metrics. Following recent works for unconditional shape generation [5, 15, 16] for calculating the shape metrics we define

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y),$$

$$\text{COV}(S_g, S_r) = \frac{|\{\text{argmin}_{Y \in S_r} D(X, Y) | X \in S_g\}|}{|S_r|},$$

$$\text{1-NNA}(S_g, S_r) = \frac{\sum_{X \in S_g} \mathbb{1}_X + \sum_{Y \in S_r} \mathbb{1}_Y}{|S_g| + |S_r|},$$

$$\mathbb{1}_X = \mathbb{1}[N_X \in S_g],$$

$$\mathbb{1}_Y = \mathbb{1}[N_Y \in S_r],$$

where in the 1-NNA metric N_X is a point cloud that is closest to X in both generated and reference dataset, i.e.,

$$N_X = \underset{K \in S_r \cup S_g}{\text{argmin}} \; D(X, K)$$

We use a Chamfer Distance (CD) distance measure $D(X, Y)$ for computing these metrics in 3D. To evaluate these point-based measures, we sample 2048 points randomly from all baseline outputs; and use 6000, 1200, 1000, 8000 generated shapes from chair, bench, lamp and table categories.

Variant	Triangle Accuracy (%) ↑	Cross-Entropy ↓
w/o Positional Encoding	79.33	0.2484
w/o Output Discretization	22.03	0.5705
w/o Residual Quantization	1.29	4.6679
w/o per Vertex Quantization	**98.64**	**0.1413**
w/ PointNet Encoder	88.73	0.1896
w/ GAT [13] Encoder	86.14	0.2015
w/ EdgeConv [14] Encoder	91.23	0.1702
w/ ResNet19 Decoder	96.29	0.1492
w/ PointNet Decoder	95.47	0.1528
MeshGPT	98.49	0.1473

Table 1. Ablations of our design choices for the encoder-decoder network on the Chair category of the ShapeNet [2] dataset.

Qualitative Results. Fig. 1 shows more unconditional generations from our model across different ShapeNet categories.

Encoder-Decoder Ablations. In Tab. 1, we show a set of ablations on the design choice for our encoder-decoder network used for learning the triangle embeddings. We measure the performance in terms of triangle accuracy, which measures average accuracy with which all 9 coordinates of faces are correctly predicted, and the cross-entropy loss on the test set.

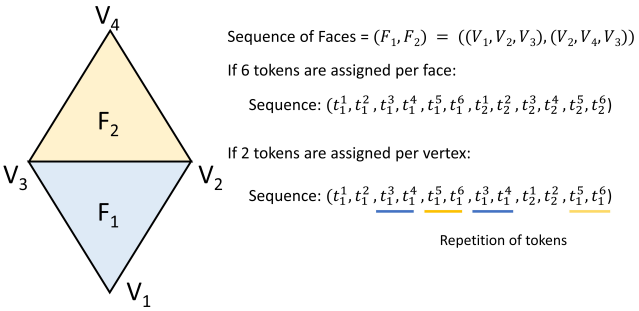

Figure 6. The effectiveness of per-vertex quantization over per-face quantization can be understood through an example where two faces share an edge as shown above. With per-face tokenization assigning 6 tokens per face, the sequence yields 12 unique tokens. In contrast, per-vertex tokenization leads to repeated tokens in the sequence due to shared vertices between faces. This repetition makes the sequence easier for the transformer to learn compared to a wholly unique sequence per face, especially when both sequences are of equal length.

We evaluate the effect of various choices – how much does the positional encoding at input help, effect of using continuous predictions instead of discrete as outputs, using vector quantization (1 token per face) instead of residual quantization (D tokens per face), encoder architecture as a point encoder, or different graph convolution operators, and decoder architecture as either ResNet19 or PointNet decoder. Note that even though for encoder-decoder reconstruction, 'w/o per Vertex Quantization' performs best, this variant works significantly worse than with per Vertex

Quantization, as shown in the main paper. Fig. 6 describes an intuition of why the embeddings from this variant are more transformer friendly.

6. Limitations

Our attention context spans 4608 tokens which covers a max length of $\frac{4608}{6} \sim 800$ faces (6 tokens/face). To scale to much larger scenes, future work could consider context window expansion (Code-LLMs), hierarchical transformers, or scalable sequential architectures (SparseAttention, Mamba). Furthermore, similar to all baselines with the exception of BSPNet, we do not ensure watertight meshes. Furthermore, even human-designed meshes (e.g., from ShapeNet) may not always be watertight. If watertightness is essential, postprocessing (e.g., Manifold++) can be applied. Furthermore, the errors in autoregressive sequence prediction and decoding can result in bad meshes.

References

[1] Henry Blumberg. Hausdorff's grundzüge der mengenlehre. 1920. 1

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1, 3, 4, 5

[3] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 45–54, 2020. 3

[4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 1

[5] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. *arXiv preprint arXiv:2303.17015*, 2023. 5

[6] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. *Advances In Neural Information Processing Systems*, 35:31841–31854, 2022. 3

[7] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. 3

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 1, 4

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 4

[10] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11523–11532, 2022. 1, 2

[11] Julieta Martinez, Holger H Hoos, and James J Little. Stacked quantizers for compositional vector compression. *arXiv preprint arXiv:1411.2173*, 2014. 1

[12] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR, 2020. 3

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 5

[14] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019. 5

[15] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19313–19322, 2022. 5

[16] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. In *Advances in Neural Information Processing Systems*, 2022. 5