

Nautilus: Locality-aware Autoencoder for Scalable Mesh Generation

Appendix

The *Appendix* is organized as follows:

- **Section 1:** gives a step-by-step demonstration of our Nautilus-style Tokenization algorithm.
- **Section 2:** further provides additional experimental results, comparisons, and analyses, including extended visualizations, quantitative evaluations of topological quality, and more in-depth quantitative ablation studies.
- **Section 3:** provides in-depth discussions on Nautilus, specifically detailing the methodology and experimental comparisons with EdgeRunner.
- **Section 4:** presents the statistical analysis conducted on our training dataset.
- **Section 5:** elaborates more details about our user study.
- **Section 6:** discusses the limitation of our approach.

We also include a *demonstration video* in our supplementary materials for better visualization.

1. Tokenization: Step-by-Step Demonstration

In our main paper, we design a novel Nautilus-style Tokenization algorithm that preserves local dependency while achieving effective sequence length compression. In Algorithm 1, we provide a step-by-step demonstration of our tokenization algorithm, including the initial sorting of vertices and faces, shell construction by vertices traversal, and the coordinates compression of traversed vertices.

2. Extensive Experiments and Analysis

Our experiments in the main paper comprehensively evaluate the effectiveness of **Nautilus**. In this section, we provide extensive experiments and analysis to explore behind its efficacy:

- Sec. 2.1: We present more generation results of Nautilus from both point cloud condition and image condition.
- Sec. 2.2: We provide additional quantitative analysis to validate the topological quality of our generated results in comparison with existing methods.
- Sec. 2.3: We present an additional quantitative ablation study to complement the qualitative ablation comparisons in the main paper.
- Sec. 2.4: We exclusively examine the impact of local dependency on tokenization representation by comparing

Algorithm 1 Step-by-Step Pipeline of Nautilus-style Tokenization

- 1: **Input:** manifold mesh asset $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ consisting of vertices $\mathcal{V} = \{v_i\}$ and faces $\mathcal{F} = \{f_i\}$.
- 2: Sort vertices \mathcal{V} by their z - y - x coordinates.
- 3: Sort faces \mathcal{F} by the smallest vertex index in each face.
- 4: Initialize a list of unvisited faces $U_f \leftarrow \mathcal{F}$.
- 5: Compute the degree $\deg(v_i)$ for all $v_i \in \mathcal{V}$, where $\deg(v_i)$ represents the number of edges connected to vertex v_i .
- 6: Select the highest-degree vertices in the first face $f_1 \in U_f$ as the initial center O_1 .
- 7: Initialize the tokenized sequence $S(\mathcal{M}) \leftarrow \emptyset$ and set shell index $k \leftarrow 1$.
- 8: **while** $U_f \neq \emptyset$ **do**:
- 9: Identify all faces in U_f that contain O_k .
- 10: Sort these faces in adjacent order, extract their surrounding vertices $\{P_{k,i}\}$.
- 11: Extend $S(\mathcal{M}) \leftarrow S(\mathcal{M}) \cup \{O_k, P_{k,1}, \dots, P_{k,\text{end}}\}$, shell k complete.
- 12: Identify all neighbor vertices of $P_{k,\text{end}}$, denoted as $\mathcal{N}(P_{k,\text{end}}) = \{N_{k,\text{end}}^i\}$.
- 13: Find the vertex with the maximum degree:

$$N_{k,\text{end}}^{\max} = \arg \max_{N \in \mathcal{N}(P_{k,\text{end}})} \deg(N).$$

- 14: **If** $\deg(N_{k,\text{end}}^{\max}) > 4$ (ensure at least 3 connected faces in the next shell):
 - 15: Set $O_{k+1} \leftarrow N_{k,\text{end}}^{\max}$.
 - 16: **Else**:
 - 17: Set O_{k+1} as highest-degree vertex in the first remaining face $f_1 \in U_f$.
 - 18: Update $\deg(v_i)$ for all $v_i \in \mathcal{V}$ to reflect the remaining unvisited edges.
 - 19: Remove all visited faces from U_f , $k \leftarrow k + 1$.
 - 20: Convert the discrete coordinates $(x, y, z) \in \mathbb{R}^{128}$ of vertices in $S(\mathcal{M})$ to (u, v) .
 - 21: Flatten the vertices represented in (u, v) coordinates to 1D sequence.
 - 22: Codebook mapping $S(\mathcal{M})$: $u_k^p = u_k^p$, $v = v + 1024$, $u_k^o = u_k^o + 1024 + 2048$, where 1024 is the codebook size of u_k^p and 2048 is the codebook size of v .
 - 23: **Output:** Tokenized sequence $S(\mathcal{M})$.
-

our Nautilus Shell representation with an upgraded version of the AMT representation that equips our coordinate compression.

- Sec. 2.5: We examine the local dependencies modeling by visualizing the attention weights in the generator.

2.1. More Qualitative Results

In Figure 1 and Figure 2, we present more generated sample from both point cloud condition and image condition.

2.2. Quantitative Study on Topological Quality

To further validate the performance improvement in locality preservation, we introduce additional quantitative metrics to assess topological quality. Since no such metric is included in prior works, we adopt two traditional measures and report

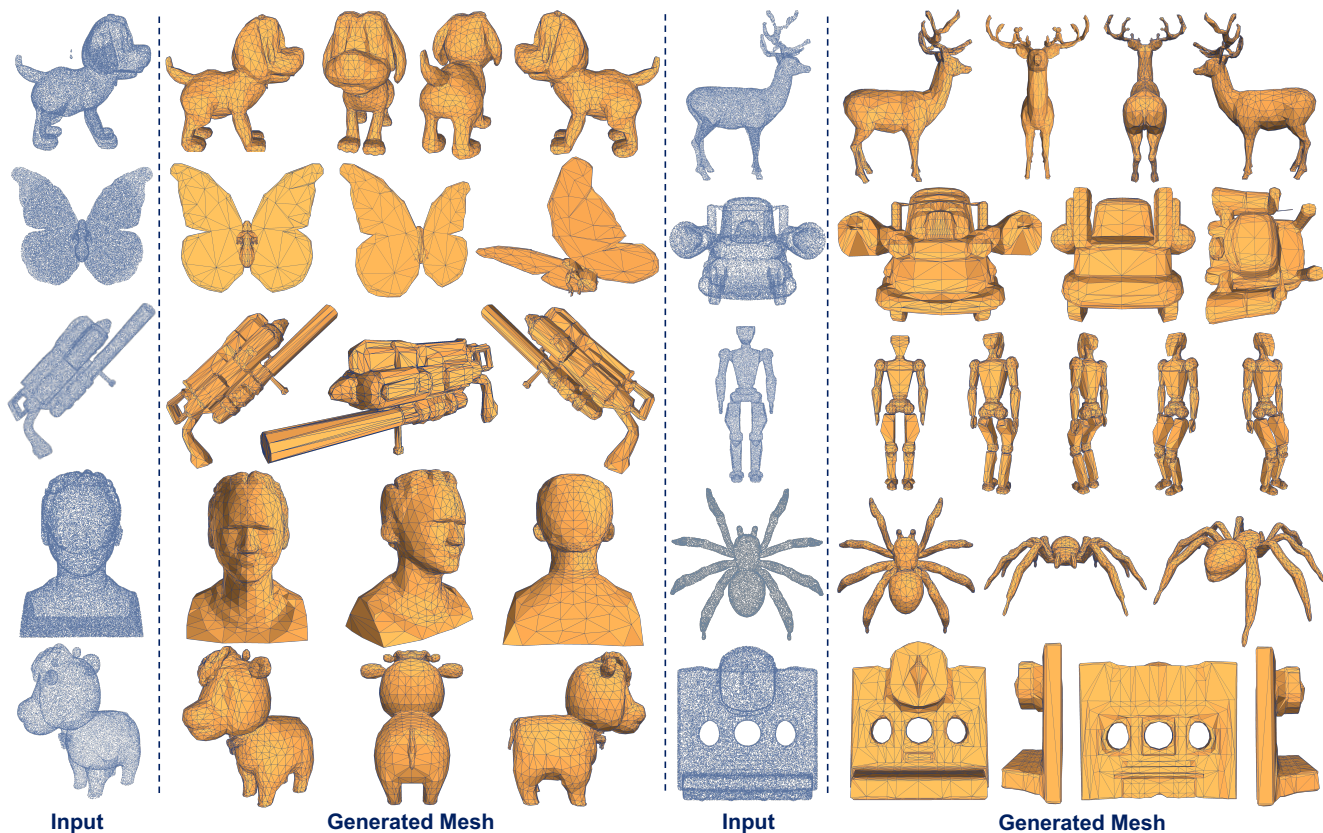


Figure 1. We release additional point cloud conditioned generation results of our Nautilus across an extensive range of object categories.

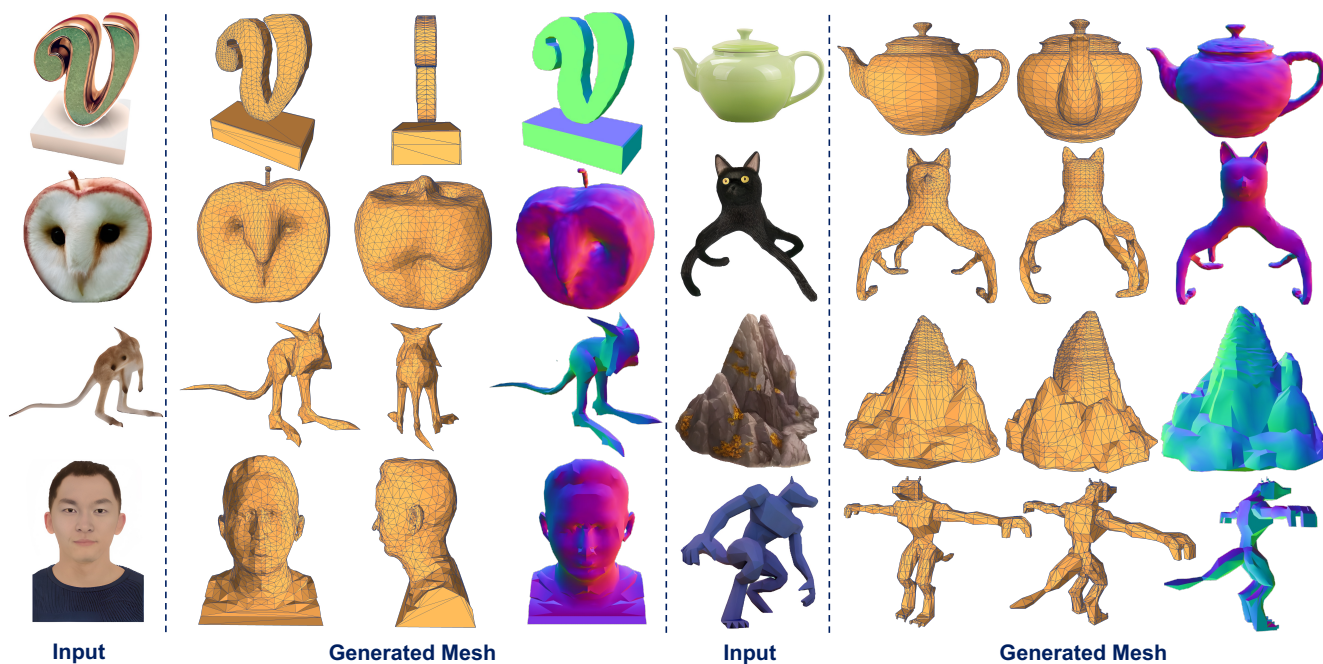


Figure 2. We release additional image conditioned generation results of our Nautilus across an extensive range of object categories. The results are presented alongside corresponding normal maps to highlight the topology.

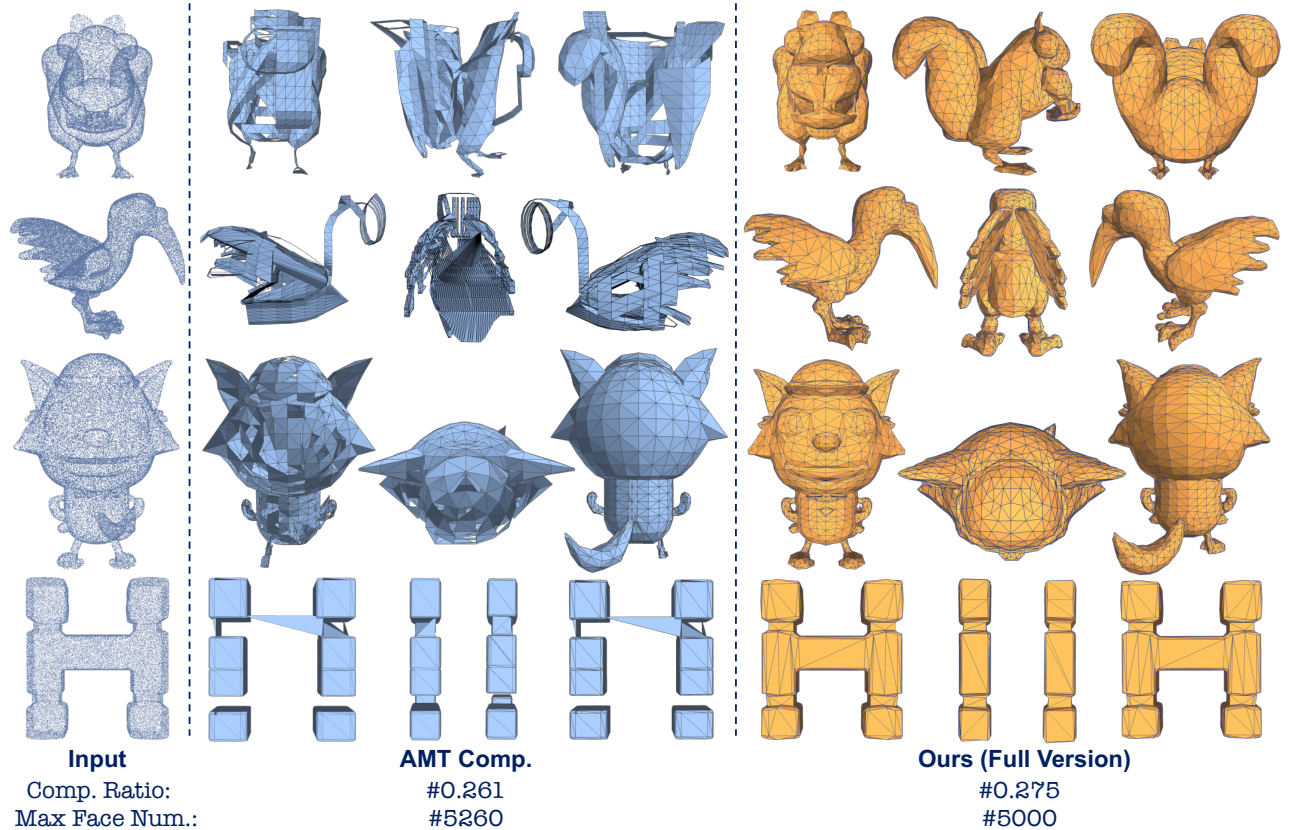


Figure 3. Ablated comparison of generated results between the representation of AMT and our Nautilus Shell representations. To independently analyze the influence of local dependency preservation, both groups are trained on the same dataset, and AMT is improved by incorporating the same coordinate compression as Nautilus to achieve a comparable number of faces, denoted as *AMT Comp.*

Methods	MeshAnything	MeshAnythingV2	Nautilus (Ours)
Surf. Holes ↓	45.0%	40.6%	4.2%
Inters. Faces ↓	16.0%	16.8%	9.6%
Miss. Parts ↓	70.4%	58.4%	8.0%
Manifold ↑	23.8%	29.0%	83.6%

Table 1. Quantitative Comparison on Topological Quality with latest open-source methods, where our Nautilus framework significantly outperform others in all topological metrics.

statistics based on the same 500 point-cloud-conditioned generation results compared in our main paper. Specifically, we identify surface holes by analyzing boundary edges and use PyMeshLab to detect intersecting faces in the generated results. Additionally, to assess the completeness of generation, we manually inspect for missing parts in the generated meshes. If no such defects or missing parts are found in a generated mesh asset, we classify it as manifold. Finally, we compute the ratio of meshes exhibiting any of the three defects as well as the proportion of manifold meshes. Notably, a single mesh may exhibit multiple defects simultaneously.

We present the results in Table 1, which show that our

method achieves a significant improvement across various topological metrics. This is attributed to our approach’s strong locality preservation and high compression rate, enabling the generator model to learn finer local structures from complex topology samples and ultimately produce meshes with higher structural fidelity.

2.3. Quantitative Ablation Study

In our main paper, we conduct a qualitative ablation study on point-cloud-conditioned generation. We here extend the analysis to a quantitative evaluation on the same high-quality test set of 500 samples used in main paper. In Table 2, we compare four configurations: using only the Coordinate Compression (*Comp. only*), using only the Nautilus Shell Representation (*Rep. only*), using our *Full Tokenization* but without the local encoder, and our *Full Version*.

To assess the generation quality, we follow a similar evaluation protocol: uniformly sampling 1,024 points from the surfaces of both the ground truth and generated meshes. The Chamfer Distance (C.Dist.) and Hausdorff Distance (H.Dist.) are then computed between these sampled point sets to quantify the reconstruction accuracy. Lower values

of these metrics indicate a closer alignment between the predicted and ground truth meshes, reflecting higher fidelity to the input conditions.

Ablation	Comp. only	Rep. only	Full Tokenization	Full Version
C.Dist. ↓	0.151	0.100	0.092	0.087
H.Dist. ↓	0.355	0.213	0.186	0.176

Table 2. We conduct quantitative ablation comparison on four specific configurations: using only the Coordinate Compression (*Comp. only*), using only the Nautilus Shell Representation (*Rep. only*), using our *Full Tokenization* but without the local encoder, and our *Full Version*.

As shown in Table 2, our findings are consistent with the qualitative ablation study, confirming that the *Full Version* outperforms all ablated configurations. Notably, the Nautilus Shell representation provides the most substantial improvement (see *Rep. only* vs. *Comp. only*), while Coordinate Compression further contributes to the mesh quality (see *Full Tokenization* vs. *Rep. only*). Additionally, the inclusion of our Local Encoder in the conditioner offers a significant performance boost (see *Full Tokenization* vs. *Full Version*). Our findings confirm that achieving both compression and preservation of local dependency, as implemented in our Nautilus, is essential to generate high-quality meshes.

2.4. Ablation Analysis with AMT Representation

To further investigate the influence of **local dependency preservation** of tokenization algorithms on generation quality, we perform additional experimental comparisons between our Nautilus Shell representation and the AMT in MeshAnythingV2, while excluding the influence of different training data and compression ratio.

Specifically, we apply the same coordinate compression to the AMT representation and train it on the same dataset as Nautilus, enabling it to model more than 5,000 faces. In Figure 3, we denote the improved AMT version as *AMT Comp.*, comparing it with our full version on challenging samples with complex topology and rich geometric details. Despite achieving a compression ratio comparable to that of Nautilus, *AMT Comp.* still exhibits severe manifold defects in its generation results, particularly **isolated strips** consisting of single-line faces.

This issue arises from the lack of local dependency preservation in AMT’s traversal. Unlike the part-by-part approach taken by human artists, AMT processes mesh faces in elongated strips, making it extremely challenging for the model to precisely connect these strips into manifold meshes. Without preserving local dependency, for each face in the sequential generation, its neighboring faces in the mesh are often located far apart in the sequence. This requires the decoder to build highly precise features to glob-

ally align and gather information from these distant neighbors. As these neighbors are essential for determining the shape of the current face according to mesh locality, failing to aggregate their information leads to poor interconnections and manifold defects.

In contrast, Nautilus preserves short-distance, local dependency, maintaining the proximity of spatially neighboring vertices in the tokenized sequence. During the generation of each face, this local dependency introduces an inductive bias, facilitating the decoder to simply gather information from nearby positions in the sequence. This approach enables Nautilus to achieve significantly better local interconnection and structural fidelity. Our results highlight the importance of preserving local dependency in tokenization algorithms for generating high-quality meshes. Additional visualization analysis is provided in Sec. 2.5.

2.5. Visualized Analysis on Local Dependency

In Supp. Sec. 2.4, we conducted additional ablation comparisons to demonstrate the significance of preserving local dependency within tokenization algorithms. In this section, we present a visualized analysis to directly examine the modeling of local dependency. Specifically, we compare the averaged attention maps of our Nautilus and that of the ablated group *AMT Comp.* designed in Supp. Sec. 2.4. For each model, we compute the averaged causal attention map across 50 samples and all 24 self-attention layers in the transformer decoder, in order to evaluate the models’ capacity to model local dependency.

We present our findings in Figure 4, where each row represents a query (Q) position and each column represents a key (K) position. Bright colors indicate stronger attention weights directed from a Q position to a K position. From Figure 4, it is evident that Nautilus exhibits highly concentrated bright regions close to the diagonal. In other words, for each query position, the attention is predominantly focused on a few nearby positions within the sequence. In contrast, the attention map of *AMT Comp.* shows more dispersed bright regions, suggesting that attention is distributed to positions farther away in the sequence compared to Nautilus. This visualization clearly demonstrates the impact of local dependency preservation on the performance of decoder. By effectively maintaining the proximity of spatially neighboring vertices in the tokenized sequence, the transformer decoder in Nautilus gathers local topology information more efficiently from nearby positions within the sequence, which contributes to its superior quality.

3. Further Discussions

3.1. Comparison with EdgeRunner

Due to the unavailability of checkpoints and the absence of normals in their point cloud demo, we are unable to

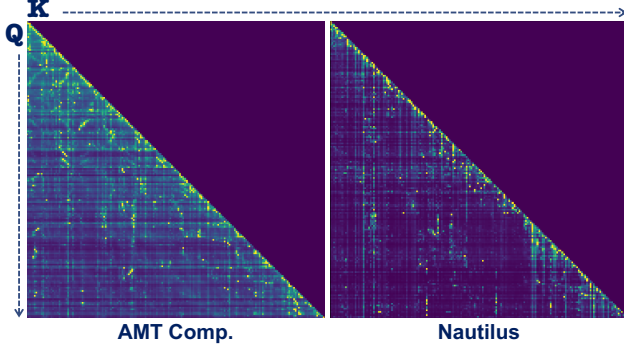


Figure 4. Comparison of averaged Attention Maps between *AMT Comp.* and *Nautilus*. The attention maps are averaged over 50 samples and all 24 self-attention layers in the decoder. Each row corresponds to a query (Q) position, and each column corresponds to a key (K) position. Bright colors indicate stronger attention weights directed from a Q position to a K position.

conduct a direct comparison with EdgeRunner under point cloud conditions. Therefore, in this section, we provide a methodological and experimental comparison to highlight the differences between our approach and EdgeRunner.

3.1.1. Methodological Comparison with EdgeRunner

In terms of methodology, both *Nautilus* and *EdgeRunner* follow the auto-regressive mesh generation paradigm pioneered by *MeshGPT*, yet they share few similarities beyond that. While *EdgeRunner* propose using the half-edge representation, our *Nautilus* approaches focus on preserving locality structure and exploiting it for more advanced tokenization compression.

Tokenization Algorithm. *EdgeRunner* is built on the half-edge algorithm and relies on an excessive number of separation tokens for faithful traversal, resulting in a suboptimal compression ratio and locality preservation (as shown in Table 1 of our main paper). Limited compression capability constrains the maximum number of faces and increases training costs, while weaker locality preservation makes the generated results more susceptible to local structural defects and the loss of fine geometric details. On the other hand, *Nautilus* leverages shared vertices and edges within mesh locality and proposes *Nautilus*-style representation with continuous shell traverse. This design naturally aligns with the mesh creation flow and eliminates the need for excessive separation tokens, allowing it to surpass *EdgeRunner* in both locality preservation and sequence compactness. As a result, our approach achieves lower training costs, a higher maximum face count, and finer local topological quality, which is demonstrated in Figure 7 in main paper.

Conditioning Mechanism. For point cloud conditioning, *EdgeRunner* employs a global encoder similar to previous methods, while we introduce a novel PointConv-based Local Encoder along with a seamless injection scheme that

integrates local features into our *Nautilus*-shell representation, effectively enhancing local dependency.

3.1.2. Experimental Comparison with EdgeRunner

Qualitative and Quantitative Comparison. In our main paper, we made a significant effort to conduct performance comparisons as comprehensively as possible. Our quantitative analysis demonstrates that our tokenization algorithm surpasses *EdgeRunner* in both compactness and locality preservation, with a compression capability nearly twice that of *EdgeRunner*. Furthermore, as shown in Figure 7, *Nautilus* achieves superior qualitative results, significantly outperforming *EdgeRunner* in image-conditioned generation by producing more detailed outputs. This improvement stems from our advanced tokenization scheme, which not only achieves a higher compression ratio but also better preserves locality.

Backbone and Training Cost. Regarding the backbone size and training step of the generator, our method employs a 24-layer transformer with 1024 hidden dimensions, which is **more lightweight** than *EdgeRunner* that uses a 24-layer transformer with 1536 dimensions. In terms of training cost, our method was trained on 8×L40 (48GB) GPUs for 2 weeks, whereas *EdgeRunner* used 64×A100 (80GB) GPUs for 1 week. Thus, our total GPU hours are **less than 1/4** of those used by *EdgeRunner*. Besides, we employs a training-free Michelangelo conditioner for image input, whereas *EdgeRunner* used 16×A800 (40GB) GPUs for 1 week on training image conditioner.

Maximum Face Ability. Due to our GPU resource limitations, we train *Nautilus* with a maximum of **12,000** tokens, whereas *EdgeRunner*, utilizing 64×A100 (80GB) GPUs, trains with about **17,200** tokens, which can be obtained from their maximum face number of **4,000** and compression ratio of 0.474. Theoretically, given the significant improvement in compression ratio, our tokenization supports the training and generation of **7,000** faces assets with the same 17,200 tokens on comparable GPU resources – **nearly twice** that of *EdgeRunner*. In the future, we are planning to extend our training data to cleaned higher face number mesh assets and further scale up our framework.

In summary, based on novel methodologies, *Nautilus* focuses on improving local structure modeling and representation compactness using mesh locality, achieving significantly higher generation quality with more faces and far less training cost comparing with *EdgeRunner*.

3.2. Impact of Sequence Compression

In this section, we provide a comprehensive discussion on how compression ratio directly impacts generation quality during both training and inference stages.

Effect in Training. During training, artist-created mesh assets in the dataset are converted into sequences through tokenization. Due to GPU memory limits, the maximum se-

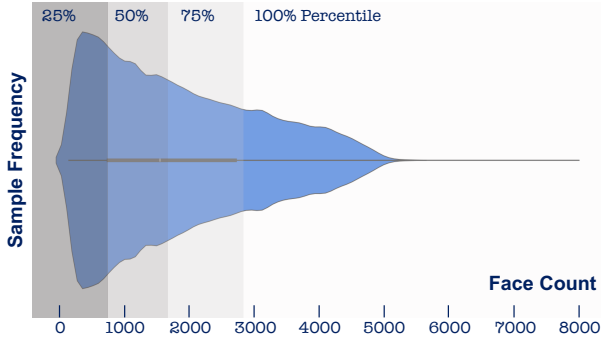


Figure 5. Statistics on the samples’ face counts in our training set, including 311K high-quality mesh assets with up to 8,000 faces.

quence length of a 500M parameter transformer decoder in autoregressive mesh generation is typically around 20,000. For efficient training, sequence lengths are generally constrained around 12,000. Within this limit, tokenization algorithms with higher compression ratios enable the inclusion of more complex, high-face-count mesh assets. Learning from such assets enables the model to handle complex geometries under challenging conditions, mitigating failures such as missing components, large surface holes, and other structural defects. With Nautilus achieving an unprecedented compression ratio of 0.275, it allows the inclusion of mesh assets containing over 5,000 faces in 12,000 tokens. This compression facilitates the model’s ability to learn from high-quality assets, enabling the generation of unprecedented topological complexity.

Effect in Inference. The inherent gap between the prediction of the next token during training and autoregressive inference during inference amplifies the cumulative errors as the length of the sequence increases. Our empirical analysis reveals that, even with well-optimized models, the validation loss at the 15,000-th position is approximately three times higher than at the 5,000-th position. Therefore, tokenization algorithms with lower compression ratios further aggravate this issue by requiring longer sequences to represent the same number of faces, increasing the likelihood of defects. This observation aligns with our ablation study, where the groups using only Nautilus Shell Representation (*Rep. Only*) demonstrates inferior generation quality compared to the *Full Tokenization*, which is attributed to its inferior compression ability.

4. Training Data Statistics

We conducted a statistical analysis of the face counts in our training dataset, with the results shown in Figure 5. Our training dataset consists of 311K high-quality mesh assets that can be tokenized into 12,000-length sequences using our Nautilus-style Tokenization Algorithm. As illustrated in the figure, about 50% of the samples exceed the maximum

face count of 1,600 supported by previous methods and even include a small subset of extremely complex samples with 5,000 to 8,000 faces. This demonstrates that our tokenization method, while typically supporting meshes with up to 5,000 faces, is capable of achieving an even higher level of compression in certain cases, enabling the tokenization of meshes with up to 8,000 faces. Leveraging this extensive and carefully curated training set, which covers a wide range of face counts, Nautilus not only ensures basic geometric modeling but also learns the ability to effectively handle complex topologies.

5. User Study Details

In this section, we provide additional details of our User Study. As shown in Figure 6, each question in our study includes the rendering visualizations of the input point cloud and the output meshes generated by the compared methods. To ensure unbiased feedback, all options are presented anonymously, and their order is shuffled for every question. Specifically, users are asked to evaluate each result by indicating whether they are satisfied with it. For each method, we collect the responses and calculate the average satisfaction rates across all questions. In total, our User Study consists of 20 questions, featuring 20 input point clouds and 60 corresponding mesh assets generated from the three comparing methods. In total, the survey received 38 responses from participants with diverse backgrounds. This carefully designed questionnaire, combined with a broad participant base, ensures the objectivity and reliability of the results.

6. Limitations

Through the locality-aware mesh tokenization and local-dependency enhancement mechanisms, our method effectively improves the manifoldness of local mesh structures and topological quality. However, it does not strictly guarantee the generation of fully manifold meshes. Specifically, while our approach significantly reduces surface holes, the improvement in reducing intersecting faces is relatively less pronounced. In future work, we plan to further address this issue by introducing more explicit structural constraints.

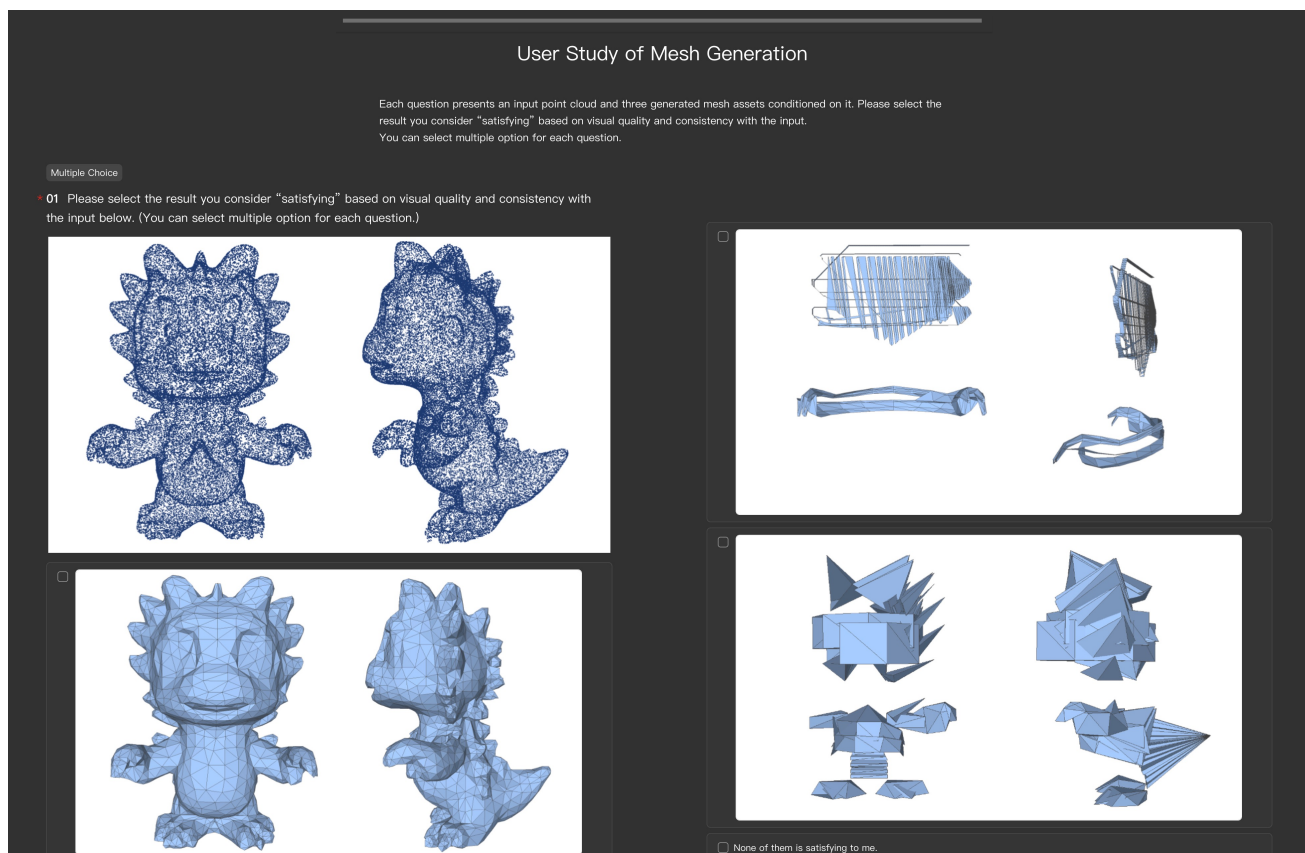


Figure 6. The Interface of Our User Study. Given the rendered images of the input condition and the generated meshes from each compared method, users are asked to indicate whether they are satisfied with each result. We calculate the average satisfaction rate for each method.