Supplementary Material for MeshPad: Interactive Sketch-Conditioned Artist-Reminiscent Mesh Generation and Editing

Haoxuan Li[1] Ziya Erkoc[1] Lei Li[1] Daniele Sirigatti[2] Vladislav Rosov[2]
Angela Dai[1] Matthias Nießner[1]
[1]Technical University of Munich [2]AUDI AG

In this material, we first introduce the details of sequentialized mesh representation in Sec. 6, followed by details of our self-supervised data processing in Sec. 7. We introduce the implementation of our user interface in Sec. 8. Then, we showcase four additional qualitative experiments in Sec. 9. Next, we provide more visual evaluation results in Sec. 10 and topological statistics of generated meshes in Sec. 11, followed by more details and results from our perceptual study in Sec. 12.

6. Mesh Token Sequence Details

As stated in Sec. 3.2, The sequential representation of a mesh consists of vertex coordinate tokens and control tokens where the <split> token separates the full sequence representing the whole mesh into subsequences corresponding to a "triangle chain" of adjacent triangles. Each subsequence has $3n$ vertex coordinate tokens where n is the number of vertices contained in the subsequence. Given a subsequence, we first decode a sequence of vertices $\{\boldsymbol{v}_k\}_{k=0}^{n}$: the x, y, and z position of $\boldsymbol{v}_k$ is obtained from the coordinate tokens at positions $3k$, $3k+1$, and $3k+2$ in the subsequence, respectively. We then get the triangle chain by forming triangles of three adjacent vertices in the vertex sequence:

$$\{\mathcal{F}\} = \{\{\boldsymbol{v}_k, \boldsymbol{v}_{k-1}, \boldsymbol{v}_{k-2}\} | \forall k : k >= 2\}. \tag{7}$$

Merging all triangle chains obtained from all subsequences results in the full mesh. When converting a mesh into sequence, we first sort every triangle in the order of $z-y-x$, then perform a depth-first traversal from the first triangle to find adjacent triangles as the first subsequence. We then subtract the triangles already traversed and start from a new triangle to obtain the next subsequence. We iterate this process until all triangles are traversed.

Our addition network autoregressively generates one new vertex (three coordinate tokens) or the <split> token for each step. If the new vertex's index in the current vertex sequence (measured by its distance from the nearest previous <split> token) is ≥ 2, it forms a triangle with the two preceding vertices.

7. Data Generation Details

In this section, we provide additional details for our data generation process (Sec. 3.4).

Random sampling of volumes. To sample volumes that satisfy $\mathcal{L}_k \subsetneq \mathcal{L}$, we first randomly define two volumes $\mathcal{L}_{\{a,b\}}$ within the bounding volume (assuming a unit cube) of the complete mesh $\mathcal{M}_c$. For each volume $\tilde{\mathcal{L}}$ of $\mathcal{L}_{\{a,b\}}$, we sample the volume as following: first, randomly choose an axis i from $\{1,2,3\}$ indicating the x, y, or z axis, then randomly select one region $\mathcal{R}$ from the four candidates: $[-\infty, a]$, $[a, +\infty]$, $[b-c, b+c]$, $[-\infty, b-c] \cup [b+c, +\infty]$, where $a \in [0.2, 0.8]$, $b \in [0.4, 0.6]$ and $c \in [0.1, 0.4]$ are uniformly sampled within their defined range. The sampled volume is then defined as:

$$\tilde{\mathcal{L}} = \{\boldsymbol{p} \in \mathbb{R}^3 | p_i \in \mathcal{R}\}. \tag{8}$$

Then we define:

$$\mathcal{L}_k = \mathcal{L}_a; \quad \mathcal{L} = \mathcal{L}_a \cup \mathcal{L}_b, \tag{9}$$

which ensures that $\mathcal{L}_k \subseteq \mathcal{L}$.

After sampling both volumes, it can happen that $\mathcal{L}_b \subseteq \mathcal{L}_a$, and as a result, $\mathcal{L} = \mathcal{L}_k$ and thus the editing part $\mathcal{M}_r$ is empty. To avoid this, we manually set $\mathcal{L}_b = \mathbb{R}^3$ in this case so that it contains the entire mesh (that is, a mesh completion task for addition).

The random volume sampling, while effective, can produce data samples that are not well-suited for training. For instance, the target mesh $\mathcal{M}$ for addition could be close to the complete mesh $\mathcal{M}_c$ with just a few triangles missing, and that the sketch is also close to the full sketch of $\mathcal{M}_c$. As a consequence, the model would learn to generate a mesh with missing triangles, even when the user provides a complete sketch. To address this issue, we additionally check the coverage of the visibility mask of $\mathcal{M}$ to the complete mesh's ($\mathcal{M}_c$) and set $\mathcal{L}_b = \mathbb{R}^3$ if the coverage is greater

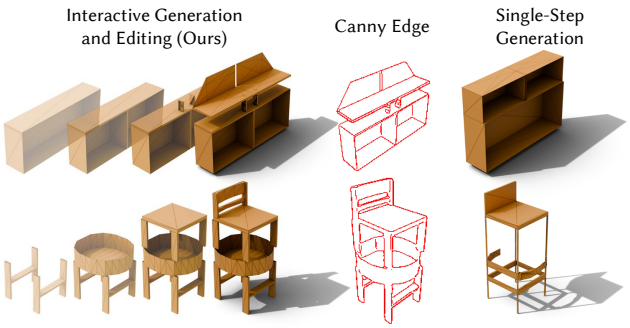

Figure 8. Comparisons between our interactive generation and editing vs. single-step generation. **Left**: meshes interactively created using our method, which effectively uses multiple localized generation steps to achieve high complexity. **Middle**: synthetic sketches generated from the final meshes via Canny Edge detection. **Right**: meshes generated by applying one pass of the addition network (without iterative editing) on the middle sketches. Our interactive editing pipeline enables creating intricate shapes that can be challenging to generate in one step.

than 95%.

Sketch generation. Automatic sketch generation is done by applying Canny edge detection on the rendered depth and normal images of $\mathcal{M}$. We then merge the two edge detection results to obtain the corresponding synthetic sketch used for training. To get mutually exclusive sets of line strokes $\mathcal{I}_{\{r,k\}}$, which correspond to different parts of the mesh, we further render a visibility mask of $\mathcal{M}_r$ in the same camera view and collect all line strokes within the mask as $\mathcal{I}_r$. $\mathcal{I}_k$ is then defined as the line strokes not in $\mathcal{I}_r$. In practice, the line strokes from the Canny edge detector could lie outside the mask by 1 pixel. Therefore, we dilate the visibility mask by 1 pixel before collecting $\mathcal{I}_r$. During training, we randomly sample camera views from the upper unit viewing hemisphere, with azimuth angles in $[-90°, 90°]$ and elevation angles in $[0°, 60°]$.

Mesh and sketch augmentations. During training, we apply random axis independent scaling to the mesh, with a scaling factor uniformly sampled in $[0.9, 1.1]$ on each axis. For the sketch, we apply a random affine transformation and a random elastic transformation to fill the domain gap between generated sketches and human drawings.

8. User Interface Implementation

We develop a Gradio [1] user interface to demonstrate our method in an interactive editing environment, accessible from browsers on all types of devices (PC, iPad, etc.). As visualized in Fig. 1, the interface contains a sketchpad for users to draw (addition) and erase (deletion) line strokes and a three.js [3] viewer displaying the mesh in real-time. Users can submit their sketches to the addition or deletion job and can always re-sketch and re-submit the job when the outcome should be iterated on. When an edit is accepted, the interface refreshes the sketch in the sketchpad to match the current mesh for further edit, which corresponds to the process depicted in Fig. 2. We refer to our supplementary video for live demos.

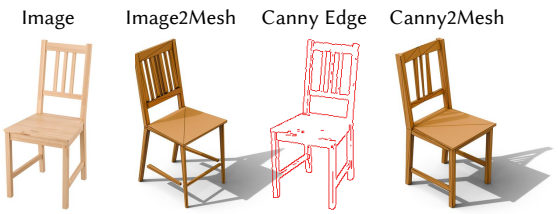

Figure 9. Image to mesh results. Benefiting from the large image foundation model RADIO, our method generalizes to image-conditioning without additional training. For better image matching and mesh quality, users can pre-process the image to get a canny edge for conditioning.

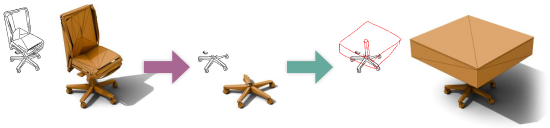

Figure 10. Our method allows us to edit directly on a given mesh (created manually or with other methods). We first load a mesh of an office chair, and automatically generate the corresponding rendered sketch. We can then delete the upper part and add a square tabletop to build a movable table.

9. Additional Qualitative Experiments

Generating complex shapes through interactive editing. Our interactive mesh editing enables not only an iterative creation process, but its localized editing focus enables construction of complex 3D shapes by decomposing their generation into a sequence of simpler part generations. As a result, artists can use our method to create a larger variety of complex shapes than the single-step generation, as demonstrated in Fig. 8. We refer to our supplementary video for more interactive modelings in action.

Image conditioned mesh generation. While our method is not trained on images, it generalizes well to image-conditioning, taking advantage of the large image foundation model RADIO [6]. It requires no re-training nor any adaptations to the pipeline to use our model for image-conditioned mesh generation. In Fig. 9, we show an example of image-conditioned generation with our method. It is worth mentioning that the mesh quality and image-to-mesh correspondence are lower for image-conditioned generation compared to sketch-conditioned generation. A simple pre-processing to get the canny edge for conditioning would boost the performance of our method.

Direct mesh editing. Another benefit of using mesh as an underlying representation is that we could edit directly on a

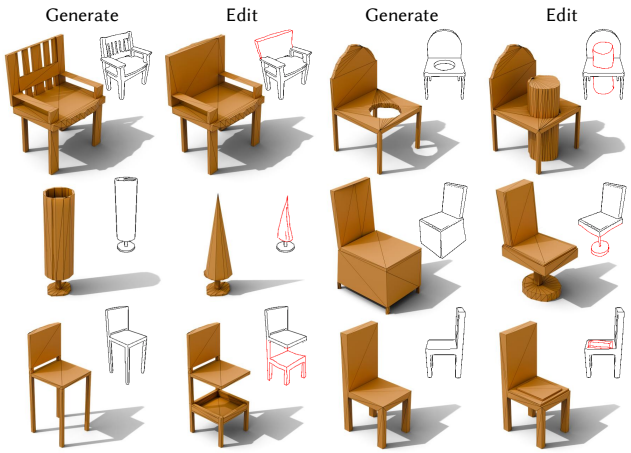

Figure 11. Visual results of our method on our hand-drawn dataset. Our method successfully performs a variety of edits, from local changes to edits that affect the majority of the shape.

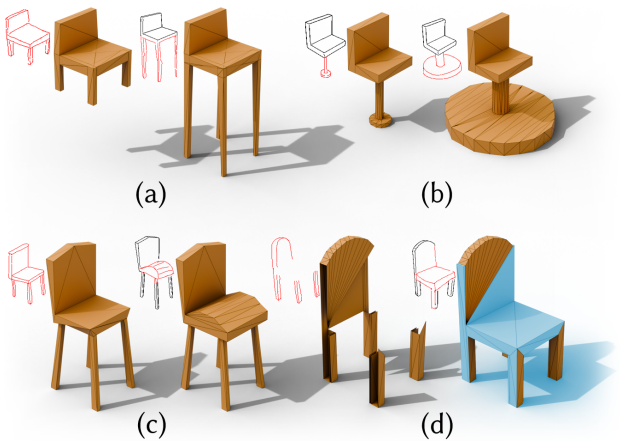

Figure 12. Qualitative examples showing the editing capability of our model. **(a, b)**: sensitivity of edited shapes to the sketch input. **(c)**: editing the middle part of the shape. **(d)**: the added part (cyan) connects to the existing part.

given mesh. Other methods, such as SENS [2], that perform editing on latent spaces struggle at editing a given mesh as it is tricky to convert the mesh to the model input. The common way is to encode the mesh to the latent space and feed it to the model. Consequently, there is no guarantee that the unedited part will remain unchanged due to the error of encoding and decoding. Our method naturally addresses this issue using explicit mesh representation as model input. This allows us to edit a given mesh without worrying about changing the unedited parts. In Fig. 10, we show an example of editing a given mesh. After loading the mesh and generating the sketch, we could perform addition and deletion as if the mesh is generated by the model.

Mesh-to-Sketch Alignment. Our method achieves superior performance on sketch alignment. Fig. 12 shows that our method can **(a, b)**: generate shapes aligning with the length and size of the sketch input; **(c)**: edit the middle part of the mesh; **(d)**: complete broken shapes via addition. While our method achieves better sketch alignment metrics than baseline methods, it does not guarantee exact alignment between generated meshes and sketches, since our model aims at high-fidelity shape generation from *free-hand* sketches.

10. Additional Visual Results

We provide more mesh generation and editing results of our method in Fig. 11, as a complement to Fig. 7. Note that our hand-drawn sketch editing dataset, while containing only 50 meshes, covers a large variety of editing tasks that are common in interactive mesh editing, from small local changes to global shape structure adjustments. Thus, it serves as a strong and challenging benchmark for sketch-based mesh editing. While trained only on self-supervised data (Sec. 3.4) without any data from real artist drawings, our method generalizes to these tasks proposed with human drawing, showing consistent performance across diverse editing tasks. This highlights the effectiveness of our self-supervised data processing approach.

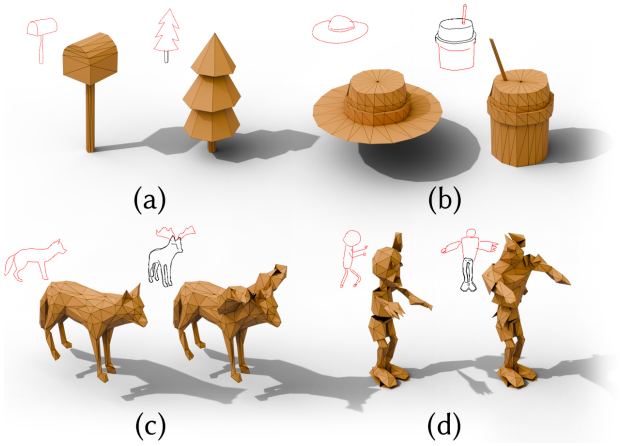

Figure 13. Qualitative results on Objaverse [5] **(a, b)** and DeformingThings4D [7] **(c, d)**. We fine-tune our networks on the two datasets and show their scalability to multiple categories.

In Fig. 13, we show the mesh editing results of our networks fine-tuned on Objaverse [5] and DeformingThings4D [7]. For Objaverse fine-tuning, we filter the meshes with fewer than 768 triangles to obtain around $9,000$ meshes. For DeformingThings4D, we sample around $1,000$ meshes from all animations and decimate the mesh to fit the limit of our setup. The result shows the scalability of our method to a large variety of meshes.

Method	#Faces	#Components	Non-manifold Edges%	#Self Intersections
LAS	33,573	1.0	0	0
LAS-MA	1,114	51.9	2.05	1,738
SENS	33,859	2.5	0	0
SENS-MA	1,513	73.4	2.22	1,787
Ours	231	12.3	2.01	259
Training Data	285	12.0	2.13	3,285

Table 5. Topological statistics of edited meshes vs. train data. Our method generates meshes that align best with the training data with significantly reduced self-intersections. Note that LAS and SENS generate over-tessellated meshes that deviate from the training data.

11. Topology of Generated Meshes

As shown in Tab. 5, our method generates meshes with similar topological statistics to the training data, while meshes of baseline methods deviate from it. LAS and SENS meshes are over-tessellated marching-cube meshes and are not ideal, even with no non-manifold edges and no self-intersections. We also surprisingly find that the number of self-intersections of our method is significantly less than the training data. We believe the reason is that the model learns to use the simplest way to represent a surface, while in ShapeNet, there are often redundant triangles.

12. Perceptual Study Details

In our perceptual study, we ask each participant to do 10 single-method evaluation tasks and 10 dual-method comparison (ours vs. baseline) tasks. Fig. 14 shows the UI for our perceptual study. The samples used for the questions are randomly chosen from all mesh editing results of all methods and are different for each participant. Our participants have a large variety: from professional artists to hobbyists, students, and individuals with no prior artistic experience.

We show in Tab. 6 again the unary user rating with the standard deviation. With the smallest deviation observed in all subjects, we show a high agreement among participants regarding the performance of our method.

	Generation Rating		Editing Rating		
Method	**GQ↑**	**GM↑**	**EQ↑**	**EM↑**	**EC↑**
LAS	3.2(1.1)	3.0(1.2)	2.7(1.1)	2.2(1.1)	2.5(1.3)
LAS-MA	2.8(1.2)	2.6(1.2)	2.4(1.3)	2.0(1.1)	2.0(1.1)
SENS	3.4(1.2)	3.5(1.1)	2.9(1.2)	1.8(1.0)	3.7(1.6)
SENS-MA	2.7(1.4)	2.8(1.4)	2.1(1.4)	1.6(1.1)	2.5(1.3)
Ours	**4.3**(0.9)	**4.3**(0.8)	**4.3**(0.8)	**4.2**(0.9)	**4.3**(1.0)

Table 6. User ratings (ranged 1-5) on generation and editing results of our hand-drawn sketch evaluation set. We benchmark against LAS [8] and SENS [2], with outputs further refined using MeshAnythingV2 (MA) [4] as a post-processing baseline. For mesh generation, participants evaluate mesh quality (GQ) and sketch matching (GM). For mesh editing, participants rate edited mesh quality (EQ), edited sketch matching (EM), and edited mesh consistency (EC) of the unedited part.

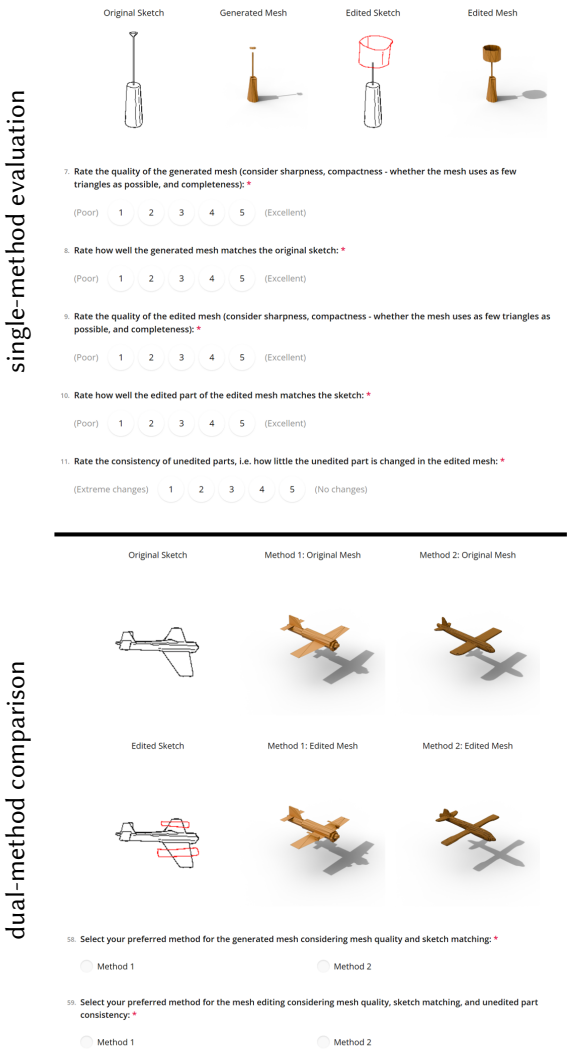

Figure 14. Screenshot of our perceptual study. **Top**: single-method evaluation. **Bottom**: dual-method comparison. Each participant is required to complete 10 single-method evaluations and 10 dual-method comparison tasks.

References

[1] Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*, 2019. 2

[2] Alexandre Binninger, Amir Hertz, Olga Sorkine-Hornung, Daniel Cohen-Or, and Raja Giryes. SENS: Part-aware sketch-based implicit neural shape modeling. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2024)*, 43(2), 2024. 3, 4

[3] Ricardo Cabello and Contributors. Three.js: Javascript 3d library. https://threejs.org/, 2010. Accessed: 2025-01-22. 2

[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, 2024. 4

[5] 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. *arXiv preprint arXiv:2212.08051*, 2022. 3

[6] Mike Ranzinger, Greg Heinrich, Jan Kautz, and Pavlo Molchanov. Am-radio: Agglomerative vision foundation model reduce all domains into one. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12490–12500, 2024. 2

[7] Takafumi Taketomi Bo Zheng Yang Li, Hikari Takehara and Matthias Nießner. 4dcomplete: Non-rigid motion estimation beyond the observable surface. *IEEE International Conference on Computer Vision (ICCV)*, 2021. 3

[8] Xin-Yang Zheng, Hao Pan, Peng-Shuai Wang, Xin Tong, Yang Liu, and Heung-Yeung Shum. Locally attentional sdf diffusion for controllable 3d shape generation. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4), 2023. 4