MeshFIM: Local Low-Poly Mesh Editing via Fill-in-the-Middle Autoregressive Generation

Dingdong Yang[1,2] **Jian Liu**[1,3] **Biwen Lei**[1] **Haohan Weng**[1] **Zhuo Chen**[1]
Song Guo[3] **Ali Mahdavi-Amiri**[2] **Hao Zhang**[2,†] **Chunchao Guo**[1]

[1]Tencent Hunyuan [2]Simon Fraser University
[3]Hong Kong University of Science and Technology
[†]Corresponding author.

Abstract

Autoregressive (AR) models generate high-quality low-poly meshes from point clouds, but typically regenerate the entire mesh when only a local region needs improvement. We introduce **MeshFIM**, a Fill-in-the-Middle framework for reference-guided local low-poly mesh editing. Given a point cloud representing the desired surface and the surrounding low-poly context, MeshFIM regenerates a selected region while retaining the mesh outside it. The original target faces are removed before generation; the desired geometry must be supplied by the reference, including inside the edited region. Five complementary designs address boundary attachment, context ordering, and generation overflow: boundary vertex markers, context positional embeddings, expanded context width, context augmentation, and a low-poly geometry encoder with gated reference-feature attenuation. Experiments on randomly sampled regions of AR-generated meshes show strong boundary matching and geometric improvement relative to adapted refinement, repair, and whole-mesh regeneration baselines. Component ablations support these design choices, and we demonstrate brush-based region selection and automatic defect repair. MeshFIM targets local tessellation and connectivity conditioned on known geometry; completing unknown geometry without a reference remains outside its scope.

1 Introduction

Low-poly assets often require local adjustments: a generated mesh may contain malformed connectivity, or an artist may need to refine a joint or improve edge flow. In these cases, regenerating an entire asset wastes work and can destroy satisfactory tessellation elsewhere. Local editing should replace a selected region while preserving the surrounding mesh. In the low-poly setting, connectivity and face layout matter alongside surface accuracy because they influence subsequent artistic editing and animation.

Task setting: known geometry, local topology generation. We assume access to a dense reference surface, supplied as a point cloud, that specifies the desired geometry throughout the edited region. The input low-poly mesh provides the surrounding tessellation and connectivity. MeshFIM removes the selected target faces before generation and predicts a replacement patch from the reference and the remaining context; the discarded patch is not a conditioning input. This setting arises when converting dense scanned or generated assets to low-poly meshes and locally improving their tessellation. It differs from reference-free hole filling, where the missing surface itself must be inferred. MeshFIM does not reconstruct unseen geometry from a damaged mesh alone.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

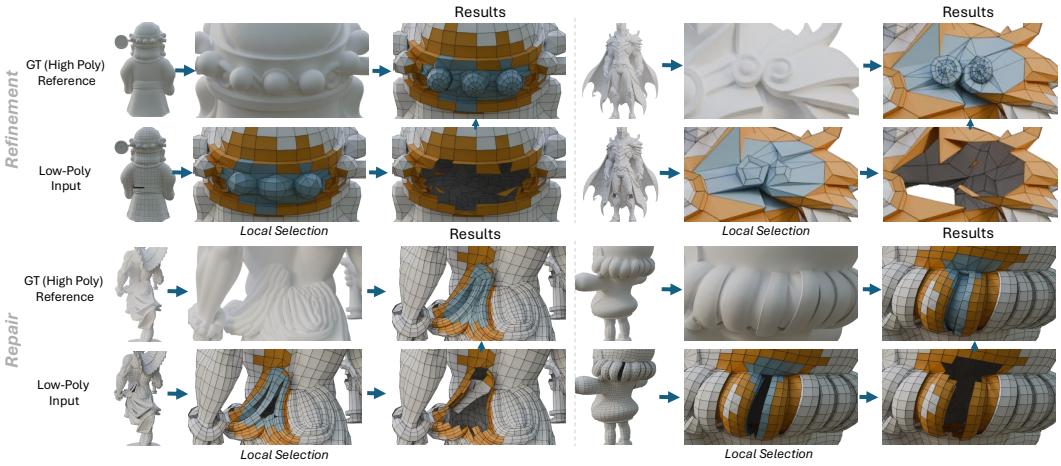


Figure 1: **MeshFIM** enables accurate local low-poly mesh editing. The selected low-poly target faces are removed before generation. A target local region (blue) is selected via flexible interactive tools (Section 5.1) or automatic defect detection algorithms (Section 5.2). Conditioned on a required dense reference surface and context mesh (orange), MeshFIM regenerates the target region for refinement (top) or repair (bottom) while encouraging boundary connectivity and coherent edge flow with surrounding existing mesh.

Classical subdivision and hole-filling methods focus on geometric criteria such as smoothness and surface approximation. Learning from artist-made meshes offers a complementary way to generate local face layouts that respect surrounding tessellation and connectivity.

Recently proposed autoregressive (AR) mesh generators can produce artist-grade low-poly meshes directly from geometric inputs such as point clouds. Models such as MeshGPT Siddiqui et al. [2024], MeshAnything Chen et al. [2025a,b], QuadGPT Liu et al. [2026], among others Hao et al. [2024], Chen et al. [2024], Zhao et al. [2025], Tang et al. [2025], Lei et al. [2025], Weng et al. [2025], Lionar et al. [2025], generate meshes token-by-token, attaining high-quality mesh topology. These generators primarily produce complete meshes, so a local change generally requires full regeneration.

We introduce MeshFIM, a local mesh editor based on *Fill-in-the-Middle* (FIM) generation from code completion [Fried et al., 2022, Guo et al., 2024]. MeshFIM autoregressively generates replacement faces conditioned on the surrounding mesh as topological context and on the desired reference surface. Unlike a one-dimensional code span, a connected mesh patch generally scatters across a globally serialized face sequence. Local infilling must also attach to exposed boundary vertices, maintain surrounding edge flow, and remain robust to imperfect inference-time context.

MeshFIM addresses these requirements through five complementary designs:

- Learned *boundary vertex markers* identify stitching anchors and encourage exact coordinate reuse.
- *Context positional embeddings* retain order within the serialized prompt to help preserve surrounding topological patterns.
- An *expanded context window* supplies broader geometric evidence to reduce generation beyond the intended region.
- *Context augmentation* exposes training to imperfect boundaries and narrows the train-inference gap.
- A *low-poly geometry encoder* uses position-aligned reference and existing-mesh features to attenuate conditioning over already-covered geometry and focus generation on the missing region.

We evaluate these choices through baseline comparisons (Section 4.2) and cumulative ablations (Section 4.3), and demonstrate brush-based selection and automatic defect repair (Section 5).

2 Related Work

Autoregressive Mesh Generation. PolyGen Nash et al. [2020] first demonstrated vertex-then-face autoregressive mesh generation. MeshGPT Siddiqui et al. [2024] introduced a VQ-VAE tokenizer followed by a GPT-style decoder. MeshAnything Chen et al. [2025a] and MeshAnythingV2 Chen et al. [2025b] scaled the approach to artist-quality meshes. Many follow-up methods further improved tokenization and training strategies Chen et al. [2024], Tang et al. [2025], Weng et al. [2025], Lionar et al. [2025], Lei et al. [2025], Zhao et al. [2025]. More recently, QuadGPT Liu et al. [2026] and Meshtron Hao et al. [2024] extended the paradigm to quad-dominant meshes by directly generating mixed quad-triangle faces in a canonical lexicographic order. All these methods generate *complete* meshes, some in the low-poly setting, but none support local low-poly mesh editing.

Local Subdivision. Local subdivision [Catmull and Clark, 1998, Zorin et al., 1996, Loop, 1987, Kobbelt, 2000, Liu et al., 2020, Zhu et al., 2025] has been a standard approach to increasing mesh density in a target region. However, these methods can only *add* (split) faces, and are not designed to rewire mesh topology. In contrast, MeshFIM regenerates topology from scratch in a local region.

Mesh Completion and Repair. Classical hole-filling and surface-reconstruction methods [Liepa, 2003, Tekumalla and Cohen, 2004, Ju, 2004, Kazhdan et al., 2006, Zhao et al., 2007, Kazhdan and Hoppe, 2013, Centin et al., 2015] infer missing surfaces from the observed mesh or samples. Optimization-based repair [Hattori et al., 2025] uses a per-shape prior to restore geometry. These methods address a different information setting from ours: MeshFIM assumes the desired surface is available as reference geometry and generates a compatible local low-poly tessellation. The comparisons therefore evaluate adapted methods under our reference-guided editing protocol, rather than establish superiority for reference-free completion. Learned boundary cues encourage attachment but do not guarantee it for every generated sample.

Fill-in-the-Middle for Sequences. FIM training was introduced for code language models [Bavarian et al., 2022, Lewis et al., 2020, Roziere et al., 2023, Guo et al., 2024], enabling infilling of contiguous spans. InCoder Fried et al. [2022] and subsequent work showed that FIM can be added to pretrained models with minimal degradation of left-to-right performance. However, since mesh resides in 3D space, its serialization token sequence depends on specific mesh topology, see Figure 2, namely where to put the missing tokens is unknown before eventual generation.

3 Method

3.1 Preliminaries: Autoregressive Mesh Generation

MeshFIM builds on recent autoregressive mesh generators introduced by Liu et al. [2026], Hao et al. [2024] that formulate mesh generation as next-token prediction. A mesh $\mathcal{M}$ with F faces is serialized into a token sequence so that each face f_j ($j = 1, \ldots, F$) is represented by n vertices ($n=4$ for quads, $n=3$ for triangles with padding to a uniform length), and each vertex is encoded as three quantized coordinate tokens, yielding a fixed-length block $\mathbf{b}_j \in \mathbb{Z}^{12}$ per face. Faces are sorted in a canonical bottom-to-top lexicographic order (by the YXZ coordinates of their lowest vertex), producing a deterministic sequence $\mathbf{x} = \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \cdots \oplus \mathbf{b}_F$, where $\oplus$ denotes concatenation. An Hourglass Transformer Nawrot et al. [2022] then models

$$p(\mathcal{M} \mid \mathbf{P}) = \prod_{i=1}^{|\mathbf{x}|} p(x_i \mid x_{<i},\ \mathbf{P}), \tag{1}$$

autoregressively predicting each coordinate token x_i conditioned on preceding tokens and a reference geometry (point-cloud $\mathbf{P}$). Following prior work, $\mathbf{P}$ is injected by a Perceiver-IO-style encoder with cross-attention queries and self-attention blocks Jaegle et al. [2021], Zhang et al. [2023], Zhao et al. [2023]. This formulation is effective for whole-mesh generation but not directly suited for local editing. A connected target region on the surface generally becomes scattered after serialization (Figure 2), and standard left-to-right decoding does not enforce exact reuse of boundary vertices when a regenerated patch is stitched back into the remaining mesh. These issues motivate MeshFIM.

Given this autoregressive representation, we cast local mesh editing as conditional infilling. The rest of this section describes the formulation (subsection 3.2) and the core design choices (subsection 3.3).

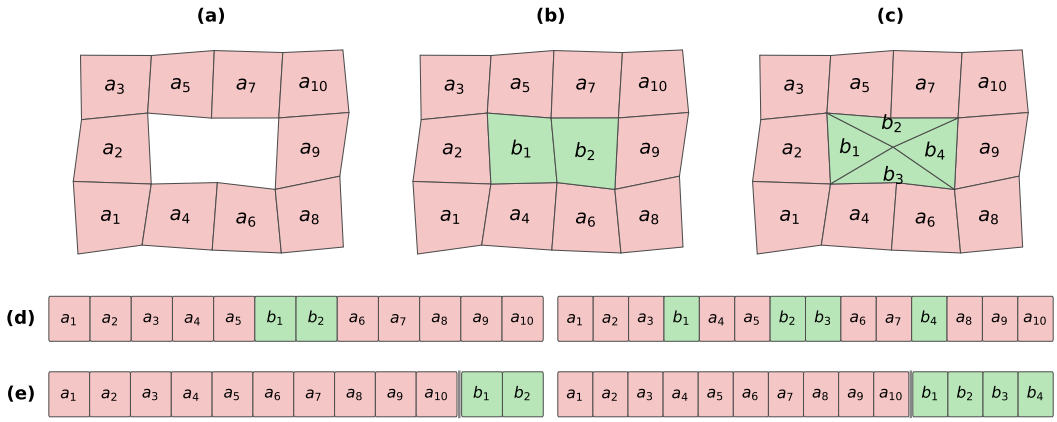


Figure 2: **Naive global serialization conflicts with local editing** (faces sorted in YXZ lexicographic order). (a) A fixed context $\mathcal{A}\backslash\mathcal{B}$. (b, c) Two valid tilings of the same hole with different face counts and shapes. (d) Under global serialization, the positions and number of target tokens (b_i, green) in the joint sequence are determined by the target geometry itself, which is unknown before generation: different tilings produce different sequences. (e) MeshFIM serializes $\mathcal{A}\backslash\mathcal{B}$ (pink) and $\mathcal{B}$ (green) independently, so that the sequence structure no longer depends on the unknown target geometry.

3.2 Problem Formulation

Let $\mathcal{M}$ denote an input low-poly mesh, and let $\mathcal{B} \subset \mathcal{M}$ denote a target region of interest (blue region in Figure 3). We remove the target faces and define the *context* $\mathcal{A}\backslash\mathcal{B}$ (orange region in Figure 3) as the set of faces within w breadth-first rings of $\mathcal{B}$. The objective is to learn a conditional distribution

$$p(\mathcal{B}' \mid \mathcal{A}\backslash\mathcal{B},\ \mathbf{P})\,, \tag{2}$$

where the point cloud $\mathbf{P}$ specifies the desired geometry, including the removed region. The target faces $\mathcal{B}$ are not supplied to either encoder or the decoder after removal; their appearance in comparisons serves only as a before-edit reference. The rest of the mesh outside $\mathcal{B}$ is retained when the patch is merged. Here $\mathcal{B}'$ denotes the generated faces that complete the region while stitching seamlessly to $\mathcal{A}\backslash\mathcal{B}$ and faithfully following the reference geometry $\mathbf{P}$. Concretely, MeshFIM serializes the context first and then decodes the replacement patch autoregressively:

$$[\mathrm{S}_{\mathrm{ctx}}] \oplus \mathrm{Tok}(\mathcal{A}\backslash\mathcal{B}) \oplus [\mathrm{E}_{\mathrm{ctx}}] \oplus \mathrm{Tok}(\mathcal{B}') \oplus [\mathrm{eos}], \tag{3}$$

where $[\mathrm{S}_{\mathrm{ctx}}]$ and $[\mathrm{E}_{\mathrm{ctx}}]$ are learned start and end sentinels, $\mathrm{Tok}(\cdot)$ denotes face tokenization, and $[\mathrm{eos}]$ ends the sequence. During training, the cross-entropy loss is applied only to $\mathrm{Tok}(\mathcal{B}')$ and $[\mathrm{eos}]$, so the context serves as a conditioning rather than a prediction target. Note that during training, $\mathcal{B}' = \mathcal{B}$.

3.3 Core Design Choices

MeshFIM adopts five complementary design choices. We first make boundary attachment constraints explicit by **boundary vertex markers**, then **expand context width** to reduce generation overflow, add **context positional embeddings** to recover context topology, and **context augmentation** during training to close the train-inference gap arising from noisy inference-time inputs, and finally introduce a **low-poly geometry encoder** to focus on generation over missing regions.

Boundary vertex markers. At the interface between $\mathcal{A}\backslash\mathcal{B}$ and $\mathcal{B}'$, even small coordinate mismatches create cracks or T-junctions. We therefore mark all context tokens that correspond to vertices on the exposed boundary. Let V_∂ denote this set. Their embeddings are modified as

$$\tilde{\mathbf{e}}_i^{\mathrm{ctx}} \;=\; \mathbf{e}_i^{\mathrm{ctx}} + \mathbf{m}_{\mathrm{bdry}}, \quad \text{if } t_i \in V_\partial, \tag{4}$$

where $\mathbf{m}_{\mathrm{bdry}} \in \mathbb{R}^d$ is a learned marker. It is an explicit cue informing the decoder which context vertices must be reused as stitching anchors, and shown to be essential for reliable boundary matching, although it is a learned cue rather than a hard decoding constraint; see our ablation studies in Table 2.

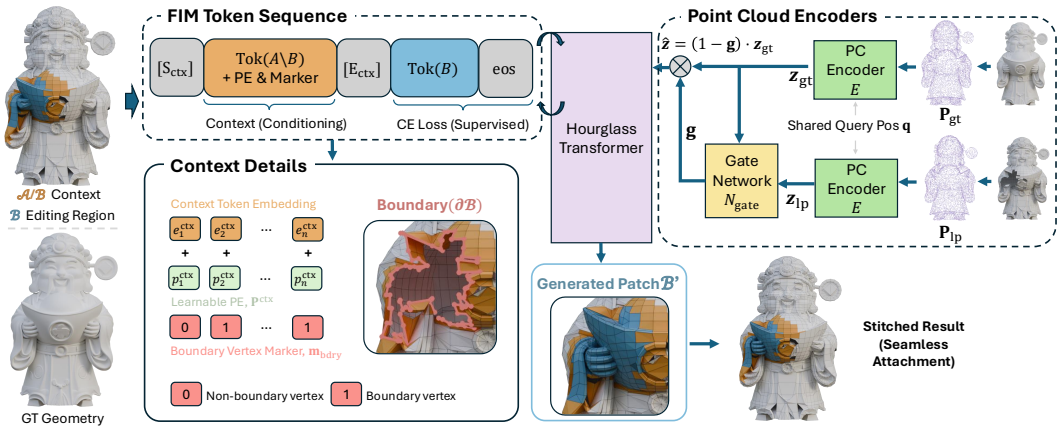


Figure 3: **MeshFIM pipeline overview.** Given an input mesh with a target region $\mathcal{B}$ and surrounding context $\mathcal{A}\backslash\mathcal{B}$, a shared point cloud encoder E with a common set of query positions $\mathbf{q}$ encodes two point clouds—the reference one $\mathbf{P}_{\text{gt}}$ and the existing low-poly mesh $\mathbf{P}_{\text{lp}}$ (with $\mathcal{B}$ removed)—into position-aligned latents $\mathbf{z}_{\text{gt}}$ and $\mathbf{z}_{\text{lp}}$. The mesh is serialized into a FIM token sequence: context tokens (with positional embeddings and boundary markers) followed by target tokens. An Hourglass Transformer autoregressively generates the new patch $\mathcal{B}'$, conditioned on the fused latents $\hat{\mathbf{z}}$ via cross-attention. The output is merged along the exposed boundary; attachment is learned rather than guaranteed.

Expanded context width. A narrow context provides only minimal information about local curvature and edge flow, which often causes *overflow generation*: newly generated faces spilling beyond the intended region. Using a wider context exposes more of the surrounding surface to ease geometry inference. In practice, three breadth-first rings provide a good trade-off between geometric coverage and prompt length. As shown in Figure 5 and Table 2, a wider context sharply reduces overflow.

Context positional embeddings. Beyond seamless stitching, a plausible edit should inherit local face density and edge flow from its neighborhood. If the context is given only as a flat token sequence, the model can treat it as an unordered bag of nearby vertices and lose the structure needed to preserve those patterns. We therefore add a learned positional encoding over the serialized context sequence:

$$\mathbf{e}_i^{\text{ctx}} \;=\; \text{Emb}(t_i^{\text{ctx}}) + \mathbf{p}_i^{\text{ctx}}, \quad i = 1, \ldots, |\,\text{Tok}(\mathcal{A}\backslash\mathcal{B})|, \tag{5}$$

where $\mathbf{p}_i^{\text{ctx}}$ is a learnable positional code. This restores order within the prompt and helps the model preserve the surrounding topological pattern. Qualitative evidence is shown in Figure 6, and the corresponding ablation study can be found in Table 2.

Low-poly geometry encoder. The point-cloud conditioner encodes the reference surface (what the mesh *should* look like) but has no knowledge of what the current low-poly mesh *already* covers. This information reaches the decoder only indirectly through the FIM context tokens. We therefore reuse the same Perceiver-IO encoder E to encode two point clouds, $\mathbf{P}_{\text{gt}}$ sampled from the reference surface and $\mathbf{P}_{\text{lp}}$ sampled from the existing low-poly mesh with $\mathcal{B}$ removed, under a *shared* set of cross-attention query positions $\mathbf{q} = \{\mathbf{q}_i\}_{i=1}^{M}$ drawn in Euclidean space:

$$\mathbf{z}_{\text{gt}} = E(\mathbf{P}_{\text{gt}}, \mathbf{q}), \qquad \mathbf{z}_{\text{lp}} = E(\mathbf{P}_{\text{lp}}, \mathbf{q}). \tag{6}$$

Sharing $\mathbf{q}$ aligns the two latent sequences position by position, so a per-slot gate is well-defined.

A lightweight gate network N_{gate} produces a vector $\mathbf{g} \in (0,1)^M$ from the two aligned latents, and the fused conditioning is an attenuation of $\mathbf{z}_{\text{gt}}$:

$$\mathbf{g} = \sigma\big(N_{\text{gate}}([\mathbf{z}_{\text{gt}};\, \mathbf{z}_{\text{lp}}])\big), \qquad \hat{\mathbf{z}} = (1 - \mathbf{g})\, \mathbf{z}_{\text{gt}}, \tag{7}$$

where multiplication broadcasts the scalar gate at each query position over feature channels. This is feature attenuation, not direct subtraction of point coordinates or a guarantee of geometric occupancy. Here $[\,\cdot\,;\,\cdot\,]$ denotes channel-wise concatenation of the two position-aligned latents at the shared query $\mathbf{q}$, σ is the sigmoid function, and N_{gate} is initialized so that training starts from $\hat{\mathbf{z}} \approx \mathbf{z}_{\text{gt}}$ (architecture and initialization in Appendix E). Intuitively, where the existing mesh already covers the reference

surface well, g_i is large and the reference conditioning at $\mathbf{q}_i$ is suppressed; where the mesh is missing (the hole), g_i stays small and the decoder sees the full reference latent, directing generation toward the missing region. This provides a continuous, spatially grounded signal that complements the discrete token context for overflow suppression and region focusing (Table 2, Figure 7).

Context augmentation. During training, contexts are extracted from ground-truth meshes, which are cleaner than those encountered at inference that may already contain quantization noise or artifacts from earlier generation steps. To narrow this gap, we perturb context vertex coordinates by random offsets drawn from $\mathcal{U}(-\delta, \delta)$. Shared boundary vertices $\partial\mathcal{B}$ are also perturbed. This exposes the model to imperfect boundaries during training to improve robustness, as confirmed by Table 2.

4 Experiments

4.1 Setup

Implementation Details. We use the AR mesh generator described in subsection 3.1: an Hourglass Transformer Nawrot et al. [2022] ($d = 1536$, depth 24, 16 heads) conditioned on point clouds encoded by an encoder Jaegle et al. [2021], Zhao et al. [2023]. We train the model on 64 computing units, each with about 90 GB memory. Context augmentation uses $\mathcal{U}(-0.002, 0.002)$ noise. Target regions $\mathcal{B}$ are sampled on the face adjacency graph with up to 1200 faces in a BFS-driven strategy. To expose the model to irregular editing targets that better resemble brush selections as well as automatically detected defects, 30% of training samples use a percolation-based region growth strategy: starting from a random seed face, the region expands stochastically along the frontier with acceptance probability $p \sim \mathcal{U}(0.55, 0.85)$ until the face budget is reached (see Appendix D for the full algorithm).

Baselines. To our knowledge, MeshFIM makes the first attempt at accurate local low-poly mesh editing/generation, hence there is no clearly suitable method to compare with. In making a best effort and considering the potential applications, we consider the following as comparable baselines:

1. Mesh Refinement: Modified Butterfly Zorin et al. [1996], Neural Subdivision Liu et al. [2020], and Neural Mesh Refinement Zhu et al. [2025].
2. Mesh Repair: Liepa Hole Filling Liepa [2003] and an optimization-based mesh repair method (SeMIGCN) Hattori et al. [2025].
3. Whole Mesh Generation + Stitch Back: we further compare with public AR mesh generation models (BPT Weng et al. [2025] and MeshAnythingV2 Chen et al. [2025b]) using a regenerate-whole-and-stitch-back scheme.

For detailed baseline implementations and more, please refer to Appendix B.

Datasets and test-region sampling. Our model is trained on public datasets (ShapeNetV2 Chang et al. [2015], Objaverse Deitke et al. [2023b], and ObjaverseXL Deitke et al. [2023a]) converted following Liu et al. [2026]. For testing, the initial low-poly meshes are produced by QuadGPT Liu et al. [2026], conditioned on point clouds sampled from the corresponding dense reference meshes; they are not ad-hoc coarse remeshes. We use 256 meshes and sample four connected target regions per mesh. Seed faces and face budgets are selected randomly, followed by BFS region growth, with percolation-based sampling also used for region diversity (Appendix D). Regions are not selected for poor initial quality. Each selected patch is removed before MeshFIM inference. Original patches shown in the figures establish the before-edit comparison and the reference values for improvement metrics; they are not model inputs. This random-region protocol is distinct from the defect-driven repair evaluation in Appendix C.

Metrics. Because local editing requires both seamless boundary attachment and geometric improvement, standard global mesh metrics are not sufficient on their own. We therefore use task-specific metrics that measure boundary correctness and local geometric quality. Specifically, we report **A-VMR** (Average Vertex Matching Ratio), the mean boundary-vertex match rate; **O-CDIR** (One-way CD Improvement Rate), the average relative reduction in patch-to-reference one-way Chamfer distance among perfect-match samples; **CD-PR** (Chamfer Distance Positive Rate), the fraction of test samples whose generated patch achieves a lower one-way Chamfer distance than the original target patch; and **#F-Inc** (Face Increase Ratio), the relative face-count change. Formal and complete definitions are provided in Appendix A.

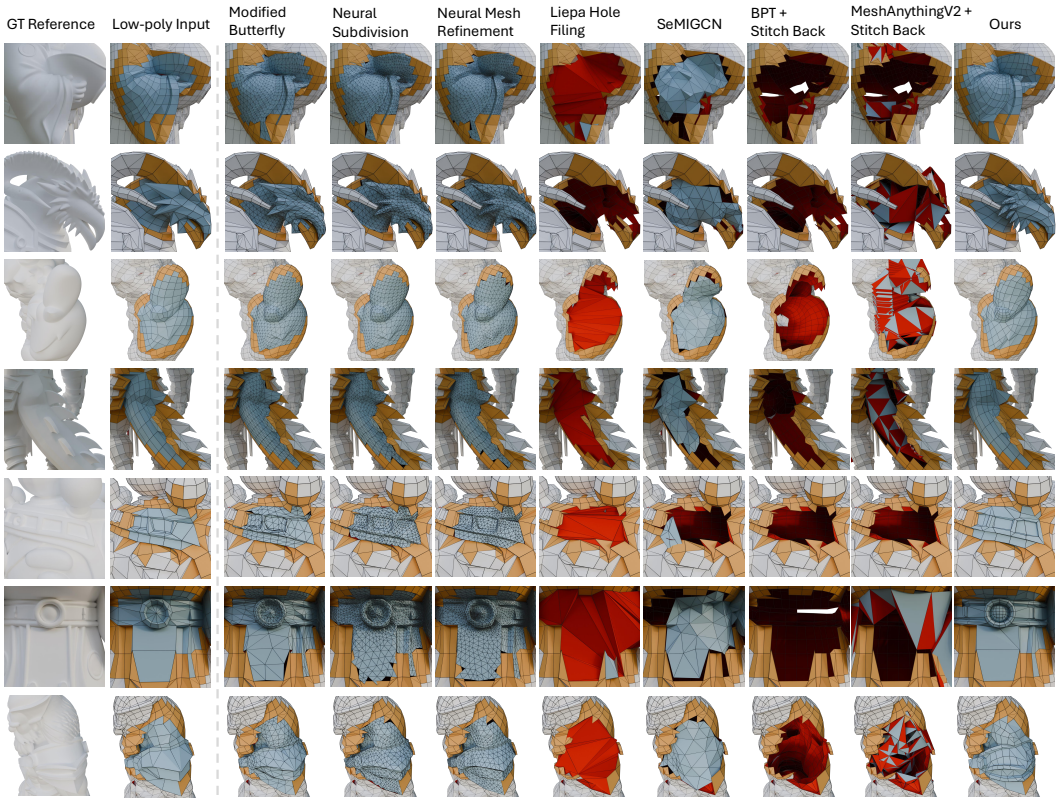


Figure 4: **Qualitative comparisons.** The original low-poly patches are from QuadGPT and are shown for before/after comparison; they are removed before MeshFIM inference. MeshFIM receives the surrounding mesh and dense reference geometry. The adapted baselines differ in access to that reference (Table 1). Dark red faces indicate back faces (i.e., the opposite side defined by the vertex winding).

4.2 Main Results

We show our quantitative results in Table 1 and qualitative results in Figure 4. MeshFIM achieves 99.75% A-VMR and 95.54% CD-PR under this protocol. The refinement and repair baselines do not use the dense reference, whereas BPT and MeshAnythingV2 do but regenerate the full mesh before cropping and stitching. These differences in task alignment and conditioning limit a direct ranking of the underlying methods. Whole-mesh generation can also fail to yield a usable cropped patch. Boundary-vertex coincidence measures only one aspect of attachment; it does not certify manifoldness or edge connectivity. Further baseline details and qualitative results are provided in Appendices B and G.

4.3 Ablation Studies

In addition to the metrics above, we report **OvR** (Overflow Rate), the fraction of test samples whose overflow ratio exceeds the threshold θ_{ovf}; and **A-Overflow** (Average Overflow), the mean overflow ratio (formally defined in Appendix A) (see Appendix A for formal definitions). All ablation rows are trained for 250k iterations under the same setting.

Table 2 reports the cumulative component ablation. Adding boundary vertex markers provides the largest adjacent-row gain in PMR (+17.41 pp), confirming that explicit attachment cues are critical for seamless stitching. Expanding the context from $w=1$ to $w=3$ sharply reduces both overflow metrics (OvR 14.84→10.16, A-Overflow 15.12→10.52), consistent with Figure 5: a wider context supplies sufficient geometric evidence to keep generation local. Context augmentation improves PMR by 8.60 pp, supporting the benefit of noisy training contexts. Context positional embeddings deliver additional topology quality, matching qualitative trends in Figure 6 where edge flow becomes aligned

Table 1: **Main results.** All numerical values are percentages (%). "GT Ref" indicates access to the dense reference geometry.

Method	A-VMR↑	O-CDIR↑	CD-PR↑	#F-Inc	Locality	GT Ref
Refinement						
Modified Butterfly	100.00	−2.00	25.00	300.00	✓	✗
Neural Subdivision	0.00	−0.002	38.28	852.25	✗	✗
Neural Mesh Refinement	33.68	0.007	60.94	845.21	✗	✗
Hole Filling / Repairing						
Liepa Hole Filling	67.64	-220.82	5.47	-73.97	✓	✗
SeMIGCN	0.00	-191.62	0.93	-41.29	✗	✗
Regen Whole + Stitch Back						
BPT + Stitch Back	2.65	-67.25	0.00	-84.98	✗	✓
MeshAnythingV2 + Stitch Back	26.90	-335.28	0.00	-20.45	✗	✓
MeshFIM (Ours)	99.75	8.25	95.54	55.83	✓	✓

Table 2: **Ablation study.** Cumulative component ablation; each row adds one component to the previous configuration. † Full model (all components enabled). †† Values in parentheses report the *no-edit reference value* for OvR and A-Overflow: the values obtained when the original target faces are used as the generated patch. The reference values are non-zero because OvR and A-Overflow metrics use a distance proximity threshold ϵ_{ovf} (Alg. 3). ‡ Naive token-axis concatenation replaces the gated fusion. ♢ Each encoder branch uses independently sampled query positions instead of the shared set $\mathbf{q}$. All values are percentages (%).

Configuration	PMR↑	A-VMR↑	O-CDIR↑	OvR↓	A-Overflow↓
Baseline	52.12	94.59	5.03	25.00	17.81
A +Boundary Vertex Markers	69.53	98.49	6.15	14.84	15.12
B +Expanded Context ($w=3$)	75.78	99.12	5.49	10.16	10.52
C +Context Augmentation	84.38	99.18	7.12	10.16	12.13
D +Context Positional Emb.	85.16	99.45	7.48	9.38	10.75
E +Low-Poly Geometry Enc. (Full†)	**88.28**	**99.75**	**8.25**	**6.25** (6.25)††	**4.40** (3.49)††
Low-poly geo. enc. ablation					
Full† w/ naive concat‡	85.16	99.51	7.50	8.59	9.27
Full† w/ no shared query♢	85.31	99.48	7.31	9.38	10.61

with surrounding patterns. Finally, the low-poly geometry encoder reduces overflow further (OvR 9.38→6.25, A-Overflow 10.75→4.40) by giving the decoder a continuous signal of where existing geometry already covers the reference surface. Our full model approaches the no-edit reference value (OvR 6.25 vs. 6.25, A-Overflow 4.40 vs. 3.49), showing that the metric also registers proximity present in the original patches. This reference is not a mathematical lower bound, and the remaining excess cannot be attributed solely to the threshold.

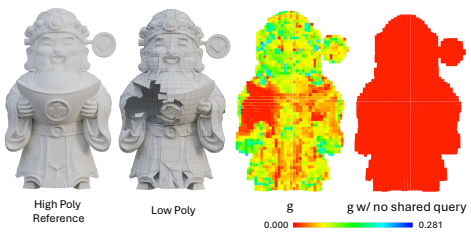


Figure 7: **Gate-value visualization** (shared *vs.* unshared query positions).

The (‡ naive concat) row replaces the gated design with a naive token-wise concatenation of the two latent sequences, $\hat{\mathbf{z}} = \mathbf{z}_{\text{gt}} \oplus \mathbf{z}_{\text{lp}}$ (where $\oplus$ denotes concatenation along the token axis, doubling the cross-attention context length). Unlike the channel-wise concatenation used inside the gate in Eq. 7, this naive variant performs no position-wise alignment and no explicit gating. The (♢ no shared query) row further demonstrates the importance of position-wise alignment: each encoder branch uses independently sampled query positions, so the i-th tokens $\mathbf{z}_{\text{gt},i}$ and $\mathbf{z}_{\text{lp},i}$

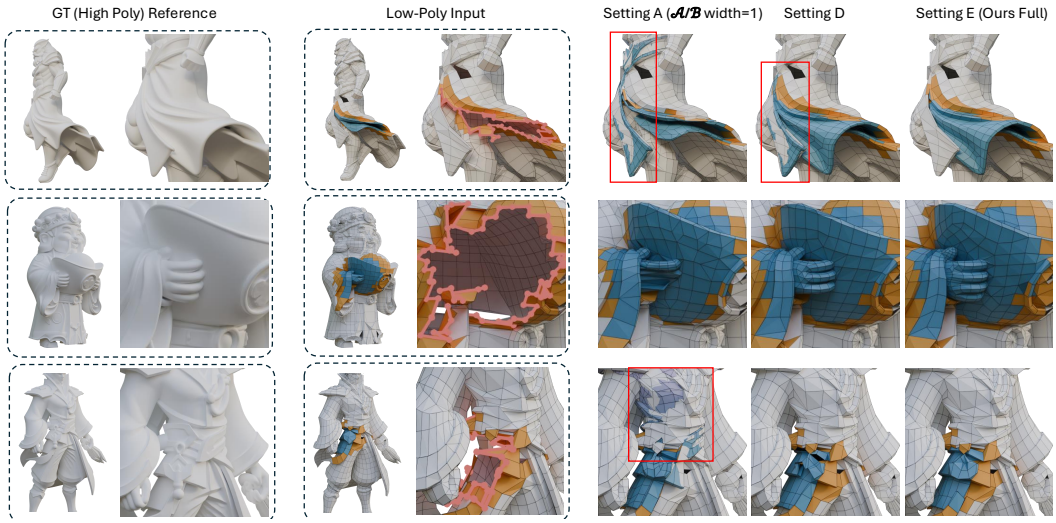


Figure 5: **Ablation qualitative results.** Settings A, D, and E correspond to configurations in Table 2. Red rectangles indicate overflow regions. Configurations A→D jointly add wider context, augmentation, and positional embeddings and reduce overflow, and the low-poly geometry encoder (D→E) further suppresses it while improving generation quality.

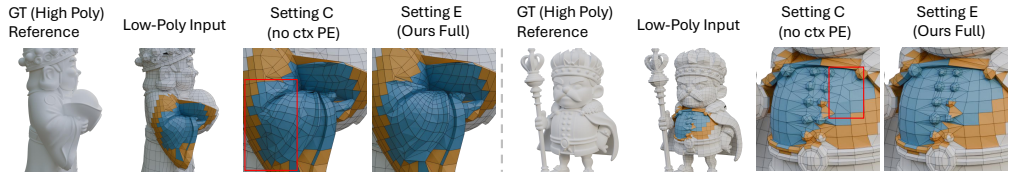


Figure 6: **Context positional embeddings on topology continuity.** Settings C and E correspond to configurations in Table 2. Red rectangles indicate the regions with poor edge flow. The C→E comparison adds both context positional embeddings and the low-poly geometry encoder; it illustrates their combined effect on edge flow rather than isolating either component.

correspond to different spatial locations. Without positional alignment, the gate receives uninformative input and collapses to near-zero values across all positions (Figure 7).

Efficiency. On a single NVIDIA H-series GPU with batch size 1 and BF16 inference, end-to-end inference for an approximately 200-face output patch takes about 1 s with less than 8 GB peak allocated GPU memory. This includes point-cloud encoding, patch decoding, and stitch-back. The inference implementation uses a context-pinned hierarchical rolling KV cache with windows of 12,288/3,072/1,024 tokens at the three Hourglass resolutions, CUDA Graphs, and face-aligned DFlash [Chen et al., 2026] speculative decoding with a 96-token block size. These measurements characterize this accelerated configuration and patch size; they do not establish a hardware-independent latency, a full scaling curve, or a measured speedup over the baselines.

5 Applications

We demonstrate two applications of MeshFIM: interactive brush-based editing (Section 5.1) and automatic defect repair (Section 5.2). Results are shown in the teaser (Figure 1).

5.1 Interactive Brush-Based Target Selection

We provide a web-based 3D interface where the user paints faces with a radius-adjustable brush to define $\mathcal{B}$. The brush selects faces under the cursor; each stroke is reduced to its largest connected component on the face adjacency graph via BFS. Subsequent strokes that are adjacent to the current selection are merged automatically, ensuring that $\mathcal{B}$ always remains a single connected region. See Appendix F for interface and implementation details.

5.2 Automatic Defect Detection

AR mesh generators can produce defective local regions such as missing face clusters or malformed topology. We build an automatic repair pipeline that can identify candidate defective regions (see Figure 8) via carefully designed graphics heuristics. The pipeline combines defect detection, reference-guided patch generation, quality gating, and merging. It requires a dense reference mesh for ray-cast confirmation, in addition to the reference point cloud used by MeshFIM. This whole pipeline can be run iteratively. Full implementation details and more results on this are in Appendix C.

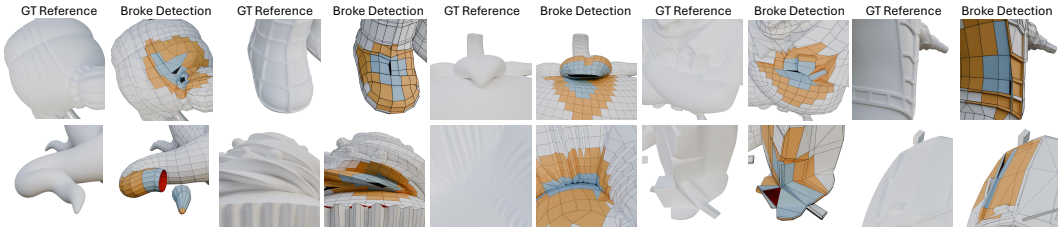


Figure 8: **Defect detection examples.** The heuristic detector (Algorithms 1 and 2) identifies candidate defects in the shown examples, ranging from missing-face clusters to small flipped faces and holes. Dense reference geometry is used to confirm candidates; these examples do not quantify detector recall.

6 Conclusion and limitations

We presented MeshFIM, a Fill-in-the-Middle framework for reference-guided local low-poly mesh editing. Boundary markers, context ordering and width, augmentation, and aligned geometry conditioning jointly improve attachment and reduce overflow. Random-region experiments and component ablations support this formulation, with brush selection and automatic defect repair illustrating its use.

Limitations and failure modes. MeshFIM requires reference geometry throughout the edited surface and cannot recover unknown geometry from a damaged mesh alone. Attachment is probabilistic: 99.75% A-VMR and 88.28% PMR mean that some patches still miss boundary vertices and may leave cracks. Vertex matching and detector-defined repair success do not certify manifoldness or absence of self-intersection. Targets must be connected; disconnected damage requires grouping or sequential processing. Results depend on reference quality and alignment. We have not established failure rates by patch size, curvature, or boundary complexity.

Possible extensions. An external image-to-3D method could reconstruct and align a dense surface to supply the reference point cloud. Deforming that reference or adding an appendage could likewise specify a new local shape for retessellation. These are prospective compositions, not direct image conditioning or experimentally validated capabilities in this work.

Acknowledgments and Disclosure of Funding

This work was conducted during Dingdong Yang's internship at Tencent Hunyuan, while he was a Ph.D. student at Simon Fraser University.

References

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. In *Seminal graphics: pioneering efforts that shaped the field*, pages 183–188. 1998.

Marco Centin, Nicola Pezzotti, and Alberto Signoroni. Poisson-driven seamless completion of triangular meshes. *Computer Aided Geometric Design*, 35:42–55, 2015.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 15343–15355, 2026. URL `https://proceedings.mlr.press/v306/chen26bv.html`.

Sijin Chen, Xin Chen, Anqi Pang, Xianfang Zeng, Wei Cheng, Yijun Fu, Fukun Yin, Zhibin Wang, Jingyi Yu, Gang Yu, et al. Meshxl: Neural coordinate field for generative 3d foundation models. *Advances in Neural Information Processing Systems*, 37:97141–97166, 2024.

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. Meshanything: Artist-created mesh generation with autoregressive transformers. In *ICLR*, 2025a.

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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13922–13931, 2025b.

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*, 36:35799–35813, 2023a.

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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023b.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*, 2022.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

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.

Shota Hattori, Tatsuya Yatagawa, Yutaka Ohtake, and Hiromasa Suzuki. Learning self-prior for mesh inpainting using self-supervised graph convolutional networks. *IEEE Transactions on Visualization and Computer Graphics*, 31(2):1448–1464, 2025.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics (ToG)*, 39(4):117:1–117:18, 2020.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Tao Ju. Robust repair of polygonal models. *ACM Transactions on Graphics (TOG)*, 23(3):888–895, 2004.

Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

Leif Kobbelt. $\sqrt{3}$-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 103–112, 2000.

Jiabao Lei, Kewei Shi, Zhihao Liang, and Kui Jia. ARMesh: Autoregressive mesh generation via next-level-of-detail prediction. In *NeurIPS*, 2025.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 7871–7880, 2020.

Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, 2003.

Stefan Lionar, Jiabin Liang, and Gim Hee Lee. TreeMeshGPT: Artistic mesh generation with autoregressive tree sequencing. In *CVPR*, 2025.

Hsueh-Ti Derek Liu, Vladimir G Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. Neural subdivision. *arXiv preprint arXiv:2005.01819*, 2020.

Jian Liu, Chunshi Wang, Song Guo, Haohan Weng, Zhen Zhou, Zhiqi Li, Jiaao Yu, Yiling Zhu, Jing Xu, Biwen Lei, and Chunchao Guo. QuadGPT: Native quadrilateral mesh generation with autoregressive models. In *ICLR*, 2026.

Charles Loop. Smooth subdivision surfaces based on triangles. 1987.

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, 2020.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571, 2022. doi: 10.18653/v1/2022.findings-naacl.117.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19615–19625, 2024.

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. In *ICLR*, 2025.

Lavanya Sita Tekumalla and Elaine Cohen. A hole-filling algorithm for triangular meshes. *School of Computing, University of Utah, UUCS-04-019, UT, USA*, 2, 2004.

Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, Tong Zhang, Shenghua Gao, and C. L. Philip Chen. Scaling Mesh Generation via Compressive Tokenization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11093–11103, 2025.

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)*, 42(4):1–16, 2023.

Ruowen Zhao, Junliang Ye, Zhengyi Wang, Guangce Liu, Yiwen Chen, Yikai Wang, and Jun Zhu. Deepmesh: Auto-regressive artist-mesh creation with reinforcement learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10612–10623, 2025.

Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 23(12):987–997, 2007.

Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, Bin Fu, Tao Chen, Gang Yu, and Shenghua Gao. Michelangelo: Conditional 3d shape generation based on shape-image-text aligned latent representation. *Advances in neural information processing systems*, 36:73969–73982, 2023.

Zhiwei Zhu, Xiang Gao, Lu Yu, and Yiyi Liao. Neural mesh refinement. *Frontiers of Information Technology & Electronic Engineering*, 26(5):695–712, 2025.

Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 189–192, 1996.

A Detailed Metric Definitions

We provide formal definitions of the evaluation metrics used in the main paper. Let $S = \{s_1, \ldots, s_N\}$ denote the test set of N samples. For each sample s, we have the ground-truth mesh $G^{(s)}$, the context region $(\mathcal{A}\backslash\mathcal{B})^{(s)}$, the target region $\mathcal{B}^{(s)}$, and the generated replacement $(\mathcal{B}')^{(s)}$. Let τ denote the vertex-matching tolerance (set to 10^{-6} throughout).

Matched boundary vertices. For sample s, let $V_\partial^{(s)}$ denote the exposed boundary-vertex set. The matched boundary vertices are those boundary vertices that coincide almost exactly (within τ) with at least one generated vertex:

$$V_{\mathrm{M}}^{(s)} = \big\{\, v \in V_\partial^{(s)} : \min_{u \in V_{\mathcal{B}'}^{(s)}} \|v - u\|_2 < \tau \big\}. \tag{8}$$

Vertex matching ratio. The per-sample vertex matching ratio measures how many boundary vertices are reproduced within the generated region:

$$r(s) = \frac{|V_{\mathrm{M}}^{(s)}|}{|V_\partial^{(s)}|} \tag{9}$$

PMR (Perfect Matching Ratio). The fraction of samples achieving perfect matching of all boundary vertices:

$$\mathrm{PMR} = \frac{1}{N} \sum_{s \in S} \mathbf{1}\big[r(s) = 1\big]. \tag{10}$$

A-VMR (Average Vertex Matching Ratio). The mean vertex matching ratio across all samples:

$$\text{A-VMR} = \frac{1}{N} \sum_{s \in S} r(s). \tag{11}$$

One-way Chamfer distance. For a point set X sampled from mesh A and a point set Y sampled from mesh B:

$$\mathrm{CD}(A \to B) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\|_2. \tag{12}$$

We sample 100,000 points from the ground truth G and 20,000 points from $\mathcal{B}$ and $\mathcal{B}'$.

O-CDIR and #F-Inc are computed only over the *perfect-match subset* $S_P = \{s \in S : r(s) = 1\}$. We use a one-way distance because $\mathcal{B}^{(s)}$ and $\mathcal{B}'^{(s)}$ are local patches, whereas $G^{(s)}$ is the full mesh. Accordingly, $\mathrm{CD}(A \to G^{(s)})$ checks whether the local patch A lies on the reference surface without penalizing the large part of $G^{(s)}$ outside the edited region.

O-CDIR (One-way CD Improvement Rate). The average relative reduction in one-way Chamfer distance:

$$\text{O-CDIR} = \frac{1}{|S_P|} \sum_{s \in S_P} \frac{\mathrm{CD}(\mathcal{B}^{(s)} \to G^{(s)}) - \mathrm{CD}(\mathcal{B}'^{(s)} \to G^{(s)})}{\mathrm{CD}(\mathcal{B}^{(s)} \to G^{(s)})}. \tag{13}$$

#F-Inc (Face Increase Ratio). The average relative change in face count:

$$\text{\#F-Inc} = \frac{1}{|S_P|} \sum_{s \in S_P} \frac{|F_{\mathcal{B}'}^{(s)}| - |F_{\mathcal{B}}^{(s)}|}{|F_{\mathcal{B}}^{(s)}|}. \tag{14}$$

CD-PR (Chamfer Distance Positive Rate). The fraction of all test samples for which the generated patch achieves a lower one-way Chamfer distance to the reference than the original target patch:

$$\text{CD-PR} = \frac{1}{N} \sum_{s \in S} \mathbf{1}\big[\mathrm{CD}(\mathcal{B}'^{(s)} \to G^{(s)}) < \mathrm{CD}(\mathcal{B}^{(s)} \to G^{(s)})\big]. \tag{15}$$

OvR (Overflow Rate). Let θ_{ovf} be the overflow ratio threshold (set to 0.01 throughout; see Alg. 3). OvR counts the fraction of test samples whose per-sample overflow ratio $O(s)$ exceeds θ_{ovf}:

$$\text{OvR} = \frac{\left|\{s \in S : O(s) > \theta_{\text{ovf}}\}\right|}{N}. \tag{16}$$

A-Overflow (Average Overflow Percentage). We define the per-sample overflow ratio $O(s)$ as the fraction of points sampled from $\mathcal{B}'^{(s)}$ that lie within distance ϵ_{ovf} of the residual faces of the original mesh (excluding $\mathcal{B}$ and $\mathcal{A}\setminus\mathcal{B}$). Let $S_{>\theta_{\text{ovf}}} = \{s \in S : O(s) > \theta_{\text{ovf}}\}$ be the subset of samples with non-negligible overflow. The average overflow percentage is:

$$\text{A-Overflow} = \frac{1}{|S_{>\theta_{\text{ovf}}}|} \sum_{s \in S_{>\theta_{\text{ovf}}}} O(s). \tag{17}$$

We set ϵ_{ovf} to the default overflow detection threshold throughout all experiments.

No-edit reference for OvR and A-Overflow. The proximity threshold can register nearby residual surfaces even for the original target patch. Substituting $\mathcal{B}'^{(s)} \equiv \mathcal{B}^{(s)}$ gives $\text{OvR} = 6.25\%$ and $\text{A-Overflow} = 3.49\%$ on the ablation set (Table 2). These are empirical no-edit reference values, not mathematical lower bounds: generated patches can have different surface coverage and proximity statistics. Approaching these values does not prove that all residual overflow is measurement noise.

B Baseline Implementation Details

Because our test meshes contain open-boundary low-poly meshes (with quads triangulated for baselines that require triangles), several baselines need non-trivial preprocessing or post-processing to produce a local patch for evaluation. Table 3 summarizes the key requirements; the rest of this section details only the non-obvious steps.

Table 3: Baseline preprocessing and evaluation setup. "Closed mesh" means the method requires a watertight manifold input. "Global" means the method operates on the entire mesh (not just the target region).

Method	Closed	Global	Stitch	Preprocessing / Notes
Modified Butterfly	✗	✓	✗	Face-id tracking after subdivision
Neural Subdivision	✓	✓	✗	fTetWild Hu et al. [2020] to close mesh + face-id remap
Neural Mesh Refinement	✓	✓	✗	fTetWild Hu et al. [2020] to close mesh + face-id remap
Liepa Hole Filling	✗	—	✗	Remove $\mathcal{B}$; fill boundary loop via min-weight triangulation
SeMIGCN	✗	—	✗	MeshFix to close hole + per-shape GCN optimization (100 iter)
BPT + Stitch Back	✗	✓	✓	Regenerate whole mesh from point cloud; spatial crop + boundary snap
MeshAnythingV2 + Stitch	✗	✓	✓	Regenerate whole mesh from point cloud; spatial crop + boundary snap
MeshFIM (Ours)	✗	✗	✗	Local generation; boundary merge

Why subdivision baselines are non-local. Neural Subdivision Liu et al. [2020] and Neural Mesh Refinement Zhu et al. [2025] are *global* subdivision operators: they take the full mesh connectivity as input and predict new vertex positions for *every* subdivided face simultaneously (the network's receptive field spans the entire mesh via pooling hierarchies). There is no mechanism to subdivide only a selected region while keeping the rest unchanged.

Patch extraction for subdivision baselines. Because subdivision is applied globally, we track which original face each subdivided face descends from: after L levels of 1-to-4 subdivision, each original face produces 4^L child faces that inherit its source id. The target-region patch is then extracted by collecting all child faces whose inherited source id belongs to $\mathcal{B}$. No stitch-back post-processing is applied, because the extracted patch is not independently generated—it is carved out of the same subdivided mesh that contains the context. However, the subdivision *itself* modifies the surrounding geometry: vertex positions along and near the boundary are repositioned by the learned (or interpolatory) subdivision rules, and the boundary of the extracted patch no longer coincides with the original context boundary vertices. This explains the low A-VMR scores in Table 1 (0% for Neural Subdivision, 33.68% for Neural Mesh Refinement).

fTetWild preprocessing (Neural Subdivision, Neural Mesh Refinement). Both methods further assume a closed manifold input and crash on open-boundary meshes. We close each test mesh using fTetWild Hu et al. [2020] with its `-manifold-surface` flag and default envelope (10^{-3} of the bounding-box diagonal). Because fTetWild may re-triangulate or add cap faces, we remap each output face to its nearest original face (centroid-to-surface query) and propagate the original face source id; cap faces (distance above $10\times$ envelope) are marked with id -1. After subdivision, the target-region patch is extracted by collecting all output faces whose inherited source id belongs to $\mathcal{B}$.

Stitch-back scheme (BPT, MeshAnythingV2). These models regenerate the *entire* mesh from the ground-truth point cloud and have no mechanism for local editing. To extract a local patch for evaluation:

1. **Spatial crop:** select faces of the regenerated mesh whose centroids fall inside the bounding box of the original $\mathcal{B}$ (expanded by 5% of the diagonal).
2. **Boundary snap:** for each context boundary vertex, find the nearest cropped-patch boundary vertex via KD-tree; if within $2\times$ the patch's average edge length, snap the patch vertex to the exact context position. Unmatched context vertices are appended as isolated vertices in the stitch-back representation. Isolated vertices do not constitute an attached surface; matching scores must not be read as connectivity certificates.

This generous tolerance favors the baselines; nonetheless, boundary matching remains poor because the independently generated mesh has no constraint to align with the surrounding context.

SeMIGCN per-shape optimization. Unlike methods that use a pre-trained model, SeMIGCN trains a small GCN from scratch for every test sample (100 iterations). The target faces are removed to expose a hole, MeshFix closes the mesh conservatively, and the GCN learns vertex displacements in a self-supervised loop. A Laplacian-based refinement step then smooths the result. Because the optimization is unsupervised and shape-agnostic, it relies on local smoothness priors rather than learned geometry, explaining its limited geometric accuracy.

C Automatic Defect Repair

C.1 Algorithmic Details

The automatic repair pipeline in Section 5.2 has four stages: detecting candidate defects with multi-view inspection and reference-mesh ray casting (Algorithm 1); extracting connected target regions and their context (Algorithm 2); generating replacement patches with MeshFIM; and filtering and merging the patches (Algorithm 3). These stages can be repeated to repair a mesh iteratively (Algorithm 4). Default parameters are listed in the algorithm captions. Figure 9 illustrates defect detection and region extraction.

C.2 Repair Results

We evaluate the iterative repair pipeline on 2560 meshes generated by an independent AR mesh generator. Table 4 reports the overall repair statistics and the per-round breakdown.

Out of 2417 damaged meshes, 1573 (65.1%) are fully repaired after a single round. Subsequent rounds fix progressively fewer objects as the remaining defects tend to be more severe or involve larger disconnected regions. After 4 rounds the pipeline achieves an overall repair rate of 88.9%.

A mesh is marked as "successfully repaired" when Alg. 1 detects zero broken-point candidates in the subsequent round. Note that our detection algorithm is deliberately sensitive: it renders the mesh from 48 viewpoints at high resolution (640×640) and confirms every candidate against the reference mesh via ray casting, so even subtle artifacts such as a single flipped face or a tiny hole spanning only a few pixels can be flagged in the shown examples (see Figure 8 for examples of detected defects at various severity levels). This detector-defined success criterion does not certify watertightness, manifoldness, or absence of defects invisible to the detector.

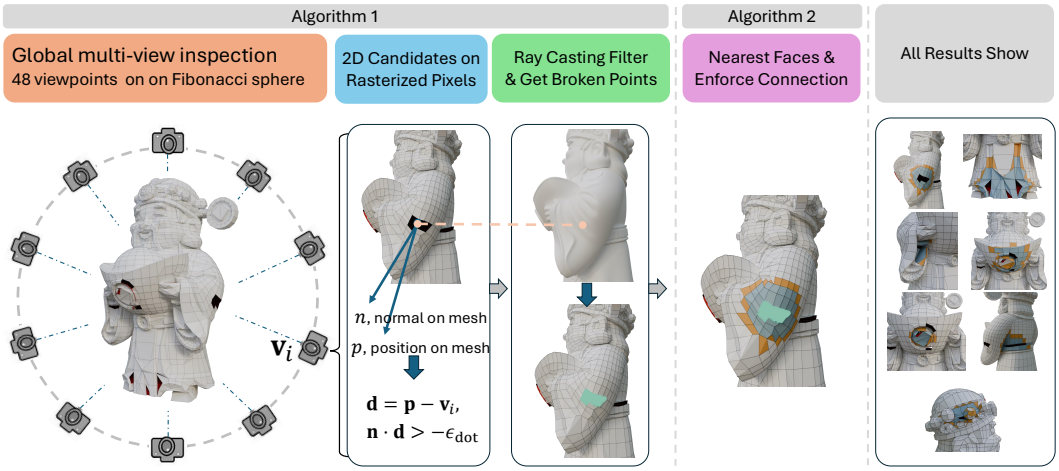


Figure 9: The automatic defect detection pipeline. The pipeline combines multi-view inspection, reference-mesh ray casting, and connected-region extraction.

Table 4: **Iterative repair statistics** on 2560 generated meshes (max $T = 4$ rounds).

Category	Count	Percentage
Total objects	2560	—
Initially damaged	2417	94.4%
Never damaged (no defects detected)	143	5.6%
Successfully repaired	2149	88.9% of damaged
Still broken after 4 rounds	268	11.1% of damaged

Round	Newly Fixed	Cumulative Fixed	Cumulative %
1	1573	1573	65.1%
2	307	1880	77.8%
3	168	2048	84.7%
4	101	2149	88.9%

D Percolation-Based Region Sampling

To prevent the model from overfitting to regular circular patches, we sample a fraction of training target regions via stochastic growth on the face adjacency graph. Starting from a random seed face, the region expands along the current frontier: for each candidate neighboring face, we add it with probability p until the target face budget is reached. p is sampled uniformly from $[0.55, 0.85]$ per example. If no frontier face is accepted in one round, the algorithm falls back to expanding through the remaining frontier so that the sampled region stays connected and continues to grow. Compared with fixed-radius BFS patches, this produces connected but irregular regions that better resemble brush selections or automatically detected defects.

E Low Poly Encoder Gate Network Implementation

This appendix details the gate network N_{gate} of section 3 (Eq. 7).

Inputs and shared query. The two point-cloud branches share a single Perceiver-IO encoder E. Its cross-attention queries are positions $\mathbf{q} = \{\mathbf{q}_i\}_{i=1}^{M}$ in Euclidean space obtained from the reference branch; the low-poly branch reuses the same $\mathbf{q}$, which is what makes the two latent sequences $\mathbf{z}_{\text{gt}}, \mathbf{z}_{\text{lp}} \in \mathbb{R}^{M \times d}$ position-aligned.

Algorithm 1 Broken Point Detection. Default parameters: $N_v = 48$ views, rasterization resolution $H = 640$, dot-product threshold $\epsilon_{\text{dot}} = 10^{-4}$, DBSCAN [Ester et al., 1996] radius $\epsilon_{\text{cls}} = 0.05$, minimum cluster size $\tau_{\text{bp}} = 10$.

Require: Mesh to repair $\mathcal{M}$, reference mesh $\mathcal{M}_{\text{ref}}$
Ensure: Broken-point clusters $\{C_1, \ldots, C_K\}$
1: Normalize $\mathcal{M}$ and $\mathcal{M}_{\text{ref}}$ to unit sphere.
2: Generate N_v viewpoints on a Fibonacci sphere.
3: **for** each viewpoint $\mathbf{v}_i$ **do**
4: Rasterize $\mathcal{M}$ at resolution H; record per-pixel face normal $\mathbf{n}$ and position $\mathbf{p}$.
5: Compute view direction $\mathbf{d} = \mathbf{p} - \mathbf{v}_i$.
6: Mark pixel as *candidate* if $\mathbf{n} \cdot \mathbf{d} > -\epsilon_{\text{dot}}$ (back-face test).
7: For each candidate, cast a ray from $\mathbf{v}_i$ through $\mathbf{p}$ against $\mathcal{M}_{\text{ref}}$; confirm the candidate if the ray hits a correctly oriented reference face first.
8: **end for**
9: Cluster all confirmed broken points with DBSCAN (ϵ_{cls}); discard clusters with fewer than τ_{bp} points.
10: **return** $\{C_1, \ldots, C_K\}$

Algorithm 2 Damage Region Extraction. Default parameters: B-expansion width $w_{\mathcal{B}} = 3$, context-expansion width $w_{\mathcal{A}\setminus\mathcal{B}} = 1$.

Require: Broken-point clusters $\{C_k\}$, mesh $\mathcal{M}$
Ensure: Damage groups $\{(\mathcal{B}_j,\ (\mathcal{A}\setminus\mathcal{B})_j)\}$
1: **for** each cluster C_k **do**
2: Map every point in C_k to its nearest face in $\mathcal{M} \to$ seed faces $F^{(k)}_{\text{seed}}$.
3: If seed faces are not face-connected, link disjoint components via shortest-path BFS on the face adjacency graph.
4: BFS-expand $F^{(k)}_{\text{seed}}$ by $w_{\mathcal{B}}$ rings $\to$ target region $\mathcal{B}_k$.
5: BFS-expand the boundary of $\mathcal{B}_k$ by $w_{\mathcal{A}\setminus\mathcal{B}}$ ring $\to$ context region $(\mathcal{A}\setminus\mathcal{B})_k$.
6: **end for**
7: Merge groups whose $\mathcal{B}$ or $\mathcal{A}\setminus\mathcal{B}$ regions overlap using Union-Find.
8: **return** $\{(\mathcal{B}_j,\ (\mathcal{A}\setminus\mathcal{B})_j)\}$

Architecture of N_{gate}. The gate vector $\mathbf{g} \in (0,1)^M$ is produced by a lightweight stack operating on the channel-wise concatenation of the two latents:

$$
\begin{aligned}
&\mathbf{h} = [\mathrm{LN}(\mathrm{sg}(\mathbf{z}_{\text{gt}}))\,;\ \mathrm{LN}(\mathrm{sg}(\mathbf{z}_{\text{lp}}))] \in \mathbb{R}^{M\times 2d},\\
&\mathbf{h} \leftarrow \mathbf{h} + \mathrm{SelfAttn}(\mathbf{h}),\\
&\mathbf{h} \leftarrow \mathbf{h} + \mathrm{FFN}(\mathrm{LN}(\mathbf{h})),\\
&\mathbf{h} \leftarrow \mathrm{TransformerEncoder}_1(\mathbf{h}),\\
&\mathbf{g} = \sigma\big(\mathbf{w}^\top \mathbf{h} + b\big),
\end{aligned}
$$

where $\mathrm{sg}(\cdot)$ denotes stop-gradient. The output head is a linear map to a scalar at each token position; its weights are initialized with $\mathcal{N}(0, 10^{-3})$ and its bias with $b = -4$, so that at the start of training $g_i \approx \sigma(-4) \approx 0.018$, yielding $\hat{\mathbf{z}} \approx \mathbf{z}_{\text{gt}}$ initially. The FFN follows the standard two-layer GELU design with hidden size $2 \cdot 2d$.

F Interactive Brush Selection Interface

Figure 10 shows our browser-based brush selection tool. The interface presents a synchronized dual-panel view: the left panel displays the low-poly mesh for brush painting, while the right panel shows the reference geometry for visual guidance. Both share the same camera.

Face-level brush. The brush selects whole faces via raycasting, supporting two modes: (i) a *2D* mode that BFS-expands from the hit face by a user-specified ring count on the adjacency graph, and

Algorithm 3 Quality Gate & Merge. Default parameters: overflow distance $\epsilon_{\text{ovf}} = 1/256$, overflow ratio threshold $\theta_{\text{ovf}} = 0.01$, weld tolerance $\tau_{\text{weld}} = 10^{-6}$.

Require: Current mesh $\mathcal{M}_{i-1}$, target region $\mathcal{B}$, generated patch $\mathcal{B}'$, context $\mathcal{A} \setminus \mathcal{B}$
Ensure: Updated mesh $\mathcal{M}_i$ or REJECT

// Quality Gate
Sample point cloud $\mathbf{P}_{\mathcal{B}'}$ on $\mathcal{B}'$.
Let $K = \mathcal{M}_{i-1} \setminus (\mathcal{B} \cup (\mathcal{A} \setminus \mathcal{B}))$ be the remaining mesh.
Compute overflow ratio: fraction of $\mathbf{P}_{\mathcal{B}'}$ within distance ϵ_{ovf} of K.
if overflow ratio $> \theta_{\text{ovf}}$ **then**
  return REJECT
end if
// Merge
Remove target faces: $\mathcal{M}_i \leftarrow \mathcal{M}_{i-1} \setminus \mathcal{B}$.
Build KD-tree on boundary vertices of $\mathcal{M}_i$.
For each vertex of $\mathcal{B}'$ within τ_{weld} of a boundary vertex, weld (snap) to the existing vertex.
Append $\mathcal{B}'$ faces and welded vertices to $\mathcal{M}_i$.
return $\mathcal{M}_i$

Algorithm 4 Iterative Hole Repair. Default parameters: max rounds $T = 4$, false-positive threshold $\tau_{\text{fp}} = 2$ points.

Require: Initial mesh $\mathcal{M}_0$, reference mesh $\mathcal{M}_{\text{ref}}$, reference point cloud $\mathbf{P}$
Ensure: Repaired mesh $\mathcal{M}_T$

for $i = 1$ **to** T **do**
  $\{C_k\} \leftarrow$ DETECTBROKENPOINTS($\mathcal{M}_{i-1}$, $\mathcal{M}_{\text{ref}}$) {Alg. 1}
  if $\sum_k |C_k| = 0$ **then**
    return $\mathcal{M}_{i-1}$ {no damage}
  else if $\sum_k |C_k| \leq \tau_{\text{fp}}$ **then**
    return $\mathcal{M}_{i-1}$ {likely false positive}
  end if
  $\{(\mathcal{B}_j, (\mathcal{A} \setminus \mathcal{B})_j)\} \leftarrow$ EXTRACTREGIONS($\{C_k\}$, $\mathcal{M}_{i-1}$) {Alg. 2}
  $\mathcal{M}_i \leftarrow \mathcal{M}_{i-1}$
  for each group $(\mathcal{B}_j, (\mathcal{A} \setminus \mathcal{B})_j)$ **do**
    $\mathcal{B}'_j \leftarrow$ MESHFIM($(\mathcal{A} \setminus \mathcal{B})_j, \mathbf{P}$)
    $\mathcal{M}_{\text{cand}} \leftarrow$ QUALITYGATEMERGE($\mathcal{M}_i$, $\mathcal{B}_j$, $\mathcal{B}'_j$, $(\mathcal{A} \setminus \mathcal{B})_j$) {Alg. 3}
    if $\mathcal{M}_{\text{cand}} \neq$ REJECT **then**
      $\mathcal{M}_i \leftarrow \mathcal{M}_{\text{cand}}$
    end if
  end for
end for
return $\mathcal{M}_T$

(ii) a *3D* mode that selects all faces whose centroids fall within a world-space sphere. An optional backface test restricts selection to visible geometry.

Connectivity enforcement. Each committed stroke is filtered to its largest connected component. Strokes not adjacent to the existing selection are flagged as orphans and excluded, guaranteeing that the final $\mathcal{B}$ is always a connected patch—a requirement of the FIM serialization.

Context preview. The tool displays the context ring $\mathcal{A} \setminus \mathcal{B}$ in real time by BFS-expanding from the boundary of $\mathcal{B}$ with a user-adjustable width, giving immediate feedback on what the model will condition on.

G More results

Additional local editing results are shown in Figure 11.

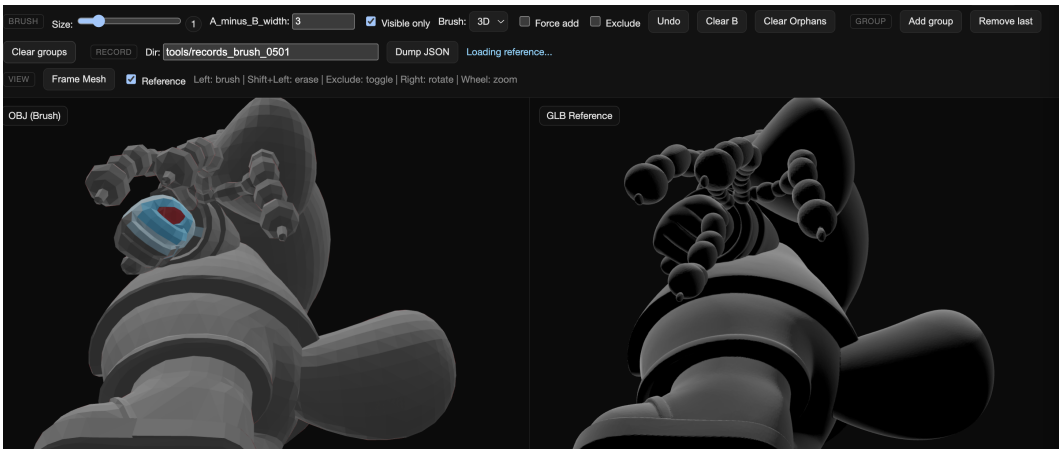


Figure 10: **Brush editing interface.** Left: low-poly mesh with target $\mathcal{B}$ (blue) and context $\mathcal{A}\backslash\mathcal{B}$ (light blue). Back-facing faces are tinted dark red to make it easier for users to observe broken regions. Right: reference GT mesh. Top bar: brush size, mode (2D/3D), context width, and navigation controls.

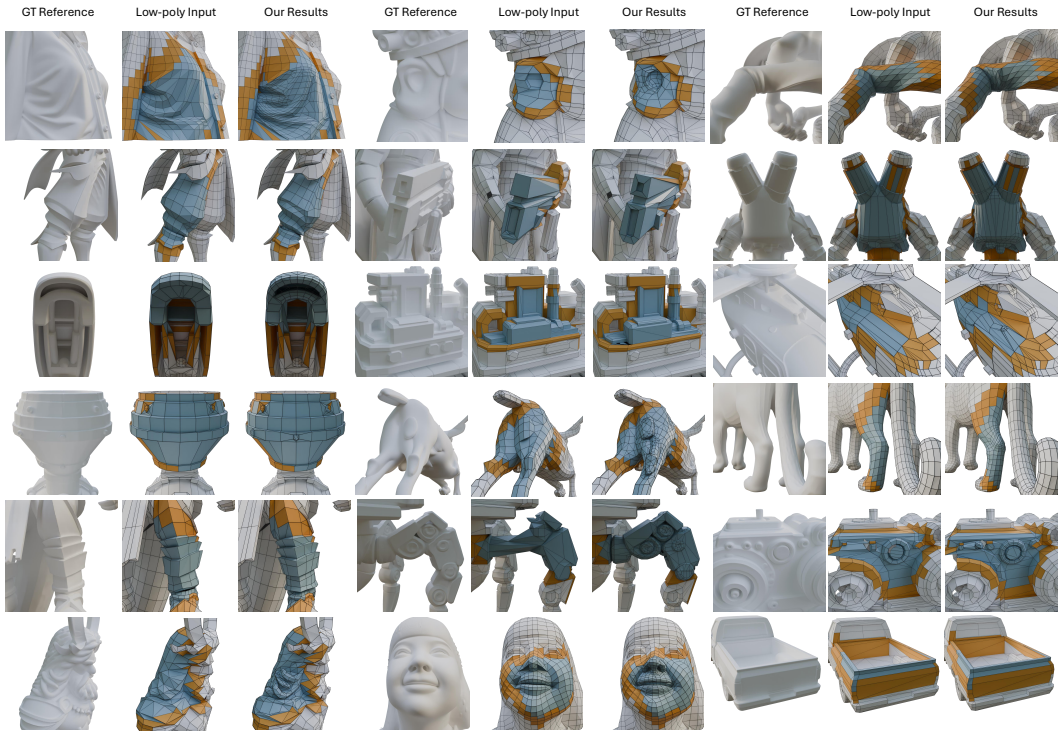


Figure 11: More results of our method. Our method is able to do accurate local editing on various types of low-poly mesh.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the contributions (a Fill-in-the-Middle framework for local low-poly mesh editing with five design choices) and the scope (local mesh editing conditioned on surrounding context). Experimental results in Section 4 validate these claims with quantitative metrics and ablation studies.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion (Section 6) explicitly discusses the limitation that the current method assumes the target region is a single connected region, and identifies extending to disconnected components as future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include theoretical results. It is a systems/methods paper with empirical evaluation.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides full implementation details including model architecture (Hourglass Transformer, d=1536, depth 24, 16 heads), training setup (64 computing units, context augmentation noise level), region sampling strategy, dataset sources (ShapeNet, Objaverse, ObjaverseXL), test set construction (256 meshes $\times$ 4 regions), and all metric definitions in Appendix A. Baseline implementation details are in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The training data uses publicly available datasets (ShapeNet, Objaverse, ObjaverseXL). Code release is under consideration but not confirmed at submission time. The paper provides sufficient algorithmic and architectural details for independent reproduction.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 and the appendices provide comprehensive details: model architecture and dimensions, training hardware, augmentation parameters, region sampling strategies (BFS and percolation-based), dataset sources and test set construction, and all hyperparameters for the automatic repair pipeline (Appendix C).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: Our evaluation is conducted on a fixed test set (Section 4.1) with deterministic inference (greedy autoregressive decoding). There is no stochastic variation across runs, so error bars are not applicable in the traditional sense. The metrics are computed over the full test set rather than subsampled.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 specifies the training setup: 64 computing units each with approximately 90 GB memory. The model architecture dimensions are fully specified, allowing estimation of compute requirements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research involves geometric mesh processing and does not raise ethical concerns. All datasets used are publicly available and properly cited. No human subjects or private data are involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work focuses on local editing of low-poly 3D meshes, a foundational geometry processing tool for computer graphics workflows (game assets, CAD, scan processing). It does not have direct negative societal implications, as it is a technical tool for mesh topology manipulation without foreseeable dual-use concerns.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The model performs local low-poly mesh editing, which does not pose risks for misuse comparable to language models or image generators. The output is geometric mesh data (vertex coordinates and face connectivity) without capacity for generating harmful content.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets (ShapeNet, Objaverse, ObjaverseXL) and baseline methods are properly cited with their original publications. The data processing pipeline follows prior work and is cited accordingly.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new datasets, models, or code as public assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used only for grammar checking and writing assistance during manuscript preparation. They are not a component of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.