WIREFORGE: STRUCTURED LOW-POLY MESH GENERATION WITH 2D WIREFRAME PRIORS

Anonymous authors
Paper under double-blind review

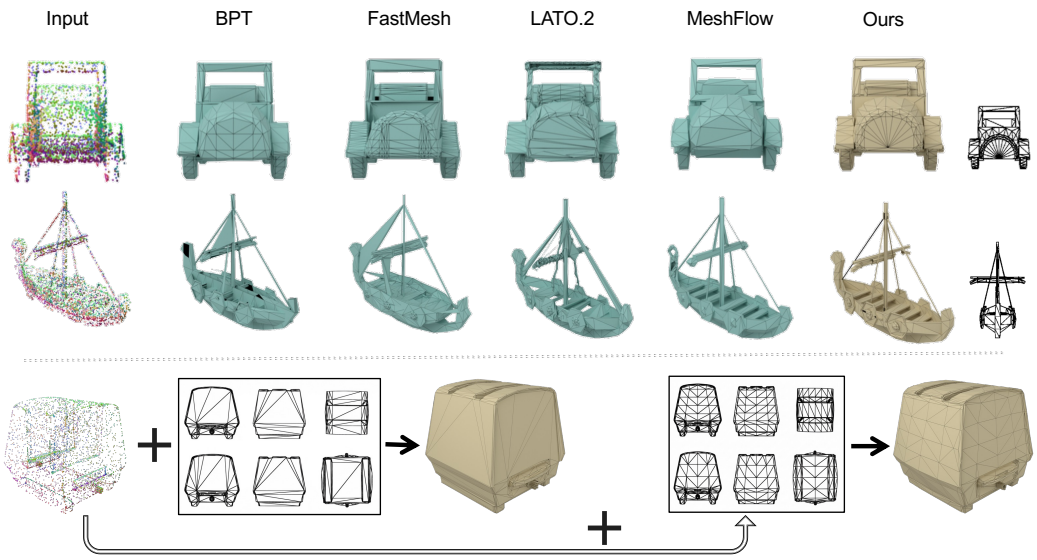


Figure 1: WireForge jointly preserves geometric fidelity and enables explicit control over low-poly edge layouts through a 2D wireframe reference. Different wireframe conditions produce distinct edge-layout styles while preserving the same input geometry.

ABSTRACT

Low-poly mesh generation is fundamentally underdetermined: the same input shape can be approximated by different polygonal layouts. Existing generators rely on surface conditioning and mesh-connectivity priors learned from the training distribution to select among these layouts, but matching the input geometry does not guarantee that edges follow salient structures. We present *WireForge*, a framework that addresses this gap by introducing 2D wireframe priors as explicit structural references for generation. These references are constructed by a wireframe-generation agent, *WirePilot*, built on off-the-shelf image-generation models. Guided by either our carefully designed default instructions or user-provided prompts, the agent generates, evaluates, and refines candidate wireframes. To translate these image-space references into 3D guidance, WireForge associates their features with spatial anchors derived from the input mesh and applies 3D positional encoding. The spatially grounded features then guide mesh generation through cross-attention, complementing geometric guidance from surface samples. This conditioning formulation can be adapted to different mesh-generation backbones, while retaining the wireframe as an editable interface for specifying edge-layout styles through natural-language instructions or sketches. Experiments demonstrate that the resulting structural guidance improves both geometric fidelity and edge-layout alignment.

1 INTRODUCTION

Low-poly mesh generation seeks to represent a 3D surface with a compact set of vertices, edges, and faces. Such representations are important for game asset creation and interactive 3D model-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing, where meshes must balance computational efficiency with geometric fidelity and ease of editing. The task is fundamentally underdetermined: even for the same target surface and face count, substantially different polygonal layouts can provide plausible approximations, while differing in contour preservation, connectivity, and structural quality. Consequently, a low-poly mesh cannot be judged by surface coverage alone. Its sparse edges must also be allocated to the structures that matter, including silhouettes, part boundaries, and sharp geometric transitions.

Recent mesh-native generative models directly synthesize polygonal structure through autoregressive prediction or diffusion-based generation. Autoregressive approaches represent meshes as vertex–face sequences or structured tokens (Nash et al., 2020; Siddiqui et al., 2024; Chen et al., 2025a; Tang et al., 2025; Weng et al., 2024; 2025), while diffusion-based approaches generate polygonal meshes either directly or through learned mesh latents (Alliegro et al., 2023; He et al., 2025; Zhao et al., 2026; Li et al., 2026). Despite their different representations, these methods largely rely on surface conditioning and mesh-connectivity priors learned from the training distribution to determine polygonal layouts. However, matching the overall surface does not necessarily imply that sparse edges follow salient contours or part boundaries. This motivates complementing surface conditioning with an explicit structural reference for edge placement.

Our key observation is that edge-layout guidance need not be inferred solely from surface-sampled point clouds and learned mesh-connectivity priors. Instead, complementary structural guidance can be expressed in the image domain, where 2D wireframes provide a compact and editable reference for edge placement along silhouettes, part boundaries, and other salient structures. Moreover, modern 2D image-generation models benefit from larger and more diverse training data than current 3D mesh generators, making them a promising source of transferable structural priors for edge-layout reasoning. Based on this insight, we present *WireForge*, a framework that translates 2D wireframe priors into spatially grounded guidance for structured low-poly mesh generation. To construct these priors, we introduce *WirePilot*, a wireframe-generation agent that leverages off-the-shelf image-generation models to generate, evaluate, and iteratively refine wireframe references from normal maps rendered from the input mesh in one or more views. WirePilot operates with default instructions while also supporting optional user prompts for customized guidance.

While prior work explores geometric and pose controls (Zhang et al., 2024; Team Hunyuan3D et al., 2025), our goal is fine-grained control over mesh edge placement and layout. Inspired by image-to-3D feature alignment in texture generation (He et al., 2026), WireForge associates wireframe features with coarse spatial anchors derived from the input mesh. These anchors and 3D positional encoding spatially ground the wireframe features, which are injected into the generative backbone through cross-attention to complement geometric guidance from surface samples. This spatially grounded conditioning preserves the wireframe as an explicit structural reference, enabling users to further control local edge layouts through natural-language instructions or sketches before mesh generation. With this structural guidance, WireForge improves both geometric fidelity and edge-layout alignment, as demonstrated by our experiments.

Our contributions are summarized as follows:

- We rethink low-poly mesh generation by introducing 2D image priors as explicit structural guidance. Our framework, *WireForge*, is the first native low-poly mesh-generation framework that enables explicit control over edge-layout styles through wireframe images.
- We design *WirePilot*, a wireframe-generation agent that leverages off-the-shelf image-generation models to generate, evaluate, and refine structural references. It supports optional user instructions to customize edge flow and density during reference construction.
- We associate wireframe features with 3D surface locations using surface-coordinate anchors and spatial positional encoding, and condition the mesh generator on them via cross-attention, allowing image-space structure to guide sparse edge placement in 3D.
- We formulate wireframe guidance as a backbone-agnostic conditioning interface that can be adapted to different mesh generators. Its explicit image representation provides a practical control interface, allowing users to specify the desired edge-layout style through natural-language instructions or sketches without retraining the mesh generator.

2 Related Work

Low-poly mesh generation. Mesh-native generators directly synthesize polygonal structure through autoregressive prediction or diffusion-based generation, including flow matching. Autoregressive methods model vertex–face sequences or mesh tokens (Nash et al., 2020; Siddiqui et al., 2024; Chen et al., 2025a;b; Tang et al., 2025), with extensions for pivot-vertex guidance, compressive tokenization, and component decoupling (Weng et al., 2024; 2025; Kim et al., 2025). Diffusion-based methods generate meshes directly or through learned mesh latents (Alliegro et al., 2023; He et al., 2025), with recent flow-matching approaches further advancing this direction (Li et al., 2026; Zhao et al., 2026). Recent industry-led advances, including Meshy T2 (Xu et al., 2026) and Nexus (Wang et al., 2026), demonstrate strong native mesh generation capabilities, but do not provide explicit conditioning on user-specified edge-layout styles. WireForge complements learned mesh-connectivity priors with explicit 2D wireframe guidance through a conditioning formulation adaptable to both generative paradigms.

Controllable generation and mesh structure. Spatial conditions enable control in image generation (Zhang et al., 2023), while CLAY, Hunyuan3D-Omni, and PoseMaster explore geometric and pose guidance for 3D assets (Zhang et al., 2024; Team Hunyuan3D et al., 2025; Yan et al., 2026). Mesh-native methods also support vertex- or face-count control (He et al., 2025; Li et al., 2026; Xu et al., 2026; Wang et al., 2026) and pivot-vertex guidance (Weng et al., 2024). Our focus is instead on explicit control over edge-layout styles through editable wireframe references. Closer to this goal, quad-meshing methods use cross fields or structural lines to guide edge flow and feature preservation (Decatur et al., 2026; Chen et al., 2026), with the final layout obtained through downstream parameterization and quad extraction. WireForge instead incorporates agent-generated, editable wireframe images as structural conditions for mesh-native generation, enabling users to guide local edge layouts through text or sketches without explicit field-based extraction.

3D-aware image conditioning Recent texture generators spatially ground 2D image features on 3D surfaces to improve cross-view correspondence. Hitem3D 2.0 combines image-VAE features, canonical coordinate maps, and 3D rotary positional encoding for geometry-aware cross-attention (He et al., 2026), while related methods also exploit surface geometry for coherent multi-view conditioning (Feng et al., 2025; Liu et al., 2025). These works motivate WireForge's grounding design, which associates wireframe features with spatial anchors derived from the input mesh before conditioning mesh generation. We adapt this principle from appearance synthesis to structural guidance: the condition specifies edge layout rather than surface appearance, and the output is polygonal geometry rather than texture.

3 Method

3.1 Overview and Problem Setting

Let $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ be a low-poly mesh with vertices $\mathcal{V}$ and faces $\mathcal{F}$. Given a dense input shape, a surface-conditioned generator models $p(\mathcal{M} \mid \mathcal{S})$, where $\mathcal{S}$ denotes features sampled from the input surface. As discussed in Section 1, surface evidence constrains the geometry but does not uniquely determine the edge layout: many polygonizations can approximate the same surface, leaving the generator to resolve topology largely from its learned mesh prior.

WireForge introduces an explicit image-space condition for this remaining structural freedom. We represent the desired layout by one or more 2D wireframe references $\mathcal{W} = \{I_v\}_{v=1}^{V}$, rendered or drawn from known viewpoints. For the same cameras, we rasterize canonical coordinate maps $\mathcal{P} = \{P_v\}_{v=1}^{V}$ and normal maps $\mathcal{N} = \{N_v\}_{v=1}^{V}$ from the input shape. The resulting model is

$$p\big(\mathcal{M} \,\big|\, \mathcal{S}, \mathcal{W}, \mathcal{P}, \mathcal{N}\big), \tag{1}$$

where $\mathcal{S}$ constrains the underlying surface and the wireframe branch provides an explicit cue for how edges may be distributed over that surface. The maps $\mathcal{P}$ and $\mathcal{N}$ are deterministic renderings of the same input shape rather than additional user-provided conditions. During training, $\mathcal{W}$ is rendered from the target mesh; at inference, it can instead be proposed or edited in image space and remains directly inspectable.

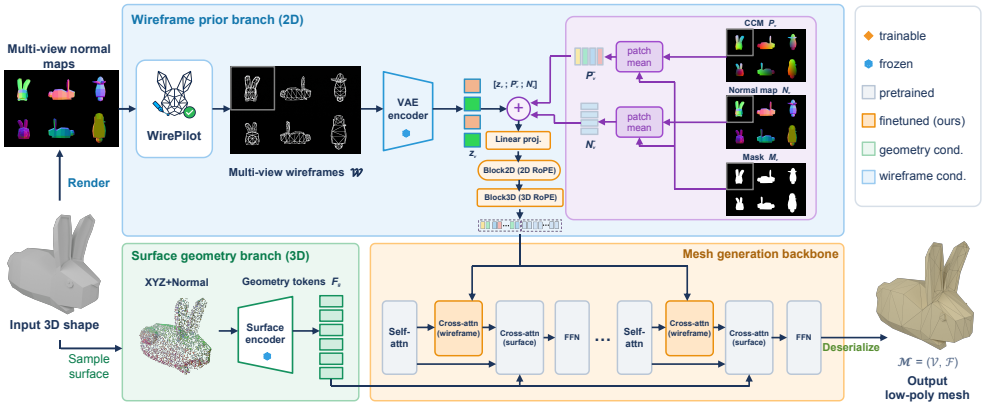


Figure 2: WireForge overview. WirePilot constructs editable multi-view wireframes, which are spatially grounded and injected into a surface-conditioned mesh generator.

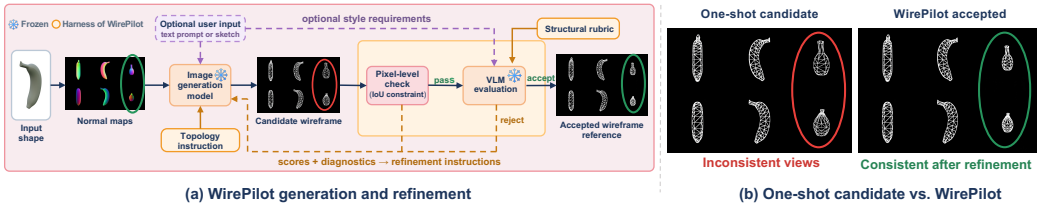


Figure 3: WirePilot workflow. **(a)** Candidate generation, geometric screening, structural evaluation, and iterative refinement. **(b)** Refinement resolves cross-view inconsistencies in a one-shot candidate.

As illustrated in Figure 2, the framework has two stages. First, *WirePilot* constructs and refines 2D wireframe references from rendered normal maps, optionally following user instructions (Section 3.2). Second, WireForge encodes these images, grounds their features on the 3D surface, fuses evidence across views, and injects the resulting tokens into a mesh generator (Section 3.3). Besides local edge placement, reference line density provides a polygon-density cue. In our implementation, this cue is passed through the wireframe rather than through a separate scalar face-count token to the mesh generator.

3.2 2D Wireframe Generation

Figure 3 illustrates the iterative WirePilot workflow. Given an input shape, WirePilot renders normal maps as geometric references and uses an off-the-shelf image-generation model (Gemini 3 Pro Image (Raisinghani, 2025)) to propose wireframes under a default edge-layout instruction. It evaluates each candidate and, when needed, converts the resulting diagnostics into refinement instructions for another generation round. Optional user prompts can customize properties such as local edge flow or density.

Generation. We render the input shape from six canonical orthographic views and arrange the normal maps in a 2×3 grid. In the default six-view mode, the image generator returns a pixel-aligned grid in one call; the same interface can also be used with a single view. Joint generation encourages cross-view consistency while keeping the reference in an ordinary image format that can be inspected before mesh synthesis.

Evaluation and refinement. *WirePilot* operates under a fixed evaluation harness that combines geometric verification with higher-level structural assessment. Each candidate first passes a deterministic image-space check that verifies silhouette coverage, out-of-bounds ink, and view ordering against the input normal maps. This check is designed to filter out common artifacts introduced by the image-generation model, such as reordered views, inconsistent object scale, spatial misalignment, and other generation failures.

A structural evaluator then scores candidates that pass the geometric checks based on edge layout, density allocation, and cross-view consistency, optionally incorporating user-specified style requirements. The resulting diagnostics are converted into refinement instructions and fed back to the image generator until a conditioning reference is selected.

3.3 WIREFRAME-CONDITIONED MESH GENERATION

The mesh-generation stage translates the 2D structural reference into features that can guide a 3D generator. The central difficulty is that an image-space edge has no explicit correspondence to a mesh element. We address this by separating the pathway into image encoding, geometric grounding and multi-view fusion, and a lightweight conditioning interface. During training, each wireframe is paired with the target mesh from which it is rendered; using multiple simplification levels associates line layout and line density with the connectivity and polygon budget of the target without introducing a separate scalar face-count token in the mesh generator.

3.3.1 AUTOREGRESSIVE MESH BACKBONE

We instantiate WireForge on a single-stage autoregressive backbone in the blocked-and-patchified tokenization (BPT) family (Weng et al., 2025). BPT factorizes quantized vertex coordinates into coarse spatial blocks and local offsets, and orders neighboring faces in patchified traversals, yielding a compact mesh-native sequence. The pretrained backbone is conditioned on surface features extracted from the input shape through its original cross-attention pathway and predicts the mesh sequence autoregressively with the standard next-token objective.

WireForge leaves this surface-conditioning pathway intact and adds the wireframe branch as a complementary structural condition. The surface features therefore constrain the target geometry, while the wireframe condition guides how the available polygons are organized over that surface.

3.3.2 WIREFRAME ENCODING WITH AN IMAGE VAE

To reduce the appearance gap between rendered and generated wireframes, we binarize the input and apply mild stroke-width augmentation during training. Each view is then encoded independently by the frozen VAE encoder of FLUX.1 (Black Forest Labs, 2024). We use the posterior mean rather than sampling, yielding a deterministic latent feature grid $z_v \in \mathbb{R}^{g\times g\times c}$ for each view. These features provide compact local descriptors and reduce sensitivity to small stroke displacement and thickness variation while retaining the overall edge-layout and density cues.

3.3.3 SPATIAL GROUNDING AND MULTI-VIEW FUSION

The VAE features lack 3D correspondence. Inspired by Hitem3D 2.0 (He et al., 2026), we associate them with canonical coordinate maps (CCMs). For each view, position and normal maps P_v, N_v rendered from the input shape are foreground-average-pooled to the VAE grid, yielding $\bar{P}_v[i,j]$ and $\bar{N}_v[i,j]$; empty cells are masked. Because these maps come from the input shape, the anchors remain tied to its geometry even when the wireframe is imperfect. We form grounded tokens as

$$f^{(0)}_{v,ij} = \mathrm{Linear}\big([z_v[i,j]; \bar{P}_v[i,j]; \bar{N}_v[i,j]]\big). \tag{2}$$

Here, v indexes the reference view, (i,j) indexes the VAE feature grid, $[\,;\,]$ denotes channel-wise concatenation, and Linear is a learned projection that produces the grounded token $f^{(0)}_{v,ij}$.

We aggregate these tokens with factorized attention. Within each view, 2D RoPE models relationships on the image grid; across views, CCM-parameterized 3D RoPE relates tokens anchored to nearby surface regions. Cells without valid surface coordinates are masked throughout this process.

A final projection produces $T \in \mathbb{R}^{M\times d_o}$. Retaining the image grid preserves local structure and line-density cues. We share encoder weights across views and concatenate their grids for fusion, allowing V to vary without changing learned parameters; only $M = Vg^2$ changes. We evaluate single- and multi-view conditioning in Appendix A.4 (Figure 8).

3.3.4 WIREFRAME CONDITIONING VIA CROSS-ATTENTION

The grounded wireframe tokens are injected into early transformer blocks through an additional cross-attention residual with a zero-initialized output projection. For the input h of a self-attention block, the wireframe branch adds

$$\Delta(h) = \beta W_o \,\mathrm{CrossAttn}(\mathrm{LN}_q(h), \mathrm{LN}_{kv}(W_{kv}T)), \qquad \mathrm{att} \leftarrow \mathrm{att} + \Delta(h), \tag{3}$$

where $\beta \in \{0, 1\}$ is a per-sample validity mask. Zero-initializing W_o makes the new branch contribute zero at the start of fine-tuning, preserving the pretrained surface-conditioned backbone. Here, att denotes output of the backbone's original self-attention sublayer before the residual connection.

Because the surface-conditioning pathway is resumed from a well-trained backbone, the model can initially rely on it as an easy shortcut and under-utilize the newly introduced wireframe branch. To avoid this shortcut, we first use a short wireframe-only warm-up, during which the surface condition is masked, forcing the new branch to learn predictive edge-layout features from the wireframe. We then restore both conditions and apply mutually exclusive condition dropout: at most one of the surface and wireframe conditions is masked for a given sample, so some samples retain both and none lose both.

The interface supports a variable number of reference views. View-wise features are concatenated before injection, so changing the number of views only changes the conditioning length M, without modifying the backbone or fusion parameters. Moreover, because 3D grounding and cross-view reasoning are completed upstream, the backbone queries need not expose 3D coordinates. This makes the interface applicable to different mesh representations. We evaluate variable-view conditioning in Appendix A.4 (Figure 8) and transfer to a flow-matching backbone in Section 4.2.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We train on around 300k 3D assets. Most are selected from Objaverse-XL (Deitke et al., 2022; 2023), complemented by 3D-FUTURE (Fu et al., 2021), ABO (Collins et al., 2022), HSSD (Khanna et al., 2024), Project Aria (Engel et al., 2023) and ShapeNet-Core (Chang et al., 2015); each source is split into training and test portions. For evaluation we use 100 objects from Toys4K (Stojanov et al., 2021) (TOYS100), selected to include both organic and hard-surface shapes; each is generated at four requested face budgets $\{5500, 3000, 1500, 800\}$, giving 400 generations per model. Hereafter, s denotes the requested face budget used to construct the wireframe reference.

Paired supervision. For each training shape, we construct targets at multiple face budgets using quadric-error decimation (Garland & Heckbert, 1997), preceded by a radius weld to remove degenerate mesh fragments. At each iteration, the target is the original mesh with probability 0.4 and otherwise a uniformly sampled valid decimation. The surface point cloud is always sampled from the original mesh, allowing the same geometric condition to be paired with different polygonal targets. Rendering details are provided in Appendix A.1.

Adapting to WirePilot references. The main model is trained with rendered wireframes, which are exactly aligned with the target geometry and consistent across views. WirePilot references may instead contain different stroke statistics and small cross-view inconsistencies. To reduce this gap, we fine-tune the model on around $9{,}000$ training samples paired with WirePilot references screened for approximate alignment with the rendered reference of the same object and budget. This stage exposes the model to deployment-time variations while filtering severely misaligned drawings.

4.2 EVALUATION

To evaluate the full system under deployment-time conditions, we use the 2×3 wireframe grids produced by *WirePilot* in this section unless stated otherwise. In this setting, the desired low-poly edge layout is not known in advance and must be proposed before mesh generation. Unlike wireframes rendered from an existing target mesh, WirePilot references include generation artifacts such as style variation, imperfect stroke width, and occasional over-drawing. The binarization, stroke augmentation, and adaptation stage described in Section 4.1 reduce sensitivity to these variations.

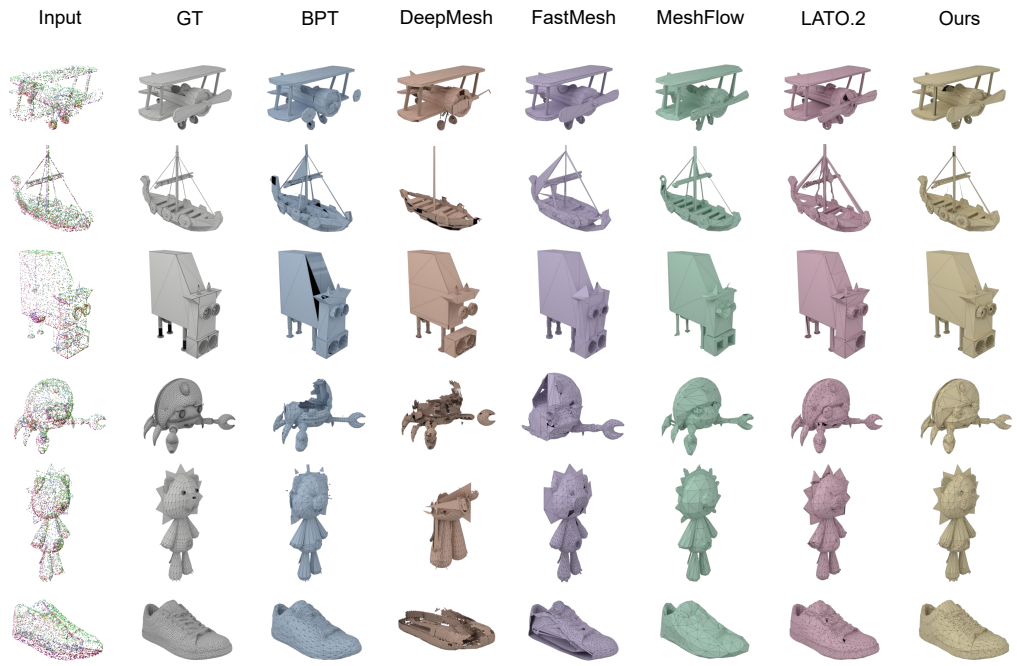


Figure 4: Qualitative comparison with representative native mesh generators.

Geometric fidelity. We evaluate geometric fidelity using Chamfer distance (CD), Hausdorff distance (HD), HD_{95}, and signed and unsigned normal consistency, computed from 30,000 surface samples. Reporting both normal metrics reduces sensitivity to inconsistent face orientation in the ground truth. As shown in Table 1, WireForge at $s = 3000$ improves over BPT in CD, HD, HD_{95}, and signed normal consistency, and slightly improves unsigned normal consistency.

Table 1: **Quantitative comparison on TOYS100.** TOYS100 is our selected subset of Toys4K, which contains both hard-surface and organic objects. All methods use the same evaluation subset and sampling protocol. HD_{95} denotes the 95th-percentile point-to-surface distance.

Model	CD ↓	HD ↓	HD_{95} ↓	NC ↑	|NC| ↑
MeshAnythingV2 (Chen et al., 2025b)	0.0145	0.1191	0.0500	-0.0813	0.8975
BPT (Weng et al., 2025)	0.0070	0.0484	0.0150	0.8716	0.9399
TreeMeshGPT (Lionar et al., 2025)	0.0106	0.1167	0.0471	0.8519	0.9107
DeepMesh (Zhao et al., 2025)	0.0128	0.0988	0.0442	0.8772	0.9417
Mesh-Silksong (Song et al., 2026)	0.0085	0.1036	0.0230	0.8291	0.9192
FastMesh (Kim et al., 2025)	0.0118	0.0632	0.0281	-0.0143	0.9103
MeshFlow (Li et al., 2026)	0.0103	0.0518	0.0215	0.8311	0.9060
LATO.2 (Long et al., 2026)	0.0080	0.0585	0.0193	0.0889	0.9049
Ours, $s = 5500$	0.0069	0.0402	0.0137	0.9076	0.9395
Ours, $s = 3000$	0.0069	0.0404	0.0137	**0.9097**	**0.9426**
Ours, $s = 1500$	**0.0068**	**0.0398**	**0.0126**	0.9037	0.9425
Ours, $s = 800$	0.0072	0.0480	0.0145	0.9052	0.9379

Qualitative comparison. Figure 4 compares WireForge with representative native mesh generators. WireForge preserves the input geometry while allocating sparse edges more coherently around salient contours, part boundaries, and geometric transitions.

Generalization to other backbones. We adapt the conditioning interface to a flow-matching model following the VAE–DiT paradigm used by MeshCraft and MeshFlow (He et al., 2025; Li et al., 2026) and the autoregressive Mesh-Silksong (Song et al., 2026). As shown in Table 2, both integrations improve all five surface metrics while adding structural conditioning; qualitative exam-

Table 2: Transfer to flow-matching and autoregressive backbones. Rows are comparable only within each backbone.

Model	CD ↓	HD ↓	HD_{95} ↓	NC ↑	|NC| ↑
Flow matching (MeshFlow), released	0.0103	0.0518	0.0215	0.8311	0.9060
+ ours	**0.0086**	**0.0460**	**0.0171**	**0.8791**	**0.9330**
Autoregressive (Mesh-Silksong), released	0.0085	0.1036	0.0230	0.8291	0.9192
+ ours	**0.0081**	**0.0751**	**0.0225**	**0.8359**	**0.9243**

ples are shown in Figure 6 (b). Since each integration requires backbone-specific fine-tuning, these results demonstrate compatibility rather than a condition-only effect.

4.3 Applications

The explicit wireframe provides direct control over polygon density and local edge placement, beyond preserving the input surface.

Density control. Because the regular image grid retains line-density cues (Section 3.3.3), the reference conveys a target polygon-density signal without a separate scalar face-count token in the mesh generator. Figure 5 shows that progressively sparser references produce coarser meshes whose layouts are reorganized rather than uniformly decimated.

Edge-layout control. Beyond controlling density, the wireframe reference specifies *where* edges are placed and how they flow across the surface. Natural-language instructions can guide WirePilot to generate or refine the reference (Section 3.2), while sketches can edit the image supplied to WireForge. The generated mesh responds to the edge flow, part boundaries, and feature loops without retraining. Text- and sketch-controlled examples are shown in Appendix A.4 (Figure 7).

4.4 Ablation Studies

Three choices carry the conditioning pathway: keeping the wireframe in the image domain, grounding it in 3D before fusion, and adapting the model to generated rather than rendered references. We ablate each in turn and report both surface and rendering metrics (Table 3).

Metric interpretation and trade-offs. As shown in Table 3, the full model achieves the best CD, HD_{95}, and FID, but does not lead NC, LPIPS, or SSIM. The corresponding gaps are small: 0.0007 in NC, 0.0004 in LPIPS, and 0.0008 in SSIM. This split is expected because the metrics capture different properties. In particular, LPIPS and SSIM reward pixel-level similarity between rendered strokes and the reference, so a variant can match the 2D ink slightly more closely while producing less accurate surface geometry or less robust rendering statistics. We therefore assess the variants jointly across surface metrics, rendering metrics, and qualitative behavior rather than selecting the model by any single column.

Image-space vs. point-based wireframe encoding. Back-projecting a wireframe into a 3D point cloud is intuitive, but it amplifies inference-time imperfections: cross-view disagreement and small stroke offsets become duplicated, displaced, or off-surface points. Compressing the entire cloud into a global shape latent can spread a local error across the condition (Figure 6 (a)). Keeping a regular image grid instead localizes an erroneous stroke to its view and region, leaving other views usable.

For the representation comparison, the unadapted image-space model and the point-based variant reach similar surface accuracy (CD 0.0078 vs. 0.0076; HD_{95} 0.0184 vs. 0.0178), and the point-based variant is marginally better on LPIPS and SSIM. The unadapted image-space model, however, improves normal consistency (0.9073 vs. 0.8692) and FID (31.4 vs. 35.7). Together with the localized robustness shown in Figure 6 (a), these results support retaining the condition on the image grid.

Rendering metrics. Surface metrics measure geometric fidelity but are insensitive to mesh connectivity. We therefore render each output under the reference cameras and report LPIPS, FID, and SSIM against the input wireframes. Because these image metrics also respond to stroke density and

style, we interpret them jointly with surface metrics, face counts, and qualitative results rather than as direct topology measures.

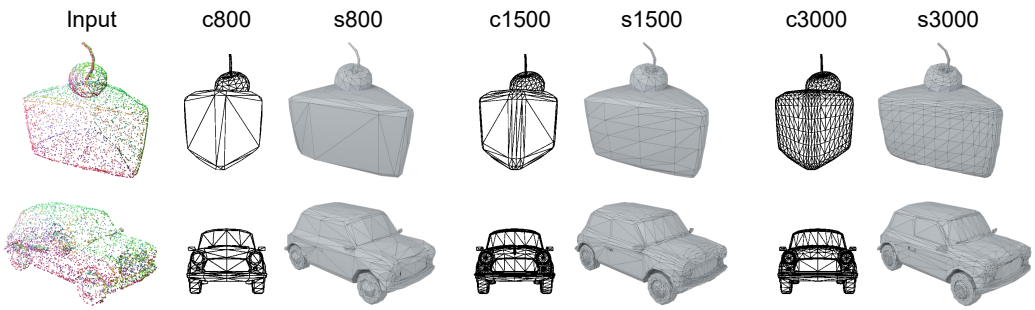
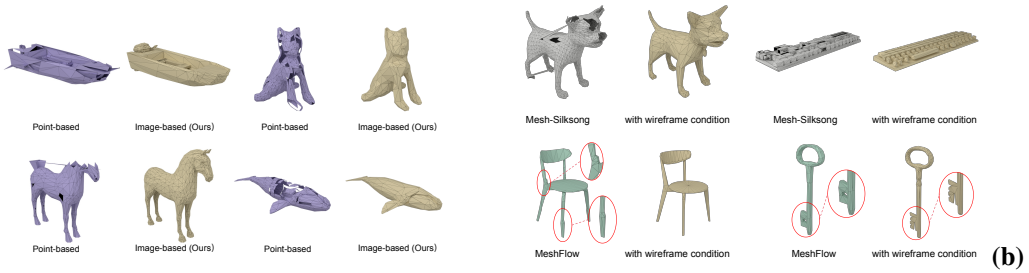


Figure 5: Wireframe conditions with increasing density produce denser generated meshes.



(a) **(b)**

Figure 6: **(a)** Point-based failures caused by noisy back-projection. **(b)** Backbone-specific integrations of our conditioning interface.

Table 3: Ablations using WirePilot references; rendering metrics compare output wireframes with their inputs.

	Surface			Rendering		
Model	CD ↓	HD_{95} ↓	NC ↑	LPIPS ↓	FID ↓	SSIM ↑
Ours (full)	**0.0070**	**0.0136**	0.9066	0.0661	**31.3**	0.9025
w/o adapting	0.0078	0.0184	**0.9073**	0.0672	31.4	0.9027
Point-based condition	0.0076	0.0178	0.8692	**0.0657**	35.7	**0.9033**
w/o CCM + 3D RoPE	0.0092	0.0264	0.8948	0.0733	36.9	0.9015

The full model achieves the best CD, HD_{95}, and FID; the small differences in the remaining columns reflect trade-offs between geometric accuracy, exact reference matching, and robustness to generated wireframes. Detailed analysis is provided in Appendix A.5.

5 Conclusion

We presented WireForge, a framework that introduces editable 2D wireframe priors as explicit structural guidance for low-poly mesh generation. By spatially grounding image-space wireframe features, WireForge complements surface conditioning with direct control over edge layouts and polygon density. Experiments demonstrate improved geometric fidelity, effective structural control, and compatibility across different mesh-generation backbones.

Limitations. WireForge depends on wireframe quality and cross-view consistency. Its autoregressive backbone may terminate early for dense outputs, whereas flow-matching backbones may introduce holes. TOYS100 also excludes highly elongated shapes on which WirePilot is unreliable.

AI USE STATEMENT

Generative AI tools assisted manuscript drafting, organization, and editing. WirePilot uses frozen Gemini 3 Pro Image to generate and refine wireframes and GPT-5.5 to evaluate them; neither model is fine-tuned. The authors reviewed all AI-assisted text and artifacts and take responsibility for the final content of this work.

REFERENCES

Antonio Alliegro, Yawar Siddiqui, Tatiana Tommasi, and Matthias Nießner. PolyDiff: Generating 3d polygonal meshes with diffusion models. *arXiv preprint arXiv:2312.11417*, 2023.

Black Forest Labs. FLUX.1. `https://github.com/black-forest-labs/flux`, 2024. Model weights: `https://huggingface.co/black-forest-labs/FLUX.1-dev`.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. MeshAnything: Artist-created mesh generation with autoregressive transformers. In *The Thirteenth International Conference on Learning Representations (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 (ICCV)*, pp. 13922–13931, 2025b.

Yuguang Chen, Xinhai Liu, Xiangyu Zhu, Yiling Zhu, Zhuo Chen, Dongyu Zhang, and Chunchao Guo. TopGen: Learning structural layouts and cross-fields for quadrilateral mesh generation. *arXiv preprint arXiv:2603.10606*, 2026.

Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F. Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. ABO: Dataset and benchmarks for real-world 3D object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21126–21136, 2022.

Dale Decatur, Jacob Serfaty, Oded Stein, Amir Vaxman, and Rana Hanocka. Look both ways before you cross: Lifting cross fields from 2D visual priors. *arXiv preprint arXiv:2605.26062*, 2026.

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.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-xl: A universe of 10m+ 3d objects. *arXiv preprint arXiv:2307.05663*, 2023.

Jakob Engel, Kiran Somasundaram, Michael Cole, Nathan Wilson, Xiaqing Ma, Tyler Gamino, Andrew Bäckström, et al. Project Aria: A new tool for egocentric multi-modal AI research. *arXiv preprint arXiv:2308.13561*, 2023.

Yifei Feng, Mingxin Yang, Shuhui Yang, Sheng Zhang, Jiaao Yu, Zibo Zhao, Yuhong Liu, Jie Jiang, and Chunchao Guo. RomanTex: Decoupling 3D-aware rotary positional embedded multi-attention network for texture synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17203–17213, 2025.

Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3D-FUTURE: 3D furniture shape with texture. *International Journal of Computer Vision (IJCV)*, 129(12):3313–3337, 2021.

Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 209–216, 1997.

Huiang He, Shengchu Zhao, Jianwen Huang, Jie Li, Jiaqi Wu, Hu Zhang, Pei Tang, Heliang Zheng, Yukun Li, and Rongfei Jia. Hitem3D 2.0: Multi-view guided native 3D texture generation. *arXiv preprint arXiv:2604.09231*, 2026.

Xianglong He, Junyi Chen, Di Huang, Zexiang Liu, Xiaoshui Huang, Wanli Ouyang, Chun Yuan, and Yangguang Li. MeshCraft: Exploring efficient and controllable mesh generation with flow-based DiTs. *arXiv preprint arXiv:2503.23022*, 2025.

Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X. Chang, and Manolis Savva. Habitat Synthetic Scenes Dataset (HSSD-200): An analysis of 3D scene scale and realism tradeoffs for object-goal navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16384–16393, 2024.

Jeonghwan Kim, Yushi Lan, Armando Fortes, Yongwei Chen, and Xingang Pan. FastMesh: Efficient artistic mesh generation via component decoupling. *arXiv preprint arXiv:2508.19188*, 2025.

Weiyu Li, Antoine Toisoul, Tom Monnier, Roman Shapovalov, Rakesh Ranjan, Ping Tan, and Andrea Vedaldi. MeshFlow: Efficient artistic mesh generation via MeshVAE and flow-based diffusion transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5849–5858, 2026.

Stefan Lionar, Jiabin Liang, and Gim Hee Lee. TreeMeshGPT: Artistic mesh generation with autoregressive tree sequencing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26608–26617, 2025.

Chenyu Liu, Hongze Chen, Jingzhi Bao, Lingting Zhu, Runze Zhang, Weikai Chen, Zeyu Hu, Yingda Yin, Keyang Luo, and Xin Wang. CaliTex: Geometry-calibrated attention for view-coherent 3D texture generation. *arXiv preprint arXiv:2511.21309*, 2025.

Hang Long, Tianhao Zhao, Junkai Lin, Youjia Zhang, Huipeng Guo, Rendong Liang, Jiale Xu, Jozef Hladký, Matthias Nießner, Yuanming Hu, and Wei Yang. LATO.2: Factorized 3d mesh generation with vertex and topology flow. *arXiv preprint arXiv:2607.10623*, 2026.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An autoregressive generative model of 3d meshes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 7220–7229, 2020.

OpenAI. Introducing GPT-5.5. OpenAI, April 2026. URL `https://openai.com/index/introducing-gpt-5-5/`. Accessed September 26, 2026.

Naina Raisinghani. Introducing Nano Banana Pro. Google Blog, 2025. URL `https://blog.google/innovation-and-ai/products/nano-banana-pro/`. Gemini 3 Pro Image, released November 20, 2025.

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 (CVPR)*, pp. 19615–19625, 2024.

Gaochao Song, Zibo Zhao, Haohan Weng, Jingbo Zeng, Rongfei Jia, and Shenghua Gao. Topology-preserved auto-regressive mesh generation in the manner of weaving silk. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Stefan Stojanov, Anh Thai, and James M. Rehg. Using shape to categorize: Low-shot learning with an explicit shape bias. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1798–1808, 2021.

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 *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Team Hunyuan3D, Bowen Zhang, Chunchao Guo, Haolin Liu, Hongyu Yan, Huiwen Shi, Jingwei Huang, Junlin Yu, Kunhong Li, Linus, Penghao Wang, Qingxiang Lin, Sicong Liu, Xianghui Yang, Yixuan Tang, Yunfei Zhao, Zeqiang Lai, Zhihao Liang, and Zibo Zhao. Hunyuan3D-Omni: A unified framework for controllable generation of 3D assets. *arXiv preprint arXiv:2509.21245*, 2025.

Hanxiao Wang, Ying-Tian Liu, Yuan-Chen Guo, Qi-Yuan Feng, Zi-Xin Zou, Ding Liang, Biao Zhang, and Yan-Pei Cao. Nexus: Native mesh generation with diffusion. *ACM Transactions on Graphics*, 45(4), 2026. doi: 10.1145/3811344. URL https://doi.org/10.1145/3811344.

Haohan Weng, Yikai Wang, Tong Zhang, C. L. Philip Chen, and Jun Zhu. PivotMesh: Generic 3d mesh generation via pivot vertices guidance. *arXiv preprint arXiv:2405.16890*, 2024.

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 (CVPR)*, pp. 11093–11103, 2025.

Jiale Xu, Rendong Liang, Yuhao Long, Siyuan Shen, Zangyueyang Xian, Xiaofei Wu, Zeyi Xu, and Yuanming Hu. Meshy T2: Fast native mesh generation with flow matching, 2026. URL https://arxiv.org/abs/2607.28675.

Hongyu Yan, Kunming Luo, Weiyu Li, Kaiyi Zhang, Yixun Liang, Jingwei Huang, Chunchao Guo, and Ping Tan. PoseMaster: A unified 3D native framework for stylized pose generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. CLAY: A controllable large-scale generative model for creating high-quality 3D assets. *arXiv preprint arXiv:2406.13897*, 2024.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3836–3847, 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 (ICCV)*, pp. 10612–10623, 2025.

Tianhao Zhao, Youjia Zhang, Hang Long, Jinshen Zhang, Wenbing Li, Yang Yang, Gongbo Zhang, Jozef Hladký, Matthias Nießner, and Wei Yang. LATO: 3d mesh flow matching with structured TOpology preserving LAtents. *arXiv preprint arXiv:2603.06357*, 2026.

A Appendix

A.1 Implementation Details

Backbone and wireframe encoder. Our primary BPT backbone uses a 24-layer pre-norm transformer with hidden width 1024, 16 attention heads, and a $12{,}000$-token context. Its shape encoder maps $10{,}240$ sampled surface points to 512 latent tokens; half of the input points are sampled uniformly and half from dihedral-angle-filtered sharp edges. For the wireframe branch, each view is binarized and normalized to 512^2: rendered training views are downsampled from 1024^2 by max pooling, whereas generated grids are first split into views and resized to the same resolution. During training, stroke dilation of 1–3 pixels is applied with probability 0.5, shared across views of the same object. We use the frozen image VAE of (Black Forest Labs, 2024), whose deterministic latent has $g = 64$ and $c = 16$. The grounding encoder uses two factorized 2D/3D attention stages with hidden width 256, and projects the final six-view grid to $M = 24{,}576$ tokens of width 64. The wireframe residual is attached to the first $L_f = 5$ transformer blocks: injecting into early layers lets the condition shape the representation before it is committed to token predictions, while limiting the number of injected layers bounds the cost of the extra cross-attention over M tokens.

Training renderings. For the selected target, we render its wireframe from six canonical orthographic views at 1024^2. From the corresponding dense input shape, we render the position map used as the CCM and the normal map under the same cameras. The target-specific wireframe associates image-space line layout and density with the corresponding mesh connectivity and polygon budget. We apply the same axis-aligned rotation to the dense input and target meshes so that the wireframe, position maps, and normal maps share a common frame.

Training schedule and condition dropout. Training starts with a two-epoch wireframe-only warm-up, in which the shape latents are dropped for every sample. The surface condition is then restored, and from that point the two conditions are dropped mutually exclusively: a sample may lose one of them, but never both. Every sample therefore keeps at least one condition, while each pathway is regularly exposed on its own, so the wireframe must guide the layout by itself rather than settle into a refinement of an already sufficient surface condition. The initial training stage lasts 17 epochs in total, including the two-epoch condition warm-up, and uses AdamW, using a learning rate of 10^{-4} after a one-epoch linear warm-up from 10^{-6} and cosine annealing to 10^{-5}, weight decay 0.01, gradient clipping at 1.0, and bf16 precision on 8 H100 GPUs. The final checkpoint comes from a four-epoch continuation at a constant 10^{-5}, once the cosine schedule has bottomed out. At inference we decode with temperature 0.5, top-$k = 50$ and top-$p = 0.95$ under a fixed seed, using the six canonical views unless stated otherwise.

A.2 WirePilot Details

Models. WirePilot uses Gemini 3 Pro Image (Raisinghani, 2025) as a frozen image generator, queried at 2K resolution with a 3:2 aspect ratio matching the 2×3 normal-map grid. GPT-5.5 (OpenAI, 2026) serves as the structural evaluator. Given the normal maps, candidate wireframe, geometric-check report, and optional user prompt, it returns scores, veto flags, and revision instructions in a fixed JSON schema. Neither model is fine-tuned.

Geometric check. For each view, we extract the normal-map silhouette and the wireframe-ink mask. We measure *coverage*, the fraction of the silhouette overlapped by the dilated wireframe, and the *out-of-bounds ratio*, the fraction of wireframe ink outside the dilated silhouette. A 6×6 cross-view IoU matrix detects view permutations. A candidate is rejected before structural evaluation if its average coverage is below 0.75 (or any view is below 0.65), its average out-of-bounds ratio exceeds 0.05 (or any view exceeds 0.15), or views are swapped. Rejected candidates directly receive an alignment-only refinement instruction.

Scoring rubric. The evaluator scores six dimensions on integer 0–5 scales with written anchors (Table 4); their weighted sum is rescaled to $[0, 100]$. Hard vetoes include missing or incorrect silhouettes, dense edge-map degeneration, missing parts, inconsistent views, off-object content, and

Table 4: WirePilot evaluation rubric. Each dimension is scored from 0 to 5 against written anchors.

Dimension	Criterion	Weight
Normal alignment	silhouette and feature lines match the normal maps	0.25
Edge flow	ordered loops that follow the form; no scattered triangles	0.20
Cross-view consistency	edges agree across the six views	0.20
Density allocation	line density follows curvature and detail; major loops close	0.15
Style	clean, artist-designed triangulated wireframe	0.12
User-prompt alignment	explicit user requirements (5 if none are given)	0.08

non-binary rendering. Soft vetoes cover heavy line noise, quad-dominant layouts, and violations of user-specified styles. Hard and soft vetoes cap the overall score at 55 and 70, respectively.

Stopping rule. Acceptance is computed deterministically from the evaluator output. A candidate is accepted if it triggers no hard veto, reaches an overall score of at least 68, and scores at least 3 on normal alignment, edge flow, and cross-view consistency. Otherwise, the previous wireframe and revision instructions are fed back to the generator. In practice, most shapes are accepted after one or two refinements.

A.3 Topology Quality

Surface metrics do not capture mesh connectivity, so Table 5 reports two edge-based statistics on a shared 60-object subset of TOYS100. The boundary edge ratio (BER) is the fraction of unique undirected edges incident to exactly one face, while the non-manifold ratio is the fraction incident to three or more faces. All meshes undergo the same repair pass before measurement. The two metrics must be interpreted jointly: a low BER alone does not imply valid topology, because surplus or overlapping faces can remove open boundaries while creating non-manifold edges. FastMesh illustrates this failure mode with a BER of 0.0033 but a non-manifold ratio of 0.7406 and 5.75 faces per vertex, nearly three times the value of about 2 expected for a closed triangle mesh; LATO.2 (3.05) and MeshFlow (2.15) show the same pattern to a lesser degree. Across all requested density levels, WireForge improves both statistics over BPT and MeshAnythingV2, reduces the non-manifold ratio of Mesh-Silksong by roughly five to six times, and achieves BER values close to the ground-truth reference while remaining comparable to DeepMesh in non-manifold ratio.

Table 5: **Topology quality on a shared 60-object subset of TOYS100.** BER is the fraction of edges incident to exactly one face; the non-manifold ratio is the fraction incident to three or more. All meshes, including the baselines, were passed through the same repair pass before measurement. The two metrics should be interpreted jointly; the ground truth is a reference row, not a competitor. F/V is the mean face-to-vertex ratio after vertex welding, which is close to 2 for a closed triangle mesh; values well above 2 indicate surplus faces.

Model	BER ↓	Non-manifold ↓	F/V
Ground truth (reference)	*0.0097*	*0.0007*	*1.96*
MeshAnythingV2	0.0631	0.0258	1.98
BPT	0.0163	0.0092	1.96
DeepMesh	0.0152	0.0062	1.96
Mesh-Silksong	0.0094	0.0345	2.04
FastMesh	**0.0033**	0.7406	5.75
MeshFlow	0.0070	0.1180	2.15
LATO.2	0.0206	0.3250	3.05
Ours, $s = 5500$	0.0091	0.0059	1.98
Ours, $s = 3000$	0.0106	0.0072	1.98
Ours, $s = 1500$	0.0102	**0.0055**	1.96
Ours, $s = 800$	0.0108	0.0064	1.95

A.4 Additional Applications

Text and sketch control. Figure 7 demonstrates two editing paths at a fixed requested polygon budget. Natural-language instructions guide WirePilot to generate or refine the reference, while sketches directly modify the wireframe supplied to WireForge. The resulting meshes respond to these spatial edits without retraining the mesh generator.

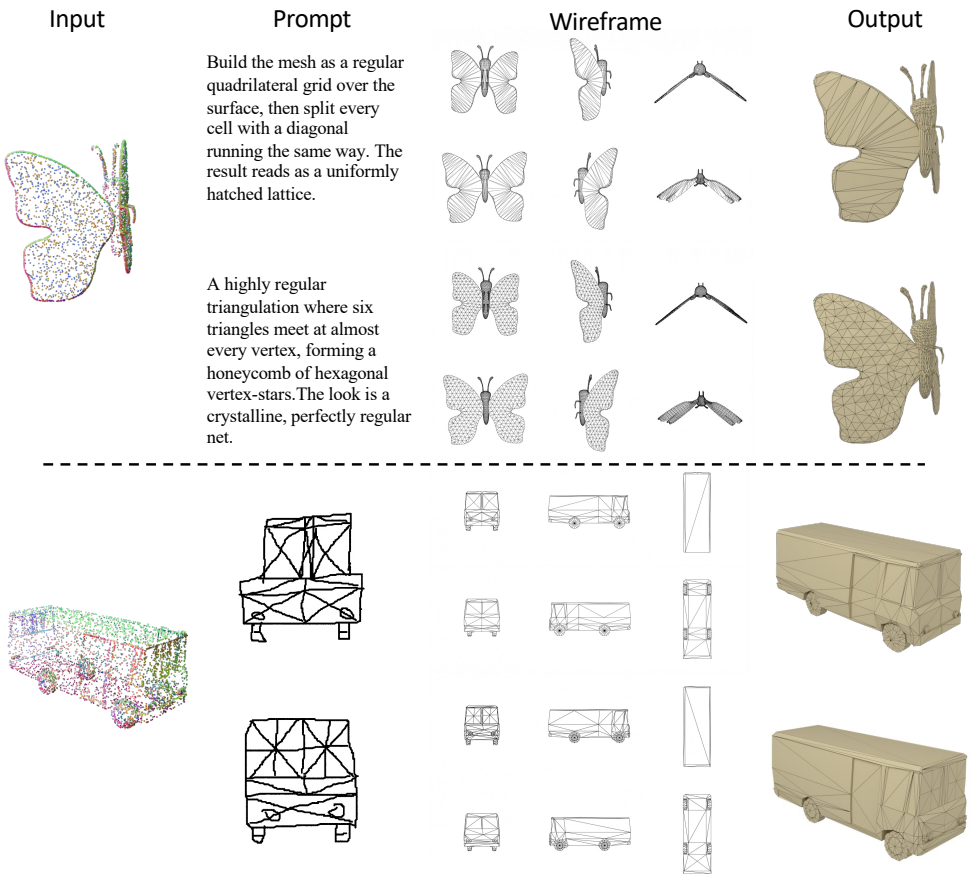


Figure 7: At a fixed requested polygon budget, text instructions guide WirePilot generation, while sketches directly edit the wireframe reference.

Single- and multi-view conditioning. The same checkpoint supports either a single reference view or a multi-view reference without retraining, since changing the number of views only changes the conditioning length. A single view provides a lightweight interface for controlling the visible region, but offers no direct guidance for edge placement on the remaining sides, whose layouts therefore tend to revert toward the mesh backbone's prior. Six-view conditioning covers the full object and provides more complete supervision for organizing edges across both visible and previously unobserved regions. Figure 8 illustrates this distinction: both settings follow the supplied front-view layout, while the six-view input also guides the wiring on the other sides of the object.

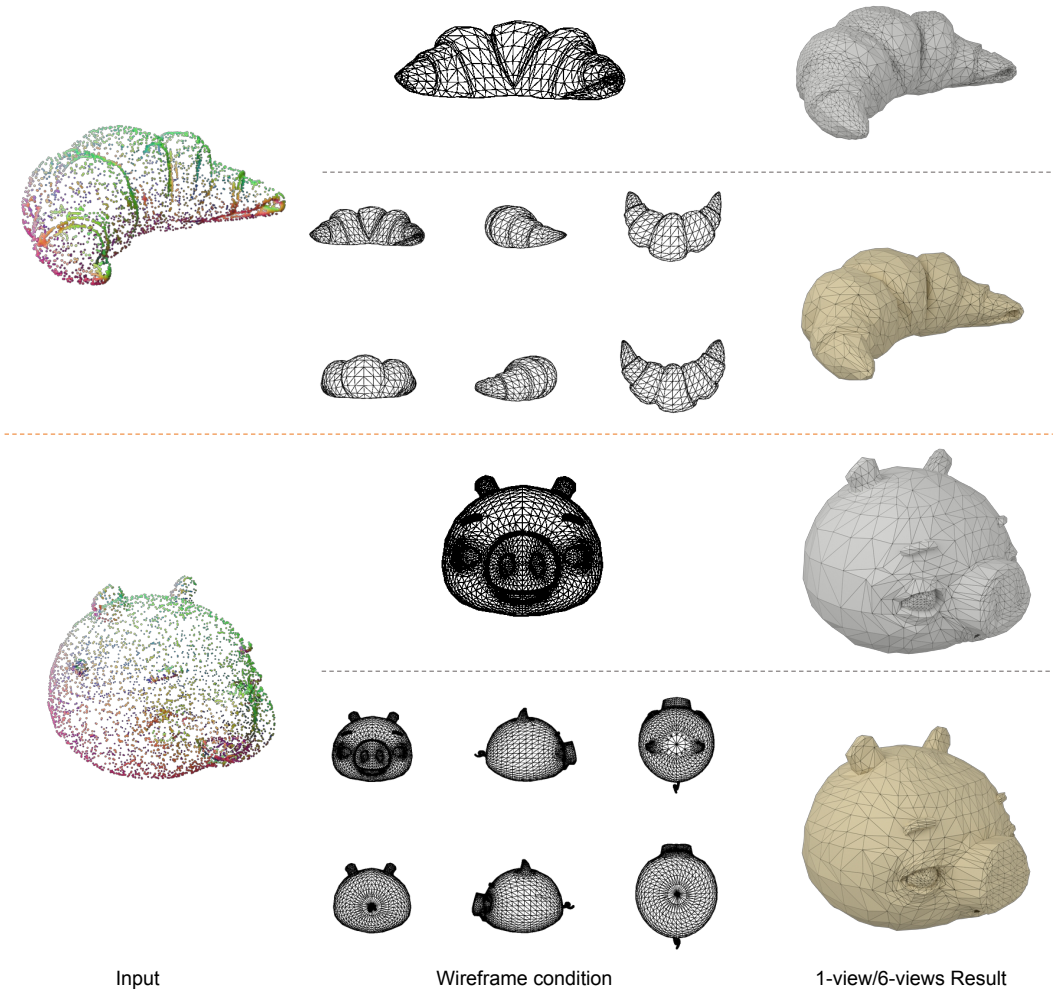


Figure 8: Single- versus six-view conditioning from the same checkpoint. Six-view conditioning additionally guides edge layouts beyond the view covered by the single reference.

A.5 Additional Ablation Results

Spatial grounding. The combined ablation in Table 3 removes the CCM and normal channels of equation 2 together with cross-view 3D RoPE. It performs worse than the full model on all reported surface and rendering metrics, consistent with the role of shared 3D coordinates in relating views of the same surface. Because this ablation removes several components at once, it supports the grounding suite as a whole rather than isolating the contribution of any individual component.

Adaptation to generated references. Training uses geometrically aligned rendered wireframes, whereas inference uses WirePilot references generated by Gemini 3 Pro Image (Nano Banana Pro). Even after screening, these generated references can contain slight cross-view 3D inconsistencies, such as small stroke displacements or locally incompatible edge placements. Fine-tuning on the screened references described in Section 4.1 therefore introduces a deliberate trade-off: it improves robustness to imperfect deployment-time wireframes while sacrificing a very small amount of exact edge-layout consistency. Relative to the unadapted model, adaptation improves CD ($0.0078 \rightarrow 0.0070$), HD_{95} ($0.0184 \rightarrow 0.0136$), LPIPS ($0.0672 \rightarrow 0.0661$), and FID ($31.4 \rightarrow 31.3$), while NC ($0.9073 \rightarrow 0.9066$) and SSIM ($0.9027 \rightarrow 0.9025$) decrease only slightly. Figure 9 illustrates the intended robustness gain: under the same generated references, the adapted model produces more coherent surfaces while retaining the requested structural guidance.

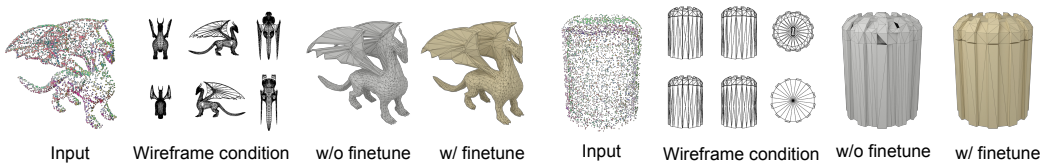


Figure 9: Adaptation to WirePilot references. The adapted model produces more coherent surfaces under the same input conditions.

A.6 User Study

We conducted a user study with 30 participants on 11 test samples to evaluate the perceptual quality of the generated meshes. For each comparison, participants were presented with results from six methods, whose spatial positions were randomly shuffled for each sample, and were asked to identify the highest-quality result and the second-highest-quality result, corresponding to the Top-1 and Top-2 selections, respectively. As shown in Table 6, our method received 56.9% of the Top-1 selections, substantially higher than all competing methods, which received 12.5%, 9.8%, 8.8%, 7.1%, and 5.1%, respectively. For the Top-2 evaluation, our method achieved 82.8%, while the competing methods obtained 33.3%, 31.6%, 21.5%, 16.5%, and 14.1%, respectively. Overall, the results demonstrate a clear preference for the meshes generated by our method in the user study.

Table 6: User study results. We report the percentage of comparisons in which each method was selected as the highest-quality result (Top-1) or as one of the two highest-quality results (Top-2).

Method	Top-1 (%) ↑	Top-2 (%) ↑
Ours	**56.9**	**82.8**
LATO.2	12.5	31.6
DeepMesh	9.8	16.5
FastMesh	8.8	21.5
MeshFlow	7.1	33.3
BPT	5.1	14.1