DELTAFLOW: SINGLE-STAGE MESH GENERATION USING FLOW MATCHING WITH LAPLACIAN POSITIONAL ENCODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Triangle meshes remain the standard 3D representation for graphics and rendering. However, current explicit triangle mesh generation methods either rely on slow autoregressive methods that generate one triangle at a time or require a multi-stage pipeline with diffusion models that hinders optimization for downstream objectives. To fill this gap, we introduce DeltaFlow, a single-stage flow matching framework for unordered triangle soups using a Diffusion Transformer (DiT) architecture. Naive triangle soup generation uses unordered tokens to represent the generated triangles, leading to meshes with duplicated or missing triangles and poor connectivity. Instead, DeltaFlow introduces Laplacian positional encoding that encodes a mesh's dual-graph Laplacian coordinates onto a shared template for rotary position embeddings (RoPE), injecting a connectivity prior into the denoising process. Experiments show that DeltaFlow generates meshes of similar quality to those generated by other mesh generative methods and substantially tighter meshes than those generated by other flow-based methods. Furthermore, our native triangle flow matching model learns a generalizable prior that enables various downstream geometry processing applications, such as mesh inpainting and point-cloud-conditioned mesh generation. Code and models will be released.

1 INTRODUCTION

Triangle mesh is the standard 3D representation in computer graphics, supporting rendering (Pharr et al., 2016), UV texturing (Lévy et al., 2023), geometric processing (Crane et al., 2013), and animation (Sumner & Popović, 2004). Its utility depends not only on geometric fidelity but also on topology: artists carefully design face connectivity to enable reliable texturing, deformation, and simulation. This connectivity describes an intrinsic structure that remains unchanged when the mesh is rotated in space. Learning to generate meshes should therefore capture how faces fit together across orientations, in addition to their geometric arrangement. Our goal is to make this structure available to the generative model, supporting coherent connectivity and geometry processing applications such as mesh completion and upsampling.

However, learning to generate such meshes with such properties remains challenging. Methods based on implicit fields (Mescheder et al., 2019; Park et al., 2019) recover meshes through isosurfacing, which offers little control over connectivity, often produces over-tessellated surfaces, and does not preserve the topology of the source mesh. Moreover, SDF-based methods typically require watertight meshes during training to define consistent inside–outside signs, restricting the available training data. SpaceMesh (Shen et al., 2024) and TriFlow (Li et al., 2026a) instead introduce continuous representations of connectivity but rely on specialized representations and additional connectivity decoding or mesh extraction procedures. Autoregressive methods learn explicit topology directly, yet must serialize meshes into long sequences, resulting in slow token-by-token inference and error accumulation. This serialization assumes a face ordering, which may not generalize well.

More recently, diffusion-based methods have adopted multi-stage pipelines for mesh generation. Methods such as Wang et al. (2026); Li et al. (2026b) first generate intermediate representations, such as mesh vertices or latent tokens, and then recover the final mesh through a separate connectivity-generation or decoding stage. Such multi-stage approach complicates the use of the trained model as a general generative prior for inverse problem applications like mesh inpainting and posterior guidance: constraints on the final mesh such as the given triangle faces must be translated into intermediate representations that diffusion operates on. This motivates a continuous gen-

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.

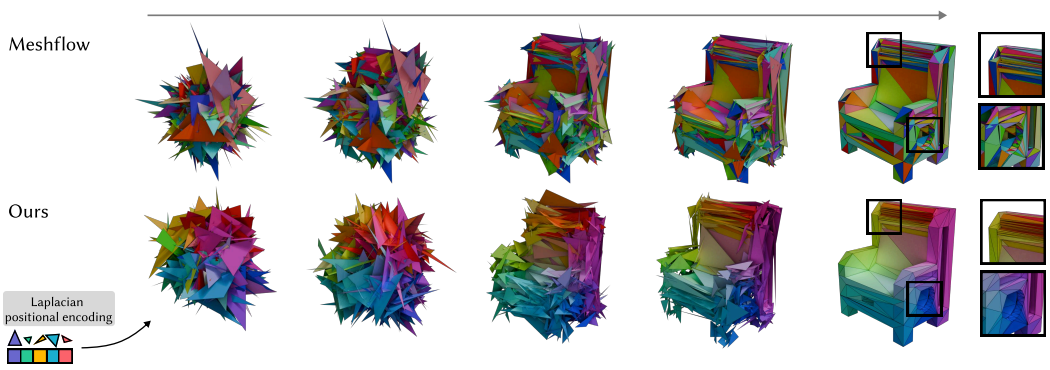

Figure 1: We introduce **DeltaFlow**, a single-stage mesh generator whose Laplacian positional embedding provides local structure to face attention. Colors visualize the positional embeddings, highlighting coherent face neighborhoods. Compared with MeshFlow (top), our model (bottom) produces meshes with fewer missing or duplicate triangles and better connectivity.

erative process directly over triangle coordinates, where the underlying losses can directly constrain sampling. MeshFlow (Sun et al., 2026) takes a step in this direction by applying flow matching directly to unordered triangle soups. While this approach is fast, meshes generated by the model often contain duplicated or missing faces, as well as nearby triangles with poor connectivity, which necessitate a second-stage mesh fixer to improve tightness.

These limitations motivate us to develop a native mesh diffusion model that can generate tight meshes. To this end, we introduce **DeltaFlow**, a single-stage mesh-generation framework based on flow matching with Laplacian positional encoding. DeltaFlow introduces a new positional encoding that captures intrinsic triangle connectivity during training and provides a connectivity prior during generation. We first map each training mesh into dual-graph spectral coordinates and then assign each triangle to a unique position on a template grid shared by meshes with the same number of faces. The assigned grid coordinates are used as positions for RoPE (Su et al., 2024) in the face-attention layers, giving relative positional signals informed by mesh connectivity. Our main contributions are:

- A Laplacian positional encoding that maps mesh connectivity to shared template, providing a neighborhood prior for native triangle-mesh generation. We incorporate these embeddings through RoPE and complement them with vertex weighting and vertex-attention.
- Strong mesh generation results on ShapeNet, with competitive generation quality across four categories, 4–16$\times$ lower tightness error than MeshFlow, and faster generation than autoregressive baselines.
- DeltaFlow learns a generalizable prior that enables training-free mesh inpainting and point-cloud-conditioned generation through direct constraints on triangles during sampling.

2 Related Work

Mesh generation through intermediate representations. A common approach to 3D generation is to model a representation that is easier to learn than an explicit mesh and subsequently reconstruct its surface. PointFlow (Yang et al., 2019) models point distributions with continuous normalizing flows, while XCube (Ren et al., 2024) generates sparse voxel hierarchies with latent diffusion. Neural implicit methods, including Occupancy Networks (Mescheder et al., 2019) and DeepSDF (Park et al., 2019), represent surfaces as level sets of learned occupancy or signed distance functions (SDFs). More recent methods scale this paradigm through latent generative models: CLAY (Zhang et al., 2024) and Direct3D (Wu et al., 2024) generate latent representations decoded into neural fields, while TRELLIS Xiang et al. (2025) and SparseFlex He et al. (2025b) use structured spatial latents from which meshes are reconstructed. All of these representations support detailed geometry, but their conversion to meshes does not retain the face connectivity of the training assets. SDF supervision also typically requires watertight surfaces to obtain consistent inside–outside labels. In

contrast, DeltaFlow learns directly from explicit meshes and generates their triangles without an intermediate surface-reconstruction stage.

Autoregressive mesh generation. Autoregressive methods learn mesh by converting meshes into token sequences. PolyGen (Nash et al., 2020) first generates vertices and then predicts faces conditioned on those vertices. MeshGPT (Siddiqui et al., 2024) learns a vocabulary of quantized triangle embeddings with a graph-based autoencoder and generates their indices with a decoder-only Transformer. MeshXL (Chen et al., 2024) instead models discrete coordinate sequences directly, avoiding a learned mesh tokenizer. Subsequent methods improve serialization efficiency through adjacency-aware traversal, as in MeshAnything v2 (Chen et al., 2025) and EdgeRunner Tang et al. (2025), or compression of coordinates and face patches, as in BPT Weng et al. (2025). MeshSilksong Song et al. (2026) further incorporates topological structure through vertex layering and ordering. Despite these advances, autoregressive decoding still requires sequential token prediction, limiting inference parallelism and allowing errors to accumulate. DeltaFlow instead updates all triangles in parallel with flow matching, without imposing autoregressive dependencies between faces.

Diffusion and flow-based mesh generation. Diffusion and flow models offer parallel alternatives to mesh serialization, with different strategies for representing connectivity. SpaceMesh (Shen et al., 2024) combines vertex diffusion with continuous embeddings from which half-edge connectivity is recovered. TriFlow (Li et al., 2026a) models topology through nearest-vertex vector fields conditioned on input geometry, using latent flow matching followed by topology extraction. MeshCraft (He et al., 2025a) and MeshFlow (Li et al., 2026b) generate continuous mesh latents and recover meshes through learned decoders. Nexus (Wang et al., 2026) instead separates vertex generation from conditional topology generation. These approaches avoid token-by-token inference, but retain a separate connectivity prediction, decoding, or extraction step. Direct triangle-soup models remove the need for a learned mesh representation. PolyDiff (Alliegro et al., 2023) applies categorical diffusion to quantized triangle coordinates, whereas MeshFlow (Sun et al., 2026) operates in continuous space using a permutation-equivariant Transformer and symmetry-aware optimal-transport coupling. However, coordinate quantization limits spatial precision, and continuous triangle-soup generation does not by itself ensure that neighboring faces meet. MeshFlow therefore uses a learned denoiser and vertex merging to improve the connectivity of its outputs. In contrast, DeltaFlow generates continuous triangle soups without a separate learned repair network. Laplacian positional embeddings, incorporated through RoPE, provide a connectivity prior without requiring target adjacency at inference.

Mesh inpainting and generative priors. Mesh inpainting reconstructs missing geometry from partial observations. Classical methods extend the surrounding surface (Liepa, 2003; Verdera et al., 2003) or reuse geometric patches (Sharf et al., 2004). A learned generative prior instead allows completion by sampling conditioned on the observed geometry. For inverse imaging problems, diffusion posterior sampling (DPS) (Chung et al., 2022) guides pretrained diffusion models using measurement-consistency gradients. FlowDPS (Kim et al., 2025) and FlowChef (Patel et al., 2025) develop related guidance strategies for flow models. Garment Particles (Nakayama et al., 2026) adapts FlowDPS for garment editing and point-cloud-conditioned generation in a joint 2D–3D particle representation. The single-stage formulation of DeltaFlow allows fixed-face constraints and geometric losses to act directly on triangle coordinates during sampling, enabling mesh inpainting and point-cloud-conditioned generation without retraining.

3 METHOD

Our goal is to generate meshes with coherent connectivity using flow matching (Lipman et al., 2022; 2024). Prior work (Sun et al., 2026) parameterizes the underlying vector field with permutation-equivariant diffusion transformer (DiT) (Peebles & Xie, 2023), but often produces non-tight meshes with duplicate triangles, thus relying on a second-stage network to fix connectivity. To address these issues, we propose DeltaFlow (Figure 2), a single-stage flow-matching DiT with Laplacian positional embeddings. The DiT incorporates connectivity conditioning through positional embeddings that provide face attention with an approximate neighborhood prior. Global vertex attention further supports alignment of vertices across triangles.

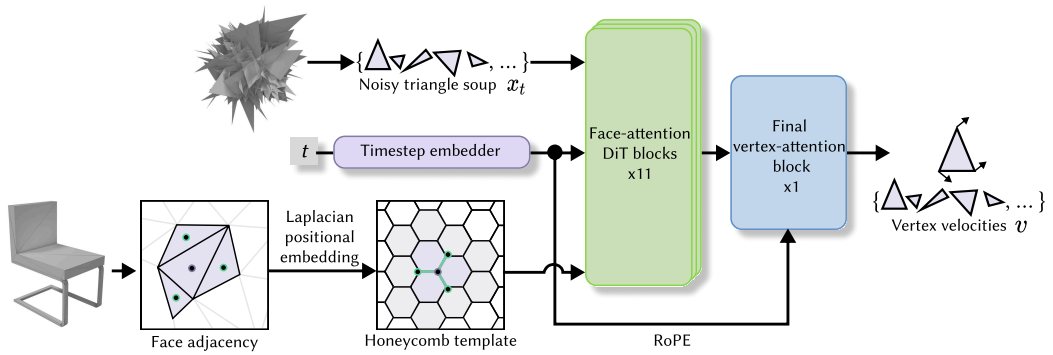

Figure 2: **Overview of DeltaFlow.** Training assigns each triangle to a template position using dual-graph Laplacian coordinates; the DiT predicts vertex directions conditioned on those positions. Sampling starts from noise triangles and the same template. The highlighted dots illustrate a face and its three edge-adjacent neighbors, and a corresponding four-node neighborhood on the template. Eleven face-attention blocks use RoPE, followed by one global vertex-attention block.

3.1 Preliminaries

We represent each mesh in a dataset of K meshes as a *triangle soup*: an unordered collection of F triangles encoded by $x_1 \in \mathbb{R}^{F\times 9}$, where each row x_1^i contains the three vertex coordinates of face i. To generate samples from the unknown data distribution p_1, we use flow matching to learn a velocity field $u_\theta(x,t)$, $t \in [0,1]$, that transports samples from a known noise distribution p_0 to p_1. We choose $p_0 = \mathcal{N}(0, I_{9F})$, which we can sample directly, and evolve faces through

$$\frac{dx_t}{dt} = u_\theta(x_t, t), \qquad x_0 \sim p_0. \tag{1}$$

During training, we pair a target tensor x_1 with independently sampled noise $x_0 \sim p_0$ and form the linear interpolation $x_t = (1-t)x_0 + tx_1$, whose velocity is $\frac{dx_t}{dt} = x_1 - x_0$. We train the field to predict this velocity by minimizing

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\rho,\, x_0\sim p_0,\, x_1\sim p_1} \left\| u_\theta(x_t,t) - (x_1 - x_0) \right\|_{\mathrm{F}}^2, \tag{2}$$

where ρ is the distribution used to sample training times and $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. The number of faces F remains fixed along each trajectory.

Sun et al. (2026) parameterize u_θ with EquiDiT, a diffusion Transformer equivariant to permutations of faces and of vertices within each face. For any face permutation $\pi \in S_F$, this means

$$u_\theta(\pi \cdot x, t) = \pi \cdot u_\theta(x, t), \tag{3}$$

where π permutes the rows of x. The same identity holds when π instead permutes the vertices within each face. EquiDiT implements face attention by averaging the three vertex features of each triangle into a face token, applying self-attention across face tokens without positional embeddings, and adding each output back to its triangle's vertex features. This preserves both face- and vertex-permutation equivariance. We modify this mechanism in Sec. 3.4. This equivariance may impose bounds on relative face motion, as we show next.

3.2 Equivariant flows constrain relative face motion

The vector field u_θ transports noisy face coordinates x_0^i and x_0^j toward their target coordinates x_1^i and x_1^j, which may require initially nearby triangles to move far apart. The non-tight meshes generated by MeshFlow (Sun et al., 2026) motivate examining how equivariance constrains face separation $\|x_t^i - x_t^j\|_2$. The following bound makes this constraint precise.

Proposition 1. *Let $u_\theta(\cdot, t)$ be permutation equivariant and L-Lipschitz in the Frobenius norm for every $t \in [0,1]$. Along any trajectory x_t of its flow, the coordinate vectors of two faces $i \neq j$ satisfy*

$$e^{-Lt}\|x_0^i - x_0^j\|_2 \le \|x_t^i - x_t^j\|_2 \le e^{Lt}\|x_0^i - x_0^j\|_2, \qquad t \in [0,1]. \tag{4}$$

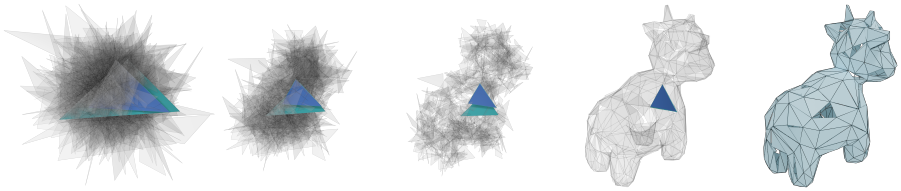

Figure 3: **Locality in equivariant flows.** Illustration of a noise-to-mesh trajectory (left to right). Equivariant flows exhibit strong locality on initial noise. This can lead to collisions and gaps. Two initially nearby faces, highlighted in cyan and blue, evolve toward the same region of the final mesh.

The bound limits how quickly faces can separate or approach one another (proof in Appendix A.1). These constraints can lead to collisions and gaps when initial face distances differ substantially from those required by the target mesh (Fig. 3). Our model breaks these permutation symmetries through fixed face positional embeddings and vertex-slot-specific weights (Secs. 3.3 and 3.4), so the equivariance-based bound no longer applies.

3.3 Laplacian positional encoding

To break permutation-equivariance, we condition u_θ with the face positional embedding $c_i \in \mathbb{R}^2$:

$$u_\theta(x_t, t; c) = \mathrm{DiT}_\theta(x_t, t, c), \qquad c = (c_1, \ldots, c_F). \tag{5}$$

During training, we use the target mesh's connectivity to assign an embedding to each face of x_1. The embedding remain fixed along each trajectory, providing persistent face identities and an approximate neighborhood prior. Permuting faces while holding c fixed changes their embedding assignments, breaking the symmetry behind Prop. 1. Different embeddings allow different velocities for faces with identical coordinates. Jointly permuting faces and embeddings gives $u_\theta(\pi \cdot x, t; \pi \cdot c) = \pi \cdot u_\theta(x, t; c)$.

A naive way to assign positional embeddings is to sort faces lexicographically by their Euclidean coordinates. This ties face identities to the coordinate frame: rotating a mesh can change the ordering without changing how its faces connect. A model that learns how faces fit together should be able to use this structure across orientations. Yet lexicographic embeddings provide inconsistent neighborhood information under rotation, and produce less tight meshes when training orientations vary (Fig. 4). This motivates embeddings based on mesh connectivity, which is unchanged by rotation.

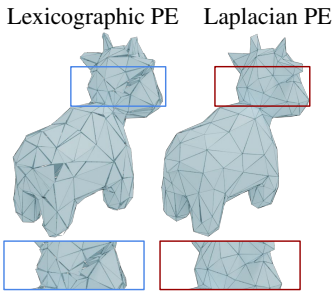

Figure 4: Lexicographic PE produces less tight meshes than Laplacian PE when target orientations vary.

Beyond distinguishing faces, we want positional embeddings to convey each mesh's neighborhood relationships. Arbitrary labels can break permutation symmetry but provide no information about which faces connect. We therefore use a low-frequency Laplacian embedding of the mesh's dual graph, whose smoothness across graph edges provides a continuous representation of face neighborhoods. We then map this mesh-specific embedding to a shared template, making the embeddings available during generation without requiring the target connectivity (Fig. 5).

Laplacian embedding. We construct the dual graph $\mathcal{G}$ of each training mesh, with one node per face and edges between faces that share an edge. Low-frequency Laplacian eigenvectors give coordinates that minimize variation across adjacent faces after degree normalization, subject to orthonormality constraints (Von Luxburg, 2007; Zhang et al., 2010). Let A be the adjacency matrix of $\mathcal{G}$. For disconnected meshes, we add weak links between spatially nearby faces before computing the embedding. We form the normalized graph Laplacian $L_{\mathcal{G}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where D is the weighted degree matrix of A. Let $\phi_0, \ldots, \phi_{F-1} \in \mathbb{R}^F$ be an orthonormal eigenbasis of $L_{\mathcal{G}}$, ordered by nondecreasing eigenvalue. We discard ϕ_0 and use the next two eigenvectors to embed each face as

$$z_i = \big(\phi_1(i), \phi_2(i)\big) \in \mathbb{R}^2 \tag{6}$$

where $\phi_k(i)$ is the entry for face i.

A shared positional template. The spectral coordinates in Eq. 6 are specific to each training mesh and unavailable when sampling from noise. We therefore map them to a common 2D template for

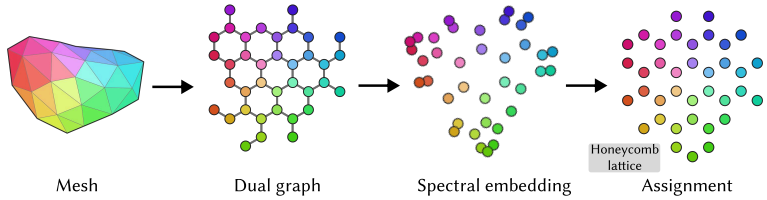

Figure 5: **Laplacian positional encoding.** The training mesh's dual graph is embedded in spectral coordinates z_i (Eq. 6). These are matched to the shared template T_F at right (Eq. 7) by the assignment in Eq. 8, giving each face a positional embedding.

each face count F. A planar honeycomb lattice (Fig. 5, right) provides a domain analogous to a local surface parameterization, with three neighbors per interior point, matching the dual degree of a closed manifold triangle mesh. A fixed enumeration of these points defines

$$T_F = \{c_1, \ldots, c_F\} \subset \mathbb{R}^2, \qquad |T_F| = F. \tag{7}$$

Every mesh with F faces uses these same embeddings, one per face, only their assignment varies.

Assignment to the template. We match the spectral coordinates in Eq. 6 to the template embeddings in Eq. 7. We center and scale each set to unit root-mean-square radius, denoting the results by $\widehat{z}_i$ and $\widehat{c}_j$. We assign faces bijectively to template positions by minimizing the distance between normalized spectral coordinates and template embeddings:

$$(R^\star, \sigma^\star) \in \operatorname*{arg\,min}_{R \in \mathcal{R},\, \sigma \in S_F} \sum_{i=1}^{F} \left\| R\widehat{z}_i - \widehat{c}_{\sigma(i)} \right\|_2^2, \tag{8}$$

where $\mathcal{R}$ accounts for eigenvector sign flips and axis swaps, and σ is a permutation assigning faces to template positions. We reorder each training mesh so that face i carries embedding c_i. At sampling time, the same embeddings are paired with noise triangles x_0^i, requiring only the face count F. These fixed embeddings distinguish faces and provide an approximate neighborhood prior. We incorporate them into the velocity field through attention using RoPE (Su et al., 2024).

3.4 DeltaFlow architecture

DeltaFlow modifies EquiDiT's face attention with vertex-slot-specific weights and connectivity-informed RoPE, breaking vertex and face permutation symmetries, respectively. We represent the input triangle soup $x_t \in \mathbb{R}^{F\times 9}$ by d-dimensional vertex features $H \in \mathbb{R}^{F\times 3\times d}$, obtained through a sinusoidal coordinate encoding and a shared MLP (Sun et al., 2026).

Vertex weighting. We replace uniform averaging of vertex features with learned channelwise weights for each vertex slot. For vertex features $h_{i,v} \in \mathbb{R}^d$ of face i, we compute

$$f_i = \sum_{v=1}^{3} a_v \odot h_{i,v}, \qquad h'_{i,v} = h_{i,v} + b_v \odot \Delta f_i, \tag{9}$$

where $a_v, b_v \in \mathbb{R}^d$ are learned weights shared across faces, $\odot$ denotes elementwise multiplication, and Δf_i is the face-attention update. Slot-specific weights break within-face permutation equivariance, removing the analogous restriction on relative vertex motion identified in Sec. 3.2. A shared residual per-vertex feed-forward network (FFN) follows the update.

Vertex attention. Face attention exchanges information through aggregated face tokens. To also support alignment at the vertex level, a final global vertex-attention block lets vertices interact directly across faces. The block is followed by a residual FFN. A shared output layer predicts a 3D velocity per vertex. Time and face-count embeddings modulate all blocks and the output layer.

4 Experiments

Dataset. We evaluate our method against open-source, state-of-the-art mesh generation methods on four ShapeNet (Chang et al., 2015) categories: chair, table, bench, and lamp, using the preprocessed data released by Sun et al. (2026).

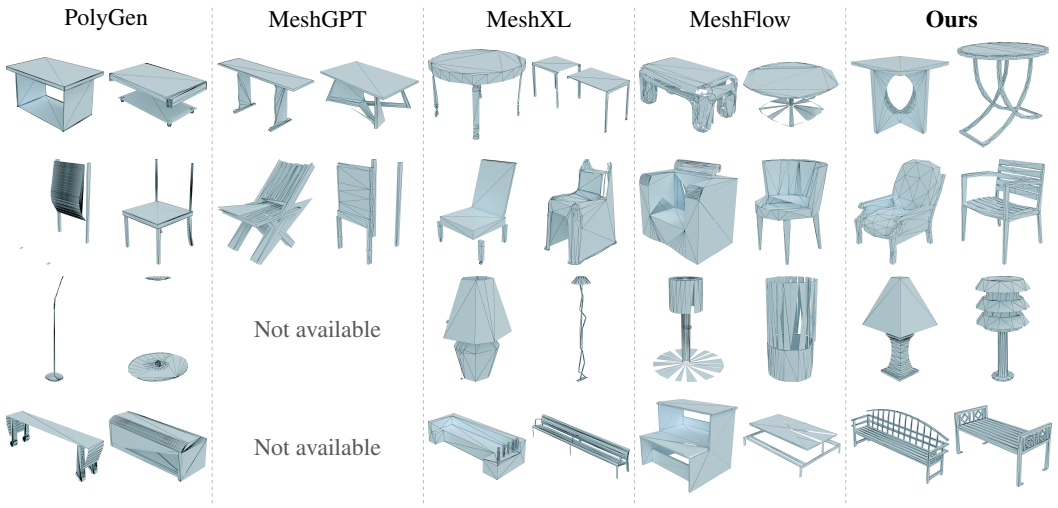

Figure 6: **Qualitative mesh generation comparison.**

	Method	#Params	Runtime (s)	Chair			Table			Bench			Lamp		
				1-NNA	Tight.↓	Dup.↓	1-NNA	Tight.↓	Dup.↓	1-NNA	Tight.↓	Dup.↓	1-NNA	Tight.↓	Dup.↓
Auto-regressive	PolyGen	174M	318	80.03	**5.01**	2.2	67.20	4.34	2.1	68.57	4.23	**4.0**	65.16	**3.95**	**2.3**
	MeshGPT	350M	32	55.35	30.44	**1.9**	**52.61**	21.67	**0.9**	–	–	–	–	–	–
	MeshXL	125M	20	**54.26**	7.15	13.4	58.56	**4.17**	15.3	**53.39**	**4.18**	14.0	**52.58**	9.13	21.4
Flow Matching	PolyDiff	132M	–	79.91	–	–	73.25	–	–	61.49	–	–	70.81	–	–
	MeshFlow	586M	2.8	53.03	66.30	4.1	57.55	85.32	**3.2**	62.68	136.8	13.6	**51.15**	103.7	11.7
	Ours	130M	2.6	**51.77**	**14.31**	**3.4**	**56.54**	**18.88**	4.8	**49.82**	**8.71**	**3.9**	51.45	**13.67**	**5.1**

Table 1: **Unconditional generation on four ShapeNet categories.** Tightness is scaled by 10^4. Best and second-best results within each method group are bold and underlined, respectively (1-NNA closest to 50%; lower Tight. and Dup.). Runtime is measured for chair generation.

Metrics. We evaluate generation quality using Chamfer-distance 1-NNA, which distinguishes generated from reference shapes using a one-nearest-neighbor classifier; values near 50% indicate similar distributions. For mesh quality, we report the mean percentage of duplicate faces (**Dup.**) and the minimum edge-assignment cost (**Tight.**), which measures how closely edges align with distinct counterparts. Lower values indicate fewer duplicates and tighter face alignment, respectively. Full definitions are provided in App. A.2.

Baselines. We select baselines that does unconditional mesh generation with publicly available checkpoints or previously reported results under our evaluation protocol. Autoregressive baselines include PolyGen (Nash et al., 2020), MeshGPT (Siddiqui et al., 2024), and MeshXL (Chen et al., 2024), evaluated using their released checkpoints. MeshGPT provides checkpoints only for Chair and Table; for MeshXL, we use the 125M variant to match our model scale. Diffusion-based baselines include PolyDiff (Alliegro et al., 2023) and MeshFlow (Sun et al., 2026). We evaluate MeshFlow using its released checkpoint and report PolyDiff's 1-NNA scores from Sun et al. (2026), as its implementation is not public.

Implementation details. We train a 130M-parameter DiT based on the MeshFlow architecture (Sun et al., 2026), incorporating vertex weighting and face attention with Laplacian positional embeddings. We replace the final DiT block with vertex attention. We train for 850k steps with a batch size of 32 on a single H100. At inference, we use 100 steps of DDIM sampling (Song et al., 2020). See the Supplementary Material for training details.

4.1 Unconditional Mesh Generation

Table 1 compares DeltaFlow against the baselines. PolyGen, MeshGPT, and MeshXL produce tight meshes but generate faces sequentially, resulting in slow sampling, and rely on vertex quantization. MeshXL is prone to duplicate faces. MeshFlow samples faster but produces non-tight meshes.

DeltaFlow achieves tightness close to autoregressive methods while providing fast sampling with a continuous output space. It reduces tightness error by 4.5–$15.7\times$ relative to MeshFlow while achieving the best or second-best 1-NNA among all evaluated methods across all four categories. Figure 6 compares generated meshes across four categories. PolyGen and MeshGPT samples exhibit missing or distorted structures, while MeshXL produces disconnected components and MeshFlow shows visible gaps between faces. Our samples show more coherent structures, including complex meshes such as curved table supports, slatted chairs, layered lampshades, and detailed bench frames.

4.2 Ablations

We ablate DeltaFlow by replacing our PE and architectural modifications separately. To assess the effect of Laplacian PE, we remove positional embedding (*No PE*). To assess our architectural modifications, we disable vertex weighting (*No vertex weighting*) and the final vertex-attention layer (*No vertex attn.*) (Sec. 3.4).

As shown in Tab. 2 and Fig. 7, No PE achieves the worst tightness, supporting the analysis in Sec. 3.2. Both vertex weighting and vertex attention further improve tightness, supporting the importance of distinguishing vertex slots and enabling direct interactions between vertices across faces.

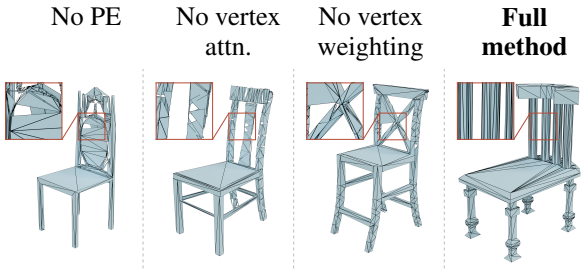

Figure 7: **Qualitative ablations** on ShapeNet Chair. Insets magnify backrest details.

Method	1-NNA	Tight.	Dup.
No PE	78.94	35.53	4.8
No vertex attention	53.55	24.14	**2.3**
No vertex weighting	55.97	17.49	3.3
Full method	**51.77**	**14.31**	3.4

Table 2: **Ablations on ShapeNet Chair.** Positional encoding, vertex attention, and vertex weighting; 100 sampling steps.

4.3 Applications

After training DeltaFlow, we can leverage its generative prior for various *training-free* inverse tasks such as point-cloud-conditioned generation, and mesh completion.

Point-cloud-conditioned Generation. Following FlowDPS Kim et al. (2025), at each sampling step we refine the predicted posterior mean $\hat{x}_1 = x_t + (1-t)u_\theta(x_t, t)$ toward a given point cloud P using gradient descent on a modified Chamfer distance objective. The refined estimate $\tilde{x}_1$ is converted back into a velocity, $\hat{u} = (\tilde{x}_1 - x_t)/(1-t)$, for $t < 1$, which we use to update the sampling trajectory. See App. A.2 for algorithmic details.

We show point-cloud-conditioned generation samples in Fig. 8 compared with the same algorithm applied to MeshFlow Sun et al. (2026). Compared to our generation samples which well aligns to the input point-cloud and maintains good mesh quality, MeshFlow generates broken meshes with self-intersecting and missing faces.

Mesh completion. Given a partial mesh, we generate the missing faces while preserving the observed triangles. Since DeltaFlow generates all faces jointly, it can complete arbitrary subsets of faces without a prescribed generation order. Inspired by diffusion inpainting (Lugmayr et al., 2022), we match observed triangles to predicted faces and hold them fixed during sampling. Figure 9 compares completion results with MeshFlow. Both methods preserve the input triangles by construction, but our completions connect more coherently to the observed geometry. Our matching step associates observed faces with Laplacian positional embeddings, allowing the generative prior to complete the remaining structure. Algorithmic details are provided in the supplementary material.

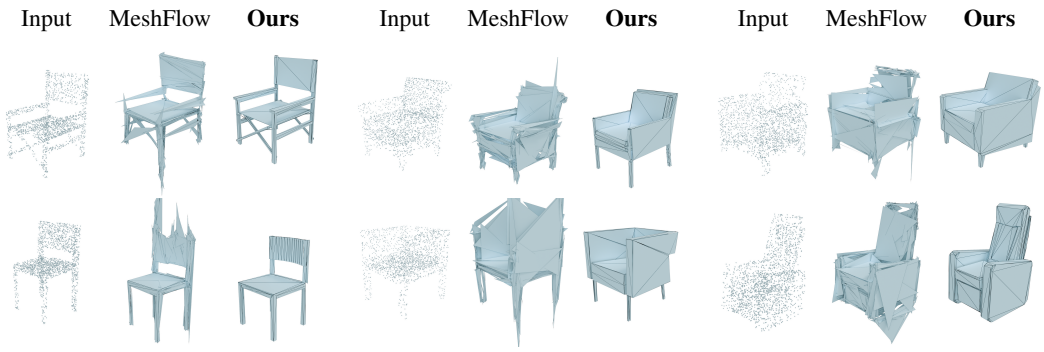

Figure 8: **Point cloud conditioning.** Each example shows a shared point-cloud input (left), a MeshFlow generation (center), and a DeltaFlow generation (right).

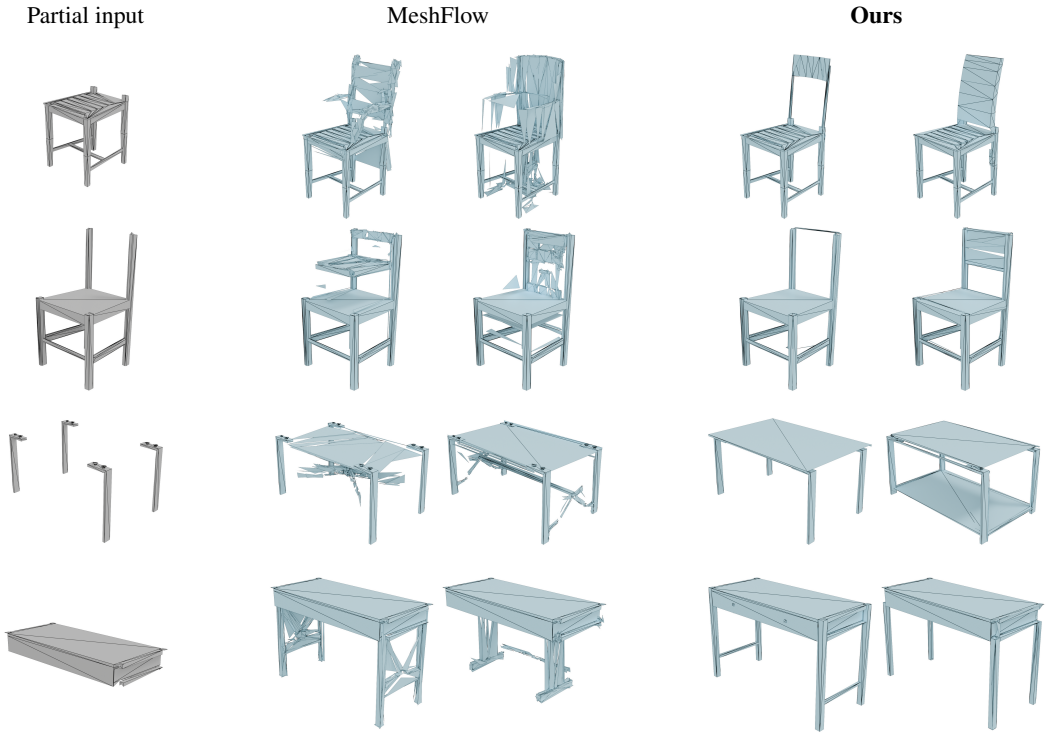

Figure 9: **Mesh completion**. Each row shows a partial input (gray, left) and completions (blue) from MeshFlow (center) and DeltaFlow (right). The missing region can be any subset of faces.

5 Conclusions and Limitations

We introduced DeltaFlow, a single-stage flow model that directly generates triangle meshes. Our Laplacian positional embedding provides an intrinsic connectivity prior during sampling, enabling substantially tighter meshes than previous diffusion-based mesh generative models. The learned generative prior also supports training-free inverse problems, as demonstrated by point-cloud-conditioned generation and mesh inpainting. Scaling training to larger and more diverse datasets could extend its use to a broader range of mesh editing and geometry processing tasks.

Preserving face neighborhoods can be challenging when assigning each mesh's spectral coordinates to a fixed template. Alternative or adaptive templates could help reduce this distortion. Representing each face as a separate token also makes attention costly for meshes with many faces, limiting scalability during both training and generation. Patch-based representations, inspired by image diffusion models (Peebles & Xie, 2023), offer a promising direction for reducing this cost.

AI USE STATEMENT

We used generative AI tools to assist with implementing and debugging methods. We also used these tools for manuscript editing and making visualization code. The authors take responsibility for the final text, claims, code, and experimental results.

REFERENCES

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

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.

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, 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 *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13922–13931. IEEE, 2025.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.

Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*, pp. 1–126. 2013.

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*, 2025a.

Xianglong He, Zi-Xin Zou, Chia-Hao Chen, Yuan-Chen Guo, Ding Liang, Chun Yuan, Wanli Ouyang, Yan-Pei Cao, and Yangguang Li. Sparseflex: High-resolution and arbitrary-topology 3d shape modeling. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14822–14833. IEEE, 2025b.

Jeongsol Kim, Bryan Sangwoo Kim, and Jong Chul Ye. Flowdps: Flow-driven posterior sampling for inverse problems. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12328–12337. IEEE, 2025.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pp. 193–202. 2023.

Haoxuan Li, Ziya Erkoç, Daniele Sirigatti, Vladislav Rosov, Lei Li, Angela Dai, and Matthias Nießner. Triflow: Generating artist-like 3d mesh topology via nearest-vertex vector fields. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026a.

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)*, 2026b. Highlight.

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

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11451–11461. IEEE, 2022.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4455–4465. IEEE, 2019.

Kiyohiro Nakayama, I-Chao Shen, Ruofan Liu, Yiming Wang, Gordon Wetzstein, and Takeo Igarashi. Garment particles: a 2d–3d symmetric garment representation for generation and editing. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–12, 2026.

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

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 165–174. IEEE, 2019.

Maitreya Patel, Song Wen, Dimitris N Metaxas, and Yezhou Yang. Flowchef: Steering of rectified flow models for controlled generations. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15308–15318. IEEE, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Matt Pharr, Greg Humphreys, Saty Raghavachary, and Pat Hanrahan. *Physically based rendering*. Elsevier, 2016.

Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4209–4219. IEEE, 2024.

Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. In *ACM SIGGRAPH 2004 Papers*, pp. 878–887. 2004.

Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. Spacemesh: A continuous representation for learning manifold surface meshes. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24)*, pp. 11, New York, NY, USA, 2024. ACM. doi: 10.1145/3680528.3687634. URL https://doi.org/10.1145/3680528.3687634.

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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19615–19625. IEEE, 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 *International Conference on Learning Representations*, volume 2026, pp. 112688–112723, 2026.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Robert W Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Transactions on graphics (TOG)*, 23(3):399–405, 2004.

Qi Sun, Kiyohiro Nakayama, Jing Nathan Yan, Qixing Huang, Alexander Rush, Leonidas Guibas, Gordon Wetzstein, Jing Liao, and Guandao Yang. Meshflow: Mesh generation with equivariant flow matching. In *ACM SIGGRAPH Conference Papers*, 2026.

Jiaxiang Tang, Max Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. Edgerunner: Auto-regressive auto-encoder for artistic mesh generation. In *International Conference on Learning Representations*, volume 2025, pp. 35913–35934, 2025.

Joan Verdera, Vicent Caselles, Marcelo Bertalmio, and Guillermo Sapiro. Inpainting surface holes. In *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, volume 2, pp. II–903. IEEE, 2003.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

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 (TOG)*, 45(4):1–14, 2026.

Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, et al. Scaling mesh generation via compressive tokenization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11093–11103, 2025.

Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. Direct3d: Scalable image-to-3d generation via 3d latent diffusion transformer. *Advances in Neural Information Processing Systems*, 37:121859–121881, 2024.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21469–21480, June 2025.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4540–4549. IEEE, 2019.

Hao Zhang, Oliver Van Kaick, and Ramsay Dyer. Spectral mesh processing. In *Computer graphics forum*, volume 29, pp. 1865–1894. Wiley Online Library, 2010.

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. *ACM Transactions On Graphics (TOG)*, 43(4):1–20, 2024.

A Appendix

A.1 Proof of Proposition 1

Let $\pi = (i\,j)$ exchange two faces and set $d_t = x_t^i - x_t^j$. Equivariance and the Lipschitz assumption give

$$\begin{aligned}\sqrt{2}\,\|u_\theta^i(x_t,t) - u_\theta^j(x_t,t)\|_2 &= \|u_\theta(x_t,t) - \pi\cdot u_\theta(x_t,t)\|_{\mathrm{F}} \\ &= \|u_\theta(x_t,t) - u_\theta(\pi\cdot x_t,t)\|_{\mathrm{F}} \\ &\le L\|x_t - \pi\cdot x_t\|_{\mathrm{F}} = \sqrt{2}L\|d_t\|_2.\end{aligned}$$

Consequently $\left\|\frac{d}{dt}d_t\right\|_2 \le L\|d_t\|_2$. For $g(t) = \|d_t\|_2^2$, Cauchy–Schwarz implies

$$-2Lg(t) \le \frac{dg(t)}{dt} = 2\left\langle d_t, \frac{d}{dt}d_t\right\rangle \le 2Lg(t). \tag{10}$$

Integrating the two differential inequalities yields $e^{-2Lt}g(0) \le g(t) \le e^{2Lt}g(0)$, including when $g(0) = 0$. Taking square roots proves Eq. (4). The result concerns full face vectors, equality of one vertex from each of two distinct faces does not place the state on the fixed set of their face swap.

A.2 Additional Implementation Details

We use 12 transformer layers: 11 face-attention layers and one final vertex-attention layer. Time t and face count F condition the network through adaLN. We learn face-count embeddings at intervals of 20 faces and linearly interpolate between the two neighboring embeddings for a given F, without classifier-free guidance. For constructing the Laplacian Embedding from disconnected meshes we augment the graph with links from each face to its six nearest neighbors by centroid distance, weighted by 0.1, original dual-graph edges retain weight 1.

For RoPE, we project each positional embedding c_i onto the three honeycomb bond directions $(0, 1)$, $(-\sqrt{3}/2, -1/2)$, and $(\sqrt{3}/2, -1/2)$ and apply rotary embeddings along each projected coordinate.

We train for 850k steps using the flow-matching objective in Eq. (2), with velocity target $x_1 - x_0$ and independent pairs $(x_0, x_1) \sim p_0 \otimes p_1$. We find independent coupling sufficient for training and omit the nested optimal-transport coupling of Sun et al. (2026), removing an $O(F^3)$ solve per sample from the training loop. We sample t from a mixture of a logit-normal distribution concentrated near $t = 0.8$, rather than $t = 0.5$, and a uniform distribution. This emphasizes the data end of the path, where faces must align precisely to share vertices, we find that emphasizing these times improves tightness.

At inference, we use 100 steps of DDIM sampling (Song et al., 2020) with full renoising, which improves tightness compared with deterministic sampling. To recover mesh connectivity from the continuous output, we discretize each coordinate axis into 512 bins, merge vertices and edges according to a distance threshold, and remove degenerate or duplicate faces.

Duplicates. For each generated mesh, we compute the percentage of exact duplicate faces, i.e., faces sharing the same three vertices, and report the mean over the sampled set. Duplicates are counted before the post-processing step that removes them. Because MeshFlow and DeltaFlow output continuous coordinates, exact duplicates are only well defined after vertex merging, so for these models we count them after merging and before duplicate removal.

Tightness. Given a triangle soup with F faces, we form its $E = 3F$ edges $(x^{i,1}, x^{i,2}), (x^{i,2}, x^{i,3}), (x^{i,3}, x^{i,1})$, where $x^{i,v} \in \mathbb{R}^3$ denotes vertex slot v of face i. In a stitched mesh every edge has an exact twin on a neighboring face, so we score how far each edge is from its best available partner. The cost between two edges a, b is given by:

$$d_{\mathrm{H}}(a, b) = \max\Big(\max_k \min_l \|a_k - b_l\|_2,\ \max_l \min_k \|a_k - b_l\|_2 \Big),$$

which is zero if the edges share both endpoints. We solve a linear assignment over all E edges (each edge is used as a partner once), forbidding an edge from matching itself or the other two edges of its own triangle, and report

$$\text{tightness} = \frac{1}{\ell E} \sum_e d_{\mathrm{H}}\big(e, \tau^\star(e)\big),$$

with ℓ the bounding-box diagonal and $\tau^\star$ the assignment. Boundary edges have no twin and are matched at nonzero cost, so tightness is bounded away from zero for open surfaces. Duplicate and degenerate faces admit zero-cost matches without reflecting connectivity and are removed prior to computing this metric. We report the mean over the set of generated meshes. Numbers are multiplied by 10^4.

Mesh completion Given a subset of faces $\{x_1^k\}_{k \in \mathcal{S}}$ of a mesh x_1 with F faces, we generate the remaining faces while keeping the given ones fixed. Since faces are unordered, we first sample unconstrained until $t_m = 0.2$, then match each given face to a generated face by linear assignment on the posterior mean $\hat{x}_1$, using Euclidean distance minimized over cyclic vertex orderings. The assignment γ stays fixed, so each given face inherits the positional embedding of its matched face. After every subsequent step we replace the matched faces with the interpolation of the given faces at the current time. Algorithm 1 summarizes the full sampling procedure.

Algorithm 1 Mesh completion

Require: model u_θ, given faces $\{x_1^k\}_{k\in\mathcal{S}}$, matching time t_m

$x_{t_0} \sim p_0$; $\gamma \leftarrow \varnothing$
for $n = 0, \dots, N-1$ **do**
  $u \leftarrow u_\theta(x_{t_n}, t_n)$; $\hat{x}_1 \leftarrow x_{t_n} + (1 - t_n)\, u$
  if $\gamma = \varnothing$ **and** $t_n \geq t_m$ **then**
    $\gamma \leftarrow$ linear assignment between $\{x_1^k\}_{k\in\mathcal{S}}$ and faces of $\hat{x}_1$
  end if
  $x_{t_{n+1}} \leftarrow t_{n+1}\, \hat{x}_1 + (1 - t_{n+1})\, \epsilon, \quad \epsilon \sim p_0$
  $x_{t_{n+1}}^{\gamma(k)} \leftarrow t_{n+1}\, x_1^k + (1 - t_{n+1})\, \epsilon^{\gamma(k)}$ **for all** $k \in \mathcal{S}$
end for
return x_{t_N}

Point cloud conditioning The guidance combines two terms: one asks that every observed point lie on the mesh, and the other asks that the mesh surface stay on the observed points:

$$\mathcal{L}(\hat{x}_1, P) = \frac{1}{|P|} \sum_{p\in P} \min_i \operatorname{dist}\left(p, \hat{x}_1^i\right)^2 + \frac{1}{|Y|} \sum_{y\in Y} \max\Big(\min_{p\in P} \|y - p\|_2 - \delta,\ 0\Big)^2. \tag{11}$$

The first term is the one-sided squared Chamfer distance from the point cloud to the mesh, where, $\operatorname{dist}(p, \hat{x}_1^i)$ denotes the exact Euclidean distance from p to the triangle spanned by face i of $\hat{x}_1$. The second term is a hinged one-sided squared Chamfer distance from the mesh to the point cloud, computed over a set of points Y sampled on the mesh surface, where the hinge δ leaves surface that is already close to the cloud unpenalized. To keep the mesh connected, the optimization does not move vertices independently. We found that optimizing each vertex on its own tears the mesh apart. Algorithm 2 summarizes the full sampling procedure.

Algorithm 2 Point-cloud conditioned sampling

Require: model u_θ, point cloud P

$x_{t_0} \sim p_0$
for $n = 0, \dots, N-1$ **do**
  $u \leftarrow u_\theta(x_{t_n}, t_n)$; $\hat{x}_1 \leftarrow x_{t_n} + (1 - t_n)\, u$; $\hat{x}_0 \leftarrow x_{t_n} - t_n\, u$
  $\tilde{x}_1 \leftarrow \arg\min_{\hat{x}_1} \mathcal{L}(\hat{x}_1, P)$
  $\hat{x}_1 \leftarrow \lambda(t_n)\, \tilde{x}_1 + \big(1 - \lambda(t_n)\big)\, \hat{x}_1$
  $\hat{x}_0 \leftarrow \sqrt{1 - t_{n+1}}\, \hat{x}_0 + \sqrt{t_{n+1}}\, \epsilon, \quad \epsilon \sim p_0$
  $x_{t_{n+1}} \leftarrow t_{n+1}\, \hat{x}_1 + (1 - t_{n+1})\, \hat{x}_0$
end for
return x_{t_N}

A.3 Locality of face attention

To examine how the Laplacian positional embeddings shape face attention, we visualize attention weights for a single query face across sampling times. We take the first and last face-attention layers of DeltaFlow, select a query face i, and compute its attention weights $\alpha_{ij} = \operatorname{softmax}_j(\ell_{ij})$ over all key faces j, averaged over heads . We display each weight α_{ij} at the key's template position c_j. Since the assignment places mesh-adjacent faces broadly at nearby template positions with some far away tokens. Locality on the template approximately corresponds to locality in the mesh's final connectivity.

Figure 10 shows that attention concentrates around the query's template neighborhood. At $t = 0$, where face coordinates are pure noise, attention is broad but already centered on the query, suggesting that the positional embeddings alone provide an initial neighborhood prior. As t increases, attention becomes more localized. This effect is strongest in the last layer, where attention at later

times is concentrated on a few template positions adjacent to the query, while the first layer maintains a wider local neighborhood.

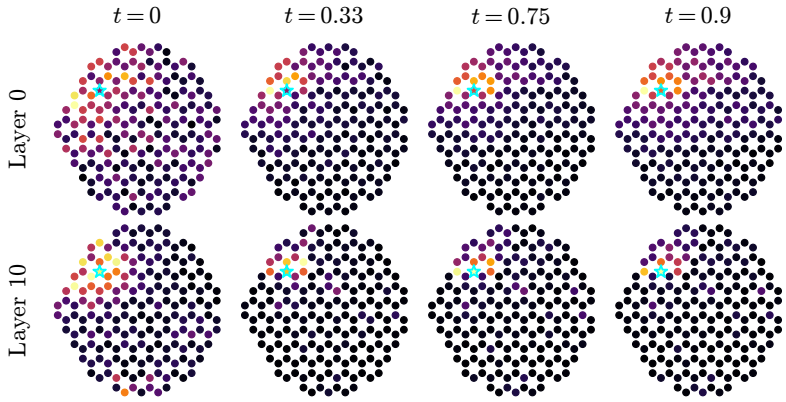

Figure 10: **Face attention on the template.** Attention weights of a query face (cyan star) over all faces, displayed at their template positions, for the first and last face-attention layers at sampling times. Brighter colors indicate higher attention. Attention concentrates around the query's template neighborhood

A.4 Additional unconditional samples

Figure 11 shows additional unconditional samples from DeltaFlow across the four ShapeNet categories. Samples are generated with 100 sampling steps using the same models evaluated in Table 1.



Figure 11: **Unconditional samples on ShapeNet chairs, tables, benches, and lamps.**