GEOMETRIC FLOW MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a flow-based generative model for 3D surface generation that leverages geometric flows within the flow matching framework. Instead of relying on generic stochastic interpolation paths, we construct data-dependent trajectories using a curvature-driven geometric evolution that progressively smooths surfaces while preserving mesh topology. This formulation defines structured transport paths between simple base shapes and complex target geometries, enabling the model to learn geometry-aware transformations. Our approach operates directly on triangle meshes and produces coherent deformation sequences governed by surface curvature. We evaluate our method on 3D hand and face mesh datasets and compare it against alternative flow construction strategies, demonstrating that curvature-driven trajectories yield more realistic, geometrically faithful meshes.

1 INTRODUCTION

Flow-based models have recently emerged as a powerful class of generative models for learning complex probability distributions (Rezende and Mohamed, 2015; Papamakarios et al., 2021). Flow matching provides a simulation-free framework for training these models (Lipman et al., 2023; Liu et al., 2023): instead of explicitly optimizing the likelihood, it directly learns the velocity field that transports samples between base and target distributions. Flow matching has been successfully applied to generate images (Lipman et al., 2023; Esser et al., 2024), audio (Le et al., 2023), video (Polyak et al., 2024), and proteins (Watson et al., 2023; Yim et al., 2023).

The generation of 3D surfaces has numerous applications in animation, gaming, and virtual and augmented reality. Existing approaches often represent surfaces as point clouds (Yang et al., 2019; Luo and Hu, 2021), voxels (Wu et al., 2016), or implicit functions (Park et al., 2019; Mildenhall et al., 2020), which do not explicitly capture the structured connectivity of meshes and often struggle to represent fine-grained surface details such as sharp contours and subtle geometric variations. While flow matching has been successfully applied across a range of domains, its use for generating 3D surfaces with explicit geometric features (e.g., mesh connectivity and curvature information) remains relatively unexplored.

In this paper, we present a flow-based model for generating 3D surfaces by combining flow matching with geometric flows, curvature-driven differential equations that describe the evolution of surfaces over time. In the same way that heat diffusion evens out temperature variations, there are geometrical flows that progressively smooth geometric irregularities and drive complex shapes toward simpler canonical configurations. Leveraging this property, our approach defines a data-dependent transport path between a simple base shape (e.g., a sphere) and complex target surfaces, while maintaining mesh validity and preserving topology, producing smooth, coherent intermediate surfaces.

Standard diffusion models/flow-based models are designed for regular 2D grids and cannot account for the 3D geometry of a surface. In image-based diffusion models, the forward process adds Gaussian noise to pixel values. This works because images have a clear separation between structure and content: the grid structure is fixed, and pixel values can be perturbed independently without invalidating the representation. A noisy image is still a valid image, but just a degraded one. Meshes do not have this property. Given fixed connectivity, vertex positions fully determine a mesh's geometry — its edge lengths, face orientations, and surface continuity. Unlike pixels, vertices are geometrically coupled through shared edges and faces, meaning that independently perturbing their positions disrupts these relationships, producing self-intersections, inverted normals, and surface folds. Connected smooth meshes form a narrow subset of vertex configurations, and small perturbations

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.

quickly break validity. We confirm this empirically in Section 4: in the first designed baseline, a model trained to denoise randomly perturbed vertices fails to learn a useful reverse process. A geometric flow, by contrast, provides a geometrically meaningful degradation, producing a natural sequence of meshes from the original shape toward a simpler form.

Concretely, we represent surfaces as triangle meshes embedded in R^3 and construct a conditional path using a geometric flow that evolves vertex positions directly in the ambient space. This yields a deterministic, geometry-aware coupling between source and target samples, producing structured trajectories that are well suited for learning within the flow matching framework. We refer to this approach as Geometric Flow Matching (GFM).

2 Background and Related Work

Flow-based generative models Flow-based generative models learn a transformation that maps samples from a base distribution p_0 to a target distribution p_{data} via a time-dependent velocity field, with the corresponding density evolving according to the continuity equation. Flow Matching learns this velocity field through regression, avoiding the expensive divergence computation required by likelihood-based training (Appendix A). Conditional Flow Matching further introduces a conditioning variable z that determines the marginal path and training objective. Once the velocity field is learned, new samples are generated by integrating the learned dynamics from an initial point $\mathbf{x}_0 \sim p_0$.

Flows in Riemannian Geometry Geometric flows are dynamical processes that evolve geometric quantities over time according to partial differential equations. A classical example is Ricci flow, which on a two-dimensional manifold evolves the metric according to curvature, gradually deforming the surface's intrinsic geometry. It can be shown that applying the normalized Ricci flow to a closed genus-zero surface causes the evolving geometry to converge to the round sphere, providing a canonical endpoint for the evolution.

While Ricci flow is an intrinsic flow (dealing only with intrinsic quantities such as the metric), extrinsic geometric flows describe how a surface evolves within an ambient space. A standard example is mean curvature flow, which smooths the surface by causing regions of high curvature to move more rapidly. Another example is Willmore flow, which minimizes the bending energy and thereby promotes smoother surface geometry. Although these geometric flows are defined differently, they all act to regularize geometry and reduce curvature variations over time (see Appendix A).

3 Method

3.1 Problem formulation

Given a mesh dataset, our goal is to learn a generative model that can produce detailed meshes starting from a simple sphere-like template. To do so, we define a forward process based on a geometric flow that progressively smooths each mesh toward a sphere, and train a neural network to reverse this process. This framework is analogous to diffusion/flow-based models in the image domain, where a forward process gradually corrupts data, and a learned network reverses it to generate samples. However, instead of adding random noise, the forward process here comes from a geometric PDE.

Let p_{data} denote the distribution[1] over a collection of closed 2D manifolds (meshes) with fixed connectivity. We represent each manifold as an embedded surface in the ambient Euclidean space $\mathbb{R}^3$. After discretization with N vertices, the surface is represented by its vertex positions in $\mathbb{R}^{3N}$. Now, geometric flows defined on the manifolds can be understood as an *induced flow* in the space of vertex coordinates. Each flow step maps one vertex configuration $X \in \mathbb{R}^{3N}$ to another, thus inducing a trajectory in the ambient Euclidean space. Our framework operates on this induced trajectory: rather than working with the geometric flow's native quantities (e.g., the metric tensor g_{ij} or the mean curvature H), we treat the sequence of vertex positions as the conditional path ψ_t in the conditional flow matching (CFM) framework.

[1]The distribution p_{data} is defined over embedded surfaces, where each surface is represented by vertex coordinates in Euclidean space. We do not model distributions on the manifold itself, so this formulation is distinct from generative modeling *on* Riemannian manifolds.

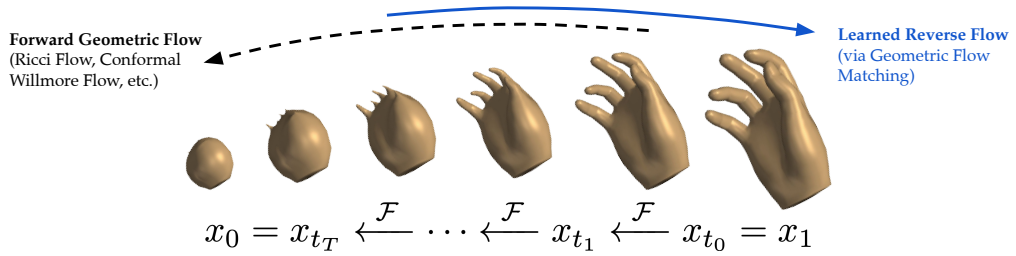


Figure 1: **Overview of Geometric Flow Matching (GFM).** A forward geometric flow $\mathcal{F}$ is applied to each target mesh x_1, producing a deterministic trajectory of progressively smoother surfaces $x_{t_0}, x_{t_1}, \ldots, x_{t_T}$ that converges to a spherical base configuration x_0 (right to left). A model operating on meshes is then trained with a conditional flow matching objective to approximate the reverse process, predicting per-vertex velocities that transport samples from the base distribution back to detailed target meshes (left to right, blue arrow).

The perspective of using these specific geometric flows as transformations that evolve meshes toward spheres provides a natural way to define the base distribution p_0. Just as standard CFM uses isotropic Gaussian noise as a structureless base distribution, we construct our base distribution from near-spherical meshes, or slightly perturbed spheres. In other words, we use surfaces of approximately constant curvature as the mesh analog of white noise.

3.2 Geometric Flow Construction

For each data sample $x_1 \sim p_{\text{data}}$, the forward geometric flow produces a discrete sequence of progressively smoother meshes

$$x_1 = x_{t_0} \xrightarrow{\mathcal{F}} x_{t_1} \xrightarrow{\mathcal{F}} \cdots \xrightarrow{\mathcal{F}} x_{t_T} = x_0. \tag{1}$$

Here, $x_0 = \mathcal{F}_T(x_1)$ denotes the final smoothed configuration, and $t_i = 1 - \frac{i}{T}$ for $i = 0, \ldots, T$. Consequently, the marginal base distribution p_0 is given by the pushforward of p_{data} under $\mathcal{F}_T$, namely $p_0 = (\mathcal{F}_T)_{\#}\, p_{\text{data}}$. Since the flow maps diverse inputs to meshes with similar, near-spherical geometry, the resulting base distribution p_0 is concentrated on near-spherical meshes. In practice, we approximate p_0 by its empirical distribution over the smoothed meshes $\{x_0^{(k)} = \mathcal{F}_T(x_1^{(k)})\}$, obtained by applying the geometric flow $\mathcal{F}_T$ to each mesh in the training set prior to training. The training set is disjoint from the held-out meshes used for evaluation.

The framework presented in this paper is not specific to a single geometric flow. In principle, any geometric flow can serve as the forward process provided it satisfies three conditions: (i) *convergence to a canonical shape*, meaning the flow drives all surfaces toward a common limit geometry (e.g., a sphere), inducing a well-defined base distribution p_0; (ii) *smooth evolution that preserves topology*, meaning the flow produces a continuous family of geometrically valid surfaces, ensuring every intermediate mesh remains a valid 2-manifold; and (iii) *numerical integrability*, meaning a discrete integrator allows computing the flow step by step, providing the training pairs needed for supervision.

However, the uniqueness properties of the forward geometric flow do not hold in reverse, as the governing partial differential equation is no longer parabolic. A similar challenge arises in standard diffusion models, where the reverse diffusion equation also lacks guaranteed classical solutions. In practice, rather than solving the reverse equation analytically, the reverse process is learned using neural networks. We follow the same principle: a network is trained to predict the reverse geometric flow, recovering surface detail from smoothed meshes step by step.

We adopt Willmore flow, an extrinsic geometric flow that evolves an embedded mesh by minimizing the bending energy $\int H^2\, dA$. This formulation naturally aligns with our mesh representation, as it directly updates vertex positions while remaining in the space of embedded surfaces. Specifically, we employ the conformal Willmore flow of Crane et al. (2013), whose discrete formulation is well suited to triangle meshes and provides a stable and efficient numerical scheme without requiring remeshing. The Willmore flow drives closed genus-zero meshes toward nearly spherical configurations while

maintaining mesh quality, thereby satisfying the above design criteria. Since the discrete Willmore-flow solver assumes closed, consistently oriented, self-intersection-free manifold meshes, these conditions are also required by our framework.

3.3 Conditional Path and Coupling

To define the conditional probability path, we replace standard linear interpolation with a data-dependent trajectory induced by a geometric flow. Specifically, we define

$$p_t(x \mid z) = \delta_{\psi_t(x_1)}(x), \tag{2}$$

where $\psi_t(x_1) = \mathcal{F}_t(x_1)$ denotes the intermediate mesh obtained by evolving x_1 under the flow. This defines a Dirac conditional path supported on a deterministic trajectory in $\mathbb{R}^{3N}$. At each time t, the mesh $\psi_t(x_1)$ remains a valid embedded surface, preserving mesh quality throughout the evolution. Since the conditional path is deterministic, the associated conditional velocity field follows directly from the continuity equation as the time derivative of the trajectory:

$$u_t^{\text{cond}}(x, z) = \frac{d}{dt}\psi_t(x_1)\bigg|_{x=\psi_t(x_1)}. \tag{3}$$

In practice, this velocity is approximated using finite differences between consecutive mesh configurations along the discrete flow sequence. Notably, if ψ_t were chosen as the linear interpolation $(1-t)x_0 + tx_1$, this formulation recovers the standard CFM velocity field.

We define a deterministic, data-dependent coupling by setting $z = (x_0, x_1)$, where $x_0 = \mathcal{F}_T(x_1)$ is obtained by the geometric flow. This induces the joint distribution

$$p(z) = p(x_0, x_1) = p_{\text{data}}(x_1)\, \delta_{\mathcal{F}_T(x_1)}(x_0). \tag{4}$$

Unlike independent couplings, which pair samples randomly, or optimal transport couplings, which require solving a global optimization problem, our approach constructs correspondences through a deterministic geometric map. This yields structured pairings that reflect the underlying surface geometry while remaining computationally straightforward.

3.4 Learning Objective

Following the CFM framework, we train the network to approximate the conditional velocity field by minimizing

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\ z\sim p(z)}\left[\left\|u_\theta(x,t) - u_t^{\text{cond}}(x,z)\right\|^2\right], \tag{5}$$

where $x = \psi_t(x_1)$ is the mesh at time t along the precomputed flow trajectory determined by $z = (\mathcal{F}_T(x_1), x_1)$. In practice, this is implemented as an MSE loss between the predicted velocities and finite-difference estimates of the ground-truth per-vertex velocities.

At inference time, we sample an initial near-spherical mesh $x_0 \sim p_0$ from the empirical base distribution of pre-computed smoothed meshes, held out from training; the model is applied autoregressively, as described below, to produce a generated mesh x_1, or any other desired x_{t_i}.

3.5 Network Architecture

Our model operates directly on triangular meshes, represented as graphs where vertices are nodes with 3D coordinates and edges encode mesh connectivity. This representation allows the network to leverage the mesh connectivity directly, without requiring parameterization. The architecture consists of Chebyshev spectral graph convolutional layers (ChebConv) Defferrard et al. (2016) followed by per-vertex fully connected layers. ChebConv layers approximate spectral graph filters using K-th order polynomials of the graph Laplacian, enabling aggregation over K-hop neighborhoods.

The model predicts per-vertex displacements between consecutive geometric flow steps rather than absolute vertex positions. This residual formulation simplifies the learning task, as successive meshes differ locally and enables autoregressive generation of full flow trajectories. The predicted displacement corresponds to the velocity field learned in flow matching, discretized over consecutive flow steps.

The network is applied autoregressively at inference: starting from a mesh at any point along the trajectory, its own predicted displacement is fed back as input to produce the next step, repeated until the end of the 120-step trajectory. Since the network takes neither time nor step index as input and instead relies only on the current vertex positions and mesh edges, the same fixed function is reused at every step, regardless of where the current mesh sits along the trajectory.

4 EXPERIMENTS

4.1 DATA GENERATION

We evaluate our method on two datasets of 3D meshes: parametric hand meshes sampled from the MANO model (Romero et al., 2017), and face meshes reconstructed from images via MediaPipe Face Mesh (Lugaresi et al., 2019). In both cases, we close any boundary loops to obtain watertight manifolds and apply 120 steps of discrete Willmore flow (Crane et al., 2013) to progressively smooth each mesh toward a near-spherical surface. The step size follows an adaptive schedule $\tau(i) = \tau_{\min} + (\tau_{\max} - \tau_{\min})\ (\alpha)^p$, where $\alpha = \frac{i}{T-1}$ and T is the total number of steps. The original implementation of Crane et al. (2013) uses a constant step size; we adopt an adaptive schedule to better accommodate meshes of varying complexity and scale. Consecutive pairs (V_t, V_{t+1}) from the resulting sequences serve as training samples. Vertex positions are normalized per step to $[-1, 1]$ via min-max scaling, and the mesh connectivity remains fixed throughout the flow.

MANO hands. Hand meshes are sampled from the MANO model by randomly drawing pose (48-dimensional) and shape (10-dimensional) parameters. Boundary loops at the wrist are capped with a triangular fan, yielding meshes with 779 vertices and 1,554 faces. Each mesh undergoes 120 steps of Willmore flow with $\tau_{\min} = 0.003$, $\tau_{\max} = 0.035$, and $p = 1.8$.

MediaPipe faces. We select 2,500 facial images from CelebA (Liu et al., 2015) and process each as follows. A 3D face mesh is extracted using MediaPipe Face Mesh, which produces a canonical triangulation of 468 vertices. As the resulting mesh is an open surface covering only the front of the face, we close it by capping the boundary loops with a skull-shaped cap of 12 concentric rings that smoothly transition from the face boundary to a pole at the back, yielding a watertight mesh of 901 vertices and 1,798 faces. We then apply BFS-based orientation correction (Jacobson et al., 2018) to ensure consistent face winding across the original and capped regions. Each closed mesh undergoes 120 steps of Willmore flow with $\tau_{\min} = 0.005$, $\tau_{\max} = 0.060$, and $p = 1.2$.

4.2 BASELINES

To isolate the effect of the transport path itself, we compare against two baseline transport paths: a Gaussian noise process and the Sphere-Sweep deformation procedure[2]. No existing mesh dataset or generative method allows this comparison directly: standard 3D generative benchmarks vary architecture, data, and evaluation protocol simultaneously, making it difficult to attribute any performance difference specifically to the choice of transport path. We therefore construct two baselines that replace only the forward transport path, while holding the dataset splits, network architecture, training procedure, and early-stopping criterion (based on validation loss) identical to our method. To construct training data, we generate sequences of meshes over $T = 120$ steps using the corresponding baseline transport process. This lets us test directly how much a geometry-aware transport path matters for learning mesh deformations.

Gaussian Noise Baseline. As a first baseline, we consider a diffusion-style forward process that progressively adds isotropic Gaussian noise to vertex positions. At each step i, noise is sampled for each vertex coordinate with standard deviation $\sigma(i)$, following a linear schedule.

The noise scale is chosen separately for each dataset (face and hand), as their geometric structures exhibit different sensitivity to perturbations. In all cases, we use the smallest noise levels that produce

[2]We also evaluated straight-line interpolation toward a radially projected sphere as an additional transport path. In practice, this construction proved unusable: intermediate meshes rapidly developed severe self-intersections before training could begin, and no analogous tuning parameter was available to avoid these degeneracies. We therefore excluded this transport path from subsequent experiments.

visible deformation while avoiding immediate degeneration of the mesh. Despite this careful tuning, the independent per-vertex perturbations do not respect the geometric coupling between neighboring vertices, and the resulting meshes quickly develop self-intersections and irregular artifacts.

Models trained on these sequences show only limited improvement in validation loss, enough to recover coarse overall shape but not fine-grained geometric detail. Training was terminated using early stopping based on validation loss.

Sphere-Sweep Baseline. We employ an empirical geometric inflation process that deforms a mesh toward a sphere, with the construction tailored specifically to hand anatomy. Unlike Gaussian perturbations, Sphere Sweep produces clean intermediate meshes without self-intersections while continuously transforming the geometry toward a simple base shape.

The method begins by identifying the palm region via PCA on the hand mesh — separating it from the fingers — and inscribing a sphere centered within this region; the sphere then expands toward a bounding sphere, propagating displacements to neighboring vertices through an elasticity-based smoothing mechanism. However, because the sphere expands outward from the center, the vertices are affected at different times depending on their distance from the center. This results in spatially and temporally uneven deformation, where neighboring vertices may undergo significantly different displacements within a single step.

Despite its heuristic nature, Sphere-Sweep represents a reasonable geometry-aware baseline, and the trained models exhibit improved optimization behavior relative to the Gaussian Noise baseline. However, unlike the geometric curvature-dependent flows, this approach needs adjustments for each dataset.

4.3 Implementation Details

We implement our model using a graph neural network operating on mesh vertices. All experiments use Chebyshev graph convolutions (Defferrard et al., 2016) with polynomial order $K = 5$. The network consists of 8 convolutional layers with channel sizes $[128, 256, 256, 512, 512, 512, 512, 512]$, each followed by ReLU activation and dropout ($p = 0.1$). The convolutional backbone is followed by 5 fully connected layers with dimensions $[512, 512, 128, 128, 16]$ and a final linear layer that maps to 3 output dimensions, producing per-vertex displacement vectors.

We train all models using the Adam optimizer with a cosine annealing learning rate schedule. All models are trained until convergence, using early stopping based on validation loss to prevent overfitting. The training objective minimizes the mean squared error between predicted and ground-truth per-vertex displacements.

4.4 Evaluation Metrics

We evaluate the quality of generated meshes using five complementary metrics. Since all meshes in our setting share a fixed triangulation (the same vertices and faces), vertex-wise correspondence is available wherever applicable. This enables direct evaluation without alignment or surface sampling and reduces variance in metric estimation.

Vertex Error (VE). We compute the mean per-vertex Euclidean distance between predicted and ground-truth meshes: $\mathrm{VE} = \frac{1}{N}\sum_{i=1}^{N} \|v_i^{\mathrm{pred}} - v_i^{\mathrm{gt}}\|_2$. This metric directly measures reconstruction accuracy at the vertex level.

Chamfer Distance (CD). To facilitate comparison with 3D generative modeling benchmarks (Yang et al., 2019; Gupta and Chandraker, 2020), we report the symmetric Chamfer distance (CD-L2) between predicted and ground-truth vertex sets $S^{\mathrm{pred}} = \{v_i\}_{i=1}^{N}$ and $S^{\mathrm{gt}} = \{u_i\}_{i=1}^{N}$:

$$\mathrm{CD} = \frac{1}{|S^{\mathrm{pred}}|}\sum_{v\in S^{\mathrm{pred}}} \min_{u\in S^{\mathrm{gt}}} \|v-u\|_2^2 + \frac{1}{|S^{\mathrm{gt}}|}\sum_{u\in S^{\mathrm{gt}}} \min_{v\in S^{\mathrm{pred}}} \|u-v\|_2^2.$$

Since all meshes share a fixed triangulation, we compute CD on mesh vertices rather than uniformly sampling points on the surface.

Table 1: Reconstruction metrics on the **MANO hand** dataset (779 vertices). VE and CD are multiplied by 10^3; NC ranges from 0 to 1; CE is mean absolute curvature difference.

Method	VE ($\times 10^3$) ↓	CD ($\times 10^3$) ↓	NC ↑	CE ↓
Gaussian Noise	113.37	11.30	0.612	5.166
Sphere-Sweep	**17.31**	**1.09**	**0.995**	**0.626**
GFM (Ours)	46.41	3.49	0.985	1.545

Table 2: Reconstruction metrics on the **face meshes generated from MediaPipe Face Mesh** (901 vertices). Same metric conventions as Table 1.

Method	VE ($\times 10^3$) ↓	CD ($\times 10^3$) ↓	NC ↑	CE ↓
Gaussian Noise	50.42	5.09	0.822	3.622
Sphere-Sweep	31.39	2.25	0.990	1.723
GFM (Ours)	**23.88**	**1.36**	**0.990**	**1.506**

Table 3: Mesh quality comparison, reported as absolute deviation from ground truth.

	Hand			Face		
Method	ΔSI (%)↓	ΔEdge Ratio↓	ΔAspect Ratio↓	ΔSI (%)↓	ΔEdge Ratio↓	ΔAspect Ratio↓
Gaussian Noise	30.77	0.29	14.17	3.50	0.05	1.00
Sphere-Sweep	0.41	**0.01**	0.11	0.30	0.04	0.10
GFM (Ours)	**0.26**	0.16	**0.04**	**0.12**	**0.01**	**0.06**

Normal Consistency (NC). We compute the mean cosine similarity between per-vertex normals of the predicted and ground-truth meshes: $\mathrm{NC} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{n}_i^{\mathrm{pred}} \cdot \hat{n}_i^{\mathrm{gt}}\right|$, where $\hat{n}_i$ denotes the unit normal at vertex i, obtained as the area-weighted average of adjacent face normals. This metric captures local surface orientation fidelity.

Curvature Error (CE). We compute the mean absolute difference in discrete mean curvature between predicted and ground-truth meshes: $\mathrm{CE} = \frac{1}{N}\sum_{i=1}^{N}|H_i^{\mathrm{pred}} - H_i^{\mathrm{gt}}|$, where the mean curvature vector is given by $\mathbf{H} = M^{-1}L\mathbf{v}$, and $H_i = \frac{1}{2}|\mathbf{H}_i|$ denotes its magnitude at vertex i. Here, L is the cotangent Laplacian and M is the Voronoi mass matrix (Meyer et al., 2003). This metric measures surface curvature preservation and is particularly relevant to our curvature-driven transport.

Self-Intersection Rate (SI). We report the percentage of faces that participate in at least one intersection with a non-adjacent face. Two faces are considered adjacent if they share a common edge or vertex and are excluded from intersection testing. Intersections are detected using axis-aligned bounding box (AABB) culling followed by triangle-triangle intersection tests on candidate face pairs. This metric quantifies mesh integrity.

4.5 Quantitative Results

Tables 1 and 2 present reconstruction metrics for the MANO hand and MediaPipe face datasets, respectively. Our GFM formulation is compared against both baselines using the same architecture and training procedure.

The Gaussian noise baseline fails to learn a meaningful reverse process, as independent per-vertex perturbations destroy the geometric structure of the mesh. The Sphere-Sweep baseline produces valid intermediate meshes, but its deformation remains spatially non-uniform and often stays close to the target geometry, which can simplify the reconstruction task while limiting the strength of the underlying transformation compared to our GFM formulation. Sphere-Sweep's construction is hand-specific, built around identifying a palm region via PCA; this notion of a center does not generalize to other shapes without redesign, whereas GFM's use of local curvature requires no such re-engineering across categories. This difference is reflected in the results: Sphere-Sweep achieves the best reconstruction accuracy on the MANO hand dataset (Table 1), while GFM outperforms both baselines on three of the four metrics on the face dataset and matches Sphere-Sweep on NC (Table 2). Table 3 reports mesh quality metrics. Our method preserves mesh integrity throughout generation, with self-intersection rates comparable to the ground truth.

Table 4: Contextual comparison with mesh reconstruction methods reported in Neural Mesh Flow (Gupta and Chandraker, 2020); not directly comparable across rows (see discussion in Sec. 4.5).

Method	Task	CD ($\times 10^3$) ↓	NC ↑	SI (%) ↓
AtlasNet (Groueix et al., 2018)	AE	4.15	0.815	24.80
AtlasNet-O + Laplacian	AE	3.81	0.811	2.02
Neural Mesh Flow (Gupta and Chandraker, 2020)	AE	5.54	0.826	0.10
NMF + Laplacian	AE	5.25	0.826	0.00
Pixel2Mesh (Wang et al., 2018)	SVR	5.48	0.706	12.29
OccNet-3 (Mescheder et al., 2019)	SVR	8.33	0.814	0.09
GFM – Hand (Ours)	Flow Recon	3.49	0.985	0.21
GFM – Face (Ours)	Flow Recon	1.36	0.990	0.12

Table 4 places our results alongside values reported in Neural Mesh Flow (Gupta and Chandraker, 2020), which evaluated AtlasNet, Pixel2Mesh, and OccNet using their pretrained implementations. Because these methods are evaluated on different datasets, object categories, and generation settings, the comparison is not intended as a ranking of methods. In particular, our hand ground-truth meshes contain some inherent self-intersections due to finger contact (SI = 0.47), making SI values for the hand dataset not directly comparable with those reported in prior work. In contrast, our face ground-truth meshes contain essentially no self-intersections (SI $\approx$ 0.00). The table therefore serves as an external reference for interpreting the scale of reconstruction and mesh-quality metrics reported in prior mesh-generation work, complementing the controlled comparisons in the preceding experiments.

4.6 Qualitative Results

Figure 2 and Figure 3 show representative reconstruction trajectories for our GFM model, Gaussian noise, and Sphere-Sweep, starting from an early point in the flow (step 2 of 120) for both the hand and face datasets. GFM produces a clean, self-intersection-free surface at every step and converges closely to the ground truth on both datasets. Gaussian noise never recovers a valid surface, remaining severely self-intersecting and geometrically jagged even at the final step, consistent with its poor quantitative performance in Tables 1–2. Sphere-Sweep produces a reasonable hand reconstruction, while the face reconstruction is less accurate, showing more distortion and asymmetry.

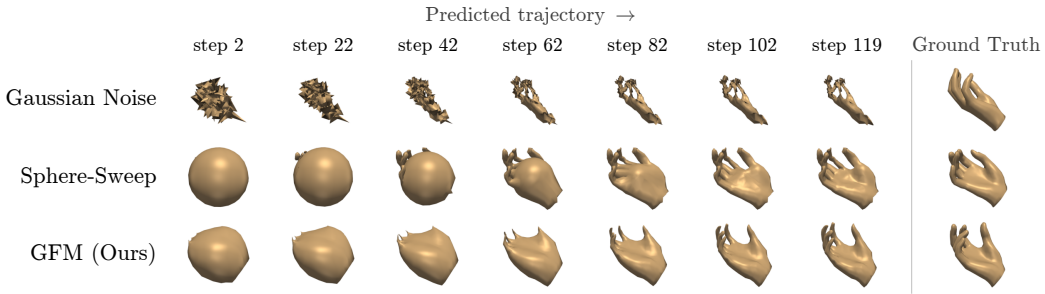


Figure 2: Qualitative comparison of hand reconstructions produced by GFM and the baseline methods.

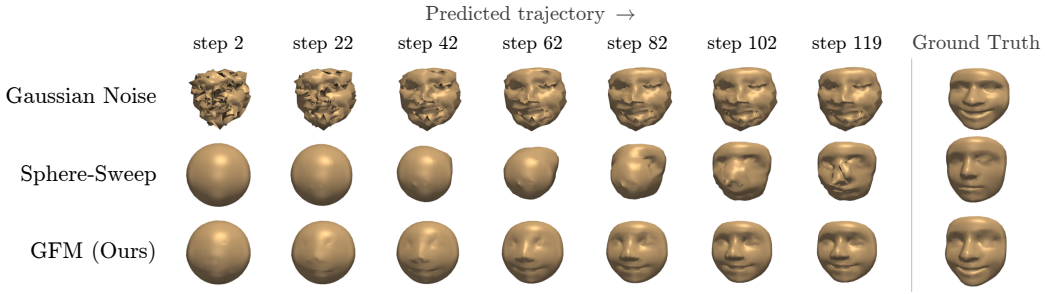


Figure 3: Qualitative comparison of face reconstructions produced by GFM and the baseline methods.

4.7 Sampling from a Continuous Base Distribution

As described earlier, our main experiments use an empirical base distribution: at inference time, generation starts from held-out near-spherical test meshes obtained through the forward geometric flow rather than from a closed-form prior. Although these base states are unseen during training, each is still obtained by smoothing a particular target mesh.

To test whether generation can instead begin from newly sampled base states that do not correspond to any individual mesh, we fit a continuous distribution to the training base states. Specifically, we fit a low-rank Gaussian via PCA to the Willmore base states of all 2,368 training face identities. Using the top 12 principal components, which capture 99.9% of the variance, we sample coefficients $a_j \sim \mathcal{N}(0, \lambda_j)$ and construct new base states as $\tilde{x}_0 = \bar{x}_0 + \sum_{j=1}^{12} a_j q_j$, where $\bar{x}_0$ is the mean training base state, and q_j and λ_j denote the j-th principal direction and its corresponding eigenvalue, respectively. We additionally consider simple linear interpolation between randomly selected pairs of training base states. In both cases, the resulting meshes are passed directly to the same pretrained GFM reverse network, without retraining or modification.

Among 100 PCA samples, 82% decode to valid meshes, where validity is defined by self-intersection $< 2\%$ and aspect ratio < 2.2, using thresholds matched to those of real reconstructions. Among 25 interpolated samples, 96% are valid. Nearest-neighbor distances to the training set are comparable to those of a held-out set of 200 real identities, suggesting that the generated samples are not simply reproducing training examples, with representative examples shown in Figure 4.

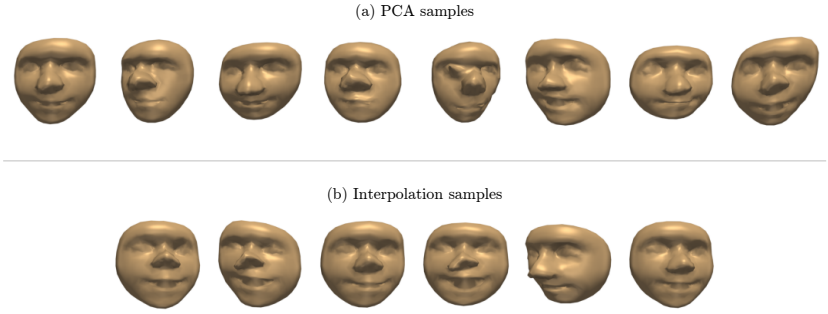


Figure 4: Examples generated from novel base states using GFM pretrained reverse network, with no retraining or fine-tuning. (a) Eight PCA samples using 12 principal components. (b) Six samples from linear interpolation between training base states. Examples were chosen for visual variety among valid outputs. The full set of 125 samples, including failures, is shown in Appendix B.

5 Conclusion

We introduced a geometric flow matching framework for 3D mesh generation, in which transport paths are constructed using geometric flows rather than generic interpolation paths. We train a graph neural network to predict the velocity field, represented as vertex-wise displacements along discretized flow trajectories, enabling the model to map simplified mesh states back to detailed target surfaces. Experiments on hand and face mesh datasets demonstrate that the proposed transport path produces accurate reconstructions while preserving global shape and fine geometric detail.

We show that generation is not limited to empirically held-out base states. Base states sampled from a low-rank Gaussian fit to the training base-state distribution or through linear interpolation between training base states produce valid outputs at high rates (82% and 96%, respectively) when decoded with the pretrained reverse network. Nearest-neighbor analysis shows that these generated samples are no closer to the training set than held-out real identities, suggesting that the learned reverse process generalizes beyond the finite set of precomputed base states.

Constructing Willmore-flow trajectories requires relatively costly offline preprocessing, which future work could address. Other directions include incorporating intrinsic flows such as Ricci flow, which would enable direct manipulation of the metric structure, and extending the framework to meshes with varying topology.

AI USE STATEMENT

Generative AI tools were used as a feedback aid for author-developed experiments, and to assist with editing the manuscript for clarity and presentation. The research ideas, methodology, experimental design, analysis, and conclusions were developed and determined by the authors. All AI-assisted suggestions were critically reviewed, and the authors take full responsibility for the final content of the paper.

ETHICS STATEMENT

This work does not involve the collection of new human-subject data. Our experiments use existing 3D mesh datasets solely for methodological evaluation. We are not aware of any ethical concerns specific to the proposed method beyond those associated with the use of the underlying datasets.

REPRODUCIBILITY STATEMENT

We provide the methodological and experimental details needed to reproduce our results throughout the paper and appendix. This includes network architecture, dataset preprocessing, training procedure, baseline construction, evaluation metrics, and relevant hyperparameters. The codebase is not included with the current submission.

REFERENCES

Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *International Conference on Learning Representations (ICLR)*, 2023.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics*, 32(4), 2013.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorber, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 216–224, 2018.

Kunal Gupta and Manmohan Chandraker. Neural mesh flow: 3d manifold mesh generation via diffeomorphic flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1747–1758, 2020.

Alec Jacobson, Daniele Panozzo, et al. libigl: A simple c++ geometry processing library. *https://libigl.github.io/*, 2018.

Matthew Le, Apoorv Vyas, Bowen Shi, Brian Karber, Leda Sari, Rashel Moritz, Mary Williamson, Vimal Manohar, Yossi Adi, Jay Mahadeokar, and Wei-Ning Hsu. Voicebox: Text-guided multilingual universal speech generation at scale. In *NeurIPS*, 2023.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision*, 2015.

Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hao, George Fan, Valentin Bazarevsky, Matthias Grundmann, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, 2021.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470, 2019.

Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, pages 35–57. Springer, 2003.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *JMLR*, 22(57): 1–64, 2021.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

Adam Polyak et al. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics*, 36(6), 2017.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018.

Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borber, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620:1089–1100, 2023.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*, 2016.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *ICCV*, 2019.

Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se(3) diffusion model with application to protein backbone generation. In *ICML*, 2023.

A Extended Background

A.1 Flows in Riemannian Geometry

Riemannian geometry is a branch of differential geometry that studies Riemannian manifolds. Riemannian geometry provides essential tools for analyzing and solving problems across various disciplines by extending Euclidean concepts such as distance, angle, and volume to curved spaces. Indeed, the Riemannian metric is the key component that allows for this generalization.

Formally, a Riemannian manifold $(\mathcal{M}^n, g)$ is a smooth real n-dimensional manifold equipped with a positive-definite inner product g_p defined on the tangent space T_pM at each point p of the manifold.

The family g_p of inner products is called a Riemannian metric g. The Riemannian metric g is indeed a rank-2 tensor. Tensors are mathematical objects that generalize familiar concepts such as scalars, vectors, and matrices. Just as a scalar is a rank-0 tensor and a vector is a rank-1 tensor, a metric tensor is a unique rank-2 tensor, defining how to compute lengths and angles at each location on the manifold. Like all tensors, the metric can be expressed in terms of its components relative to a local coordinate system. In terms of a coordinate basis $\{\frac{\partial}{\partial x^i}\}$ with dual basis $\{dx^i\}$, the metric tensor g has the local representation $g = g_{ij}dx^i \otimes dx^j$, where g_{ij} are its components in these coordinates [3]. Here, we employ the Einstein summation convention, in which repeated indices are implicitly summed over their ranges. The tensor product $\otimes$ indicates that the metric acts as a bilinear form in the tangent space, taking two tangent vectors and producing a scalar.

Another central concept in Riemannian geometry is the curvature tensor, which describes how a manifold deviates from Euclidean or flat geometry. The Riemann curvature tensor R^i_{jkl} is a rank-4 tensor constructed from the metric and its derivatives. Intuitively, the Riemann tensor measures how much the geometry of a Riemannian manifold is locally different from the geometry of flat Euclidean space. The Ricci tensor R_{ij} is derived by contracting the Riemann tensor on two indices: $R_{ij} = R^k_{ikj}$. The Ricci scalar R is obtained by further contracting the Ricci tensor with the metric: $R = g^{ij}R_{ij} = R^i_i$. These tensors form a natural hierarchy of geometric information, from the most detailed (Riemann) to the most averaged (scalar curvature), each capturing different aspects of the manifold's geometry.

Building on the concept of curvature tensors, geometric flows serve as powerful dynamical tools in Riemannian geometry, providing insights into a manifold's underlying topology, geometry, and intrinsic or extrinsic properties. These evolutionary processes have become fundamental techniques for analyzing and transforming the geometric and topological structure of manifolds. A geometric flow describes the evolution of a geometric quantity over time in a manner governed by a partial differential equation. One prominent example is the Ricci flow, introduced by Richard Hamilton in 1982. Ricci flow deforms the initial metric in proportion to its Ricci curvature, effectively smoothing the manifold over time. Ricci flow achieved widespread recognition when Grigori Perelman employed it, alongside advanced techniques, to solve the Poincaré conjecture, one of the most celebrated problems in topology.

Ricci flow is formally defined on the Riemannian manifold $(\mathcal{M}^n, g)$. Under this flow, the Riemannian metric g_{ij} evolved as follows:

$$\partial_t g_{ij} = -2R_{ij} \tag{6}$$

where R_{ij} is the Ricci tensor. The parameter t is typically interpreted as an additional dimension that represents time. In this paper, we work with Ricci flow on 2-dimensional closed manifolds, i.e., surfaces without holes, so we set $n = 2$ and the Euler characteristic $\chi = 2$.

In many situations, the Ricci flow can shrink the space over time. To address this, the normalized Ricci flow is often used, which preserves the volume of the space. The equation of normalized Ricci flow for two-dimensional manifolds is

$$\partial_t g_{ij} = -2R_{ij} + rg_{ij} \tag{7}$$

where r is the average of Ricci scalar over the manifold. Defining ν as the measure of the manifold, r is:

$$r = \frac{\int_{M^2} R d\nu}{\int_{M^2} d\nu} = \frac{4\pi\chi}{A} = \frac{8\pi}{A} \tag{8}$$

where A is the total surface of the manifold. In addition, the curvature of a 2-dimensional surface is completely encoded in its Ricci scalar curvature, and the Ricci tensor is simplified to $R_{ij} = \frac{1}{2}Rg_{ij}$. Therefore, the Normalized Ricci flow for a two-dimensional manifold can be written as:

$$\partial_t g_{ij} = (r - R)g_{ij} \tag{9}$$

According to the Uniformization Theorem, there exists a unique solution $g(t)$ for the equation (9). If we assume the initial metric is $g(0) = g_0$, as $t \to \infty$, the metrics $g(t)$ converge uniformly in any C^k norm to a smooth metric g_∞ of constant curvature, which in our setting will be the metric of a sphere.

[3] From now on, we are demonstrating each tensor by its components, instead of using abstract notation

In 2D, the Ricci flow evolves the metric within its conformal class, meaning the metric tensor at each time can be written as $g_t = e^{2\phi_t} g_0$ where g_0 is a fixed or canonical background metric in the same conformal class. If we assume the manifold has no genus (Euler characteristic $\chi = 2$), then g_0 represents the metric of a sphere. In this case, it can be shown that the equation (9) reduces to a heat equation for the conformal factor ϕ:

$$\frac{\partial \phi}{\partial t} = r - R_0 + 2\Delta_0 \phi \tag{10}$$

where Δ_0 is the Laplace-Beltrami operator with respect to the metric g_0. Assuming the background metric g_0 is a standard round metric (sphere with radius $L = 1$), then $R_0 = 2$, and above relation turns to:

$$\frac{\partial \phi}{\partial t} = 2\Delta_0 \phi \tag{11}$$

Using numerical techniques and discretizing the Laplace-Beltrami operator, one can solve this equation. Solving for ϕ provides a direct representation of how the conformal class of metrics evolves through the normalized Ricci flow. In other words, this equation explicitly describes how metrics in the conformal class of g are transformed into one another through the Ricci flow. The result at $t \to \infty$ represents the canonical or back ground metric g_0 in the conformal class with constant curvature.

It is also worth mentioning that if we start from a perfect sphere and run the normalized Ricci flow or even its reverse, we do not get any development. In other words, the sphere is a fixed point of the normalized Ricci flow because the right-hand side of the equation (9) is equal to zero in this case.

While intrinsic geometry, as exemplified by Riemann curvature, focuses on the properties inherent to the manifold itself, extrinsic geometry examines how a manifold is embedded within an ambient space. A fundamental concept in extrinsic geometry is principal curvature κ_1 and κ_2, which are extrinsic quantities measuring how a surface bends in specific directions within an ambient space. Mean curvature H, is defined for a surface (2D manifold) as the average of the principal curvatures κ_1 and κ_2, as $H = \frac{\kappa_1 + \kappa_2}{2}$. Mean curvature plays a crucial role in various geometric analyses, including the study of minimal surfaces.

Mean curvature flow evolves a smooth, closed surface $M \subset \mathbb{R}^3$ by moving each point inward at a speed equal to the local mean curvature. one represents the surface by an immersion $f(\cdot, t) : \mathcal{M} \to \mathbb{R}^3$. The flow is then given by the geometric PDE:

$$\frac{\partial f}{\partial t} \;=\; -\,H\,\mathbf{n}, \tag{12}$$

where H is the mean curvature and $\mathbf{n}$ is the unit normal[4]. Intuitively, this equation states that sharper regions (high H) move faster, causing the surface to shrink and smooth out over time. Having the identity $H\mathbf{n} = -\frac{1}{2}\Delta f$, equation (12) can be written as:

$$\frac{\partial f}{\partial t} \;=\; \frac{1}{2}\Delta f. \tag{13}$$

Like the Ricci flow, mean curvature flow also has a normalized variant, known as the volume-preserving mean curvature flow, which, under suitable conditions (e.g., strict convexity), evolves the surface to a round sphere. Although mean curvature flow and Ricci flow are distinct flows (one extrinsic and the other intrinsic), defined based on different quantities, note the similarity in structure between equations (13) and (11). As mentioned earlier, there exist multiple geometric flows that can be categorized in various ways. For example, flows such as the Ricci flow and mean curvature flow tend to smooth out the geometry of the manifold, reducing irregularities and often converging to simpler, more canonical forms. However, Ricci flow is an intrinsic flow defined solely by the intrinsic curvature of a manifold and does not require any embedding in a higher-dimensional space. In contrast, the mean curvature flow is an extrinsic flow, defined based on the extrinsic curvature, and involves manifolds embedded in $\mathbb{R}^n$.

[4]Note that the immersion or position function f is indeed a R^3 or vector-valued function. Therefore, (12) is a vectorial equation, and we apply the Laplacian component-wise.

Willmore flow is an extrinsic geometric flow defined in terms of the mean curvature H, which deforms a surface to reduce its bending energy. In the discrete setting, conformal Willmore flow is introduced, harnessing both the intrinsic and extrinsic properties of manifolds. By combining these features, conformal Willmore flow achieves more stable and predictable smoothing of surfaces. Mainly because it restricts surface deformations to conformal transformations, thereby smoothing out bends without introducing shear distortions.

In many geometrical flows, finite or infinite-time singularities can arise, so smoothing is not guaranteed unconditionally. However, depending on the flow, the dimension of the manifold, and assumptions such as convexity or curvature bounds, there are theorems ensuring the existence (and sometimes uniqueness) of solutions over certain time intervals, or even globally. For example, for a compact 2D manifold without boundary, the uniformization Theorem states that by flowing any initial metric under the normalized Ricci flow, one obtains a smoothly evolving family of metrics converging to the constant curvature uniformizing metric. There is a similar theorem, known as Huisken's Theorem, for the normalized mean curvature flow of closed, smooth, strictly convex 2D manifolds embedded in $\mathbb{R}^3$, which guarantees that the flow exists for all time and converges to a round sphere.

Conformal transformations frequently appear in the study of geometric flows on two-dimensional manifolds, emphasizing their importance as essential tools for surface fairing and smoothing. For example, in two dimensions, Ricci flow can be expressed as conformal changes to the metric, progressively achieving uniform curvature. Similarly, conformal Willmore flow in discrete geometry exemplifies how such transformations enable stable, distortion-free smoothing processes. Consequently, conformal transformations play a foundational role in geometric analysis, facilitating the refinement of surfaces.

A.2 Flow-based generative models

The concept of flow arises naturally in the study of vector fields. Given a smooth vector field on a manifold, its integral curves are the smooth curves whose velocity at each point coincides with the vector field at that point. Together, these integral curves generate a one-parameter family of diffeomorphisms on the manifold, known as a flow.[5] The flow $\varphi : \mathcal{N} \times \mathbb{R} \to \mathcal{N}$ takes a point x on the manifold $\mathcal{N}$ and a time $t \in \mathbb{R}$, and returns $\varphi(x, t)$ i.e., where x flows to at time t. The velocity field u is associated to the flow φ by the ODE

$$\frac{d\varphi(x,t)}{dt} = u(\varphi(x,t), t). \tag{14}$$

The flow $\varphi(x, t)$ satisfies the group properties:

$$\varphi(x, 0) = x \tag{15}$$

$$\varphi(\varphi(x,t), s) = \varphi(x, t+s). \tag{16}$$

Furthermore, instead of working with φ, we can introduce a family of maps $\varphi_t : \mathcal{N} \to \mathcal{N}$. Additionally, for convenience, we assume that the parameter t is restricted to the interval $[0, 1]$. One can go further and define $\mathbf{x}(t) = \varphi(x_0, t)$, meaning that the trajectory $\mathbf{x}(t)$ is obtained by fixing one initial point x_0 and letting time vary. It can also be shown equivalently as $\mathbf{x}_t = \varphi_t(\mathbf{x}_0)$. Then, we can rewrite the flow equation (14) with the features 15 as

$$\frac{d\mathbf{x}(\mathbf{t})}{dt} = u(\mathbf{x}(t), t) \qquad \mathbf{x}(0) = \varphi(x_0, 0) = x_0. \tag{17}$$

Having defined a diffeomorphism or map φ on the manifold, we may ask how other geometrical quantities are transformed under this map. The short answer is we push forward any covariant quantity along the flow of the map, and we pull back any contravariant quantity. The push-forward operation is usually demonstrated by $\varphi_\sharp$, and the pull-back operation by $\varphi^\sharp$. For example, the probability distribution at time t is obtained by the push-forward operation, given by $p_t = \varphi_{t\sharp} p_0$, which with the assumption of u being a locally Lipschitz, it can be written as

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot (p_t(\mathbf{x}) u_t(\mathbf{x})) = 0. \tag{18}$$

[5]Note that this notion of flow differs from the geometric flow that has been defined in the previous section. Here, the flow acts on a fixed background manifold, whereas a geometric flow deforms the manifold itself.

This equation, called the continuity equation, ensures the transformation satisfies the conservation of probability [6]. In the following, we explain how the concept of flow, ODE given in (14) or equivalently (17), and continuity equation (18) are used in generative models like Continuous Normalizing Flows (CNF), Flow Matching (FM) and Conditional Flow Matching (CFM). Being only interested in generative models in Euclidean space, we set $\mathcal{N} = \mathbb{R}^d$.

Assume that we have a finite number of samples, represented by $\mathbf{x}_{data}$, with the probability distribution $p_{data}(x)$ over them. The goal of generative modeling is to sample more from the distribution p_{data} that is unknown to us. The approach of flow-based models for solving this problem is to use a flow φ_t that transfers samples from a base distribution p_0 into a new distribution p_1 that closely approximates the target distribution p_{data}. In other words, in this framework, a random variable $\mathbf{x}_0 \sim p_0$ is transformed such that $\varphi_{t=1}(\mathbf{x}_0) = \mathbf{x}_1$, and the resulting distribution p_1 aligns as closely as possible with the desired target distribution $p_{data}(x)$.

Continuous Normalizing Flows (CNF) is one of the first flow-based generative models, whose training is based on maximizing the log-likelihood of available samples. Considering both the flow ODE and continuity equation, we can rewrite the continuity equation in log-density form:

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{x}(1) + \int_1^0 u_\theta(\mathbf{x}(t), t)dt \\ \log p_1(\mathbf{x}_1) &= \log p_0(\mathbf{x}_0) - \int_0^1 \nabla \cdot u_\theta(\cdot, t)(\mathbf{x}(t))dt \end{aligned} \tag{19}$$

with the boundary conditions $\mathbf{x}(1) = \mathbf{x}_{data}$, and $\log p_1(\mathbf{x_1}) = \log p_1(\mathbf{x}_{data})$. By solving these two equations simultaneously, we can obtain both $\mathbf{x}_0$ and $\log p_1(\mathbf{x}_{data})$[7]. Standard maximum likelihood training proceeds by computing the log-likelihood gradient with respect to θ. However, this computation often faces scalability challenges in high dimensional spaces, mainly due to the need to compute the divergence term. To overcome this, Flow Matching (FM) (Lipman et al., 2023) was proposed as a simulation-free approach to train the neural network. Instead of likelihood maximization, FM learns the optimal parameter θ^* of the parametric velocity u_θ by solving the regression problem with the loss function:

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\mathbf{x}_t\sim p_t}\left[\|u_\theta(\mathbf{x}_t, t) - u(\mathbf{x}_t, t)\|^2\right] \tag{20}$$

However, since the target probability distribution p_{data} is unknown to us, pre-selecting a probability path or velocity field is not a trivial task.

Conditional Flow Matching (CFM) (Lipman et al., 2023) extends FM by introducing a conditioning variable z that modulates the probability paths over time. The probability path can be written as a marginal over a joint involving the conditional variable:

$$p_t(x_t) = \int p(z)p_t(x_t \mid z)dz = \mathbb{E}_z\big[p_t(x \mid z)\big]. \tag{21}$$

with boundary conditions

$$\begin{aligned} p_0(x) &= \mathbb{E}_z\big[p_0(x \mid z)\big] \\ p_{data}(x) &= \mathbb{E}_z\big[p_1(x \mid z)\big] \end{aligned} \tag{22}$$

for $\forall x$. It can be shown that the associated conditional velocity field $u_t^{\text{cond}}(\mathbf{x}, z)$ also satisfy:

$$u_t(x) = \mathbb{E}_{z|x,t}\big[u_t^{\text{cond}}(\mathbf{x}, z)\big], \tag{23}$$

The conditional probability path also satisfies the continuity equation written in terms of the corresponding conditional velocity field $u_t^{cond}(\mathbf{x}, z)$:

$$\frac{\partial p_t(x \mid z)}{\partial t} + \nabla_x \cdot \big(p_t(x \mid z)\, u_t^{cond}(\mathbf{x}, z)\big) = 0, \tag{24}$$

[6] Observing that the divergence of a vector field is equal to the trace of its Jacobian shows that the continuity equation is the continuous analog of the change-of-variables formula

[7] Please note that we have assumed a closed form for p_0.

For now, assume we have a way to compute or have the conditional velocity $u_t^{cond}(\mathbf{x}, z)$, satisfying all the mentioned equations. The conditional flow matching loss function is defined as:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\mathbf{x}_t\sim p_t(x|z),z\sim p(z)}\left[\|u_\theta(\mathbf{x}_t,t) - u_t^{\text{cond}}(\mathbf{x},z)\|^2\right], \tag{25}$$

ensuring that the learned velocity field $u_\theta(\mathbf{x}_t, t)$ closely approximates the conditional velocity field $u_t^{\text{cond}}(\mathbf{x}, z)$. Here, the key feature is that CFM training achieves the same optimization objective as standard flow matching; namely the gradient of the conditional flow matching satisfies the equivalence

$$\nabla_\theta \mathcal{L}_{CFM}(\theta) = \nabla_\theta \mathcal{L}_{FM}(\theta). \tag{26}$$

Once the velocity field $u_\theta(\mathbf{x}, t)$ is learned, new samples can be generated from the target distribution p_1. The process involves sampling an initial point $\mathbf{x}_0 \sim p_0$ and evolving it according to the learned velocity field $u_\theta(\mathbf{x}, t)$ using an ODE solver, such as the forward Euler method:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + u_\theta(\mathbf{x}, t)\Delta t \tag{27}$$

until $t = 1$. This procedure yields a sample $\mathbf{x}_1$ that follows the target distribution p_1.

In the remainder of this section, we review the key design choices within the CFM framework. These involve two key components: the *conditional probability path*, which governs the shape of the interpolation between source and target, and the *coupling*, which determines how source and target samples are paired (Tong et al., 2024). Together, these components determine the structure of the learned dynamics and the complexity of the resulting generative model.

Conditional probability paths. A widely adopted family assumes Gaussian conditional paths of the form (Lipman et al., 2023; Albergo and Vanden-Eijnden, 2023)

$$p_t(x \mid z) = \mathcal{N}\big(\mu(z,t),\ \sigma^2(z,t)\, I\big), \tag{28}$$

with corresponding conditional velocity field:

$$u_t^{\text{cond}}(x,z) = \frac{\sigma'(z,t)}{\sigma(z,t)}\big(x - \mu(z,t)\big) + \mu'(z,t). \tag{29}$$

A common choice is linear interpolation between (x_0, x_1):

$$\mu(z,t) = (1-t)x_0 + tx_1,$$

which yields straight-line trajectories in the low-noise limit.

Couplings. The coupling determines how source and target samples are paired during training. In the CFM framework, this pairing is specified through the choice of conditioning variable z and its distribution $p(z)$. Common choices include:

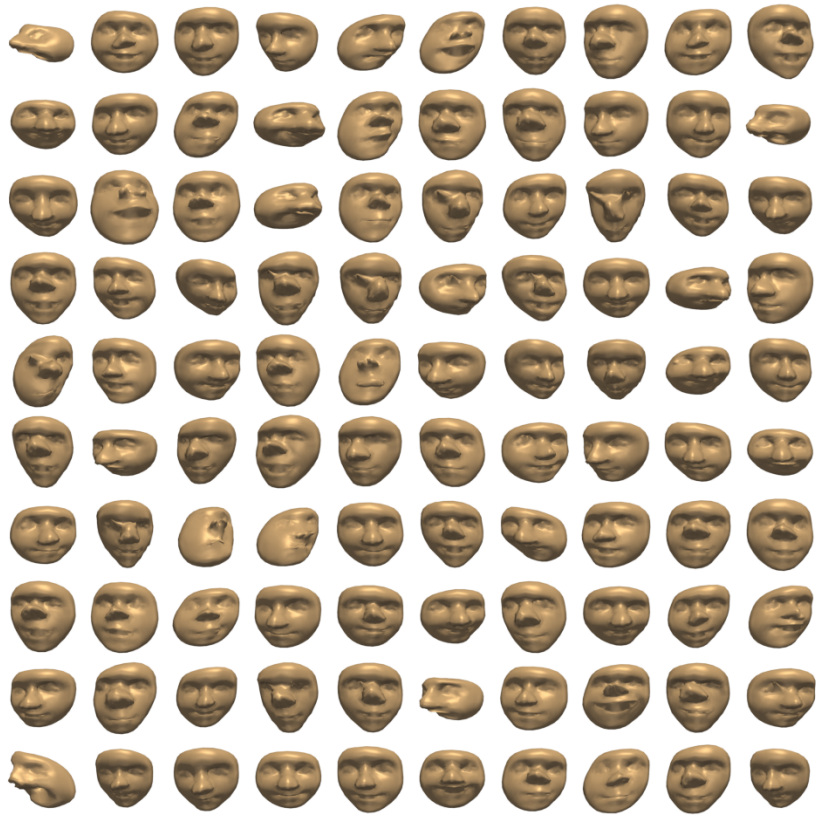
1. **Single-variable conditioning:** $z = x_1$, with $x_1 \sim p_{\text{data}}$. In this case, source samples are drawn independently from p_0, implicitly defining an independent pairing between x_0 and x_1.
2. **Independent coupling:** $z = (x_0, x_1)$ with $p(z) = p_0 \times p_{\text{data}}$, where source and target samples are drawn independently.
3. **Optimal transport coupling:** $z = (x_0, x_1)$, where pairs are sampled from an optimal transport plan $\pi \in \Pi(p_0, p_{\text{data}})$, producing more structured correspondences (Tong et al., 2024).

B Complete PCA and Interpolation Sample Gallery

Figure 5 shows the complete set of samples from the experiment in Section 4.7: all 100 PCA-sampled base states and all 25 interpolated base states, decoded by the same pretrained GFM reverse network. This includes the 18 PCA failures and one interpolation failure reported in the main text, visible as

collapsed or elongated shapes, alongside the valid outputs illustrated in Figure 4. We note that self-intersection and aspect ratio do not capture every failure mode: close inspection of the numerically valid samples reveals a further subset with a localized nose/mouth-region artifact not reflected in these thresholds.

(a) PCA samples (100 total)



(b) Interpolation samples (25 total)



Figure 5: Complete set of all 125 samples from novel base states using the unmodified, already-trained reverse network of GFM. The set includes 100 PCA samples with 12 principal components and 25 samples obtained by linear interpolation between training base states. No samples were removed based on visual quality, and failure cases are included.