# Explicit flows for implicit surfaces

CAMILLE BUONOMO, CNRS, Université Lyon 1, INSA Lyon, France
JULIE DIGNE, CNRS, Université Lyon 1, INSA Lyon, France and LIRIS, France
RAPHAËLLE CHAINE, Université Lyon 1, CNRS, INSA Lyon, France and LIRIS, France

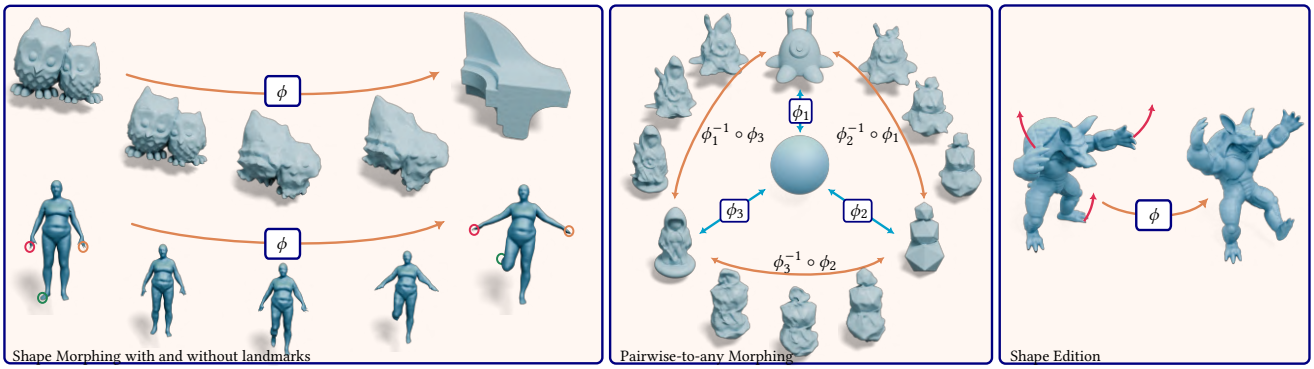


Fig. 1. We provide a flow construction allowing to solve various problems in Geometry Processing: interpolation with and without landmarks (left); pairwise-to-any morphing (middle) where we provide pairwise morphings to a canonical shape (here a sphere) and get for free all other morphings; and shape editing (right).

Shape deformation for morphing or editing purposes is a central challenge in Computer Graphics. While numerous methods exist, few allow for the explicit evaluation of the deformation at arbitrary times and locations directly, without resorting to intricate advection or interpolation schemes. In this paper, we propose a method that provides an explicit expression of the deformation parameterized as a flow, for continuously deforming shapes defined implicitly. Implicit surfaces are indeed particularly well suited for deformation tasks, since they inherently account for both the surface and the enclosed volume. Our approach leverages invertible neural networks to ensure theoretically that the deformation is a valid flow, while also providing differential quantities useful for geometric regularization. We demonstrate applications of this flow to shape morphing with and without landmarks, shape editing, and pairwise-to-any morphing where we compute pairwise morphings to a canonical shape allowing to deduce transformations between any pair through flow composition. The code for our method is available at https://github.com/camillebnm/explicit_flows_for_implicit_surfaces.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; **Animation**; **Neural networks**.

Additional Key Words and Phrases: Flows, Shape Morphing, Shape Editing

**ACM Reference Format:**
Camille Buonomo, Julie Digne, and Raphaëlle Chaine. 2026. Explicit flows for implicit surfaces. *ACM Trans. Graph.* 45, 4, Article 115 (July 2026), 17 pages. https://doi.org/10.1145/3811331

Authors' Contact Information: Camille Buonomo, CNRS, Université Lyon 1, INSA Lyon, Lyon, France, camille.buonomo@liris.cnrs.fr; Julie Digne, CNRS, Université Lyon 1, INSA Lyon, Lyon, France and LIRIS, Lyon, France, julie.digne@cnrs.fr; Raphaëlle Chaine, Université Lyon 1, CNRS, INSA Lyon, Lyon, France and LIRIS, Lyon, France, raphaelle.chaine@liris.cnrs.fr.


ACM 1557-7368/2026/7-ART115
https://doi.org/10.1145/3811331

## 1 Introduction

Shape morphing and deformation are fundamental tasks of Computer Graphics and Geometry Processing. They can be framed as the problem of determining intermediate shapes between a source shape and either a target shape (for shape morphing), or a few known pointwise displacements (for shape editing), or even a combination of both (for morphing a source shape into a target shape respecting some – possibly sparse – known correspondences). These problems are fundamentally ill-posed, given that different valid intermediate steps may exist. Hence, it is necessary to add some constraints on the deformation (e.g. as-rigid-as-possible [Sorkine and Alexa 2007], least displacement [Rabin et al. 2011], volume preservation [Buonomo et al. 2025]).

In this paper we are interested in preserving the structure of the shape, ensuring that no unintended connected components or topological features, such as handles or holes, are introduced during the deformation. This inherently assumes that the source and target shapes share a common structure, specifically they should have the same topology. While diffeomorphisms preserve such structure, with the added benefit of being invertible, they only establish a direct mapping between source and target, while we need a progressive, invertible deformation as intermediate shapes are essential to our approach. Instead, we look for this mapping in the form of a mathematical flow, a space-and time-parameterized mapping, that defines a trajectory for each point of the shape. Mathematical flows indeed present significant advantages, as they satisfy the group law for composition. This property is essential, for guaranteeing that the inverse flow remains fully consistent with the forward flow. To do so, our flow definition leverages invertible neural networks combined with time-parameterization, producing directly the flow-transported points without resorting to computationally expensive

discrete Euler schemes. A key advantage of our definition is its efficiency: Both forward and inverse flows can be computed for arbitrary time intervals $t$ at the exact same cost, and their derivatives can be obtained with the same computational efficiency. These flows can then be optimized to deform a shape into another, using their implicit representations.

To summarize, our key contributions are as follows:

- A method that computes an exact mathematical flow using invertible neural networks, enabling the equally efficient computation of both the flow and its inverse, and similarly for their derivatives.
- A supervision of the explicit mapping leveraging the shape implicit representation, without resorting to time stepping methods, allowing us to compute at a low cost implicit representations for the deformed shapes at arbitrary times.
- Applications to shape morphing with and without landmarks, shape transitions through a canonical shape, and shape editing.

## 2 Related Work

Deforming shapes can be considered a specific instance of deforming distributions. To provide context, we begin by reviewing recent advances in flow-based generative models, then circle back to address the specifics of shapes defined by implicit surfaces and their deformation processes

*Mappings in generative models.* Data generation can often be cast as learning a mapping between a basic, easy to sample, distribution and the empirical distribution one wants to mimic. To ensure the input distribution can be pushed forward through the mapping, the mapping itself must be a diffeomorphism. This led to the development of invertible neural architectures that allow exact likelihood computation of the optimized mapping. These architectures may differ in their expressiveness as detailed in a comprehensive survey on invertible networks [Kobyzev et al. 2021]. Among the proposed architectures, some approaches model the mapping by combining a series of coupling layers that are invertible by design: each coupling layer splits the components of its input vector into two separate blocks. The layer modifies only the components of the first block by an invertible analytical function of the components in the second block, thus ensuring the invertibility. Many coupling functions have been proposed [Dinh et al. 2015],[Dinh et al. 2017],[Ziegler and Rush 2019], [Ho et al. 2019], [Müller et al. 2019], [Durkan et al. 2019], [Jaini et al. 2019]. Among those, affine coupling layers used in NICE [Dinh et al. 2015] and RealNVP [Dinh et al. 2017] offer moderate expressiveness but are fast, stable and easy to implement. Moreover, their inverse evaluation has the same cost as the forward evaluation. Finally, Lipschitz residual networks [Behrmann et al. 2019] are based on a residual network architecture, with a contractive residual function ensuring the reversibility of the network. The inverse mapping is then computed by fixed point iterations. This architecture is probably the least expressive.

Although only the final state matters for generative applications, incorporating deformation dynamics has been studied as a way to get more expressivity. Continuous Normalizing Flow (CNF) propose to learn the velocity field of an ODE and to compute its flow (i.e., samples trajectories) via an integration scheme ([Chen et al. 2019],[Grathwohl et al. 2018],[Finlay et al. 2020]). They can be seen as the continuous equivalent of Lipschitz residual networks. CNFs theoretically offer unlimited expressivity for representing continuous diffeomorphisms, provided the network is sufficiently wide, and the ODE integrator is accurate. Their main drawback lies in their computational cost, although some approaches have strived to reduce it [Onken et al. 2021].

Flow Matching approaches take a different approach: they directly learn the velocity field underlying the transformation of a source distribution into the target distribution. For supervision, the vector fields can come from random matching between source and target points, or more clever Optimal Transport based matching [Lipman et al. 2023; Tong et al. 2024]. While this eliminates the need for ODE solving during the training phase, it remains necessary during inference.

Another line of research aims to completely bypass the computational cost of ODE integration by training a model to directly predict the integral of the unknown vector field in the ODE. Neural Flows [Biloš et al. 2021] parameterize in time different invertible architectures to obtain continuous non-intersecting trajectories between source and target distributions. But as noticed by [Bizzi et al. 2024], such architecture lack the group law property of flow. Instead, NCF [Bizzi et al. 2024] use the conjugation of an affine vector field and an embedding operator to compute more expressive flows that still verify the complete group law. But the authors' implementation is limited to even dimensional spaces.

Finally, Optimal Transport approaches are an alternative for mapping distributions, which has been applied to shape interpolation ([Feydy et al. 2017; Lévy 2015; Solomon et al. 2015]). While this minimizes trajectory lengths, such approaches cannot handle deformations corresponding to rigid rotations and often introduce tearing in the deformed shape.

*Shape deformation.* We now turn to the deformation of shapes, focusing on implicit surfaces for brevity. Implicit shapes are defined as an iso-level set of a scalar function defined in the ambient space [Bloomenthal et al. 1997]. Among implicit representations, Signed Distance Fields (SDF) are probably the most popular, but this property is hard to maintain, especially within a deformation framework. Advecting an implicit shape by a velocity field is not addressed by an ODE but by a partial differential equation (PDE) known as the Level Set Equation (LSE) as evidenced by Osher and Sethian [Osher and Fedkiw 2003]. The LSE has been leveraged by Tao et al. [2016] for deforming shapes, by enforcing the vector field to be a Killing Vector Field near the surface. This forces the deformation to be as rigid as possible, a property which is central in shape deformation [Sorkine and Alexa 2007]. The LSE was also used by Tam and Schmidt [2011] to reconstruct the explicit velocity of an implicit surface evolution.

Recent approaches are targeting implicit neural representations to model shapes from a set of surface points. These approaches parameterize the implicit function by a small MLP [Xie et al. 2022]. They differ on whether they require oriented normals associated with the input points or whether they leverage a latent shape space [Mescheder et al. 2019; Park et al. 2019]. For Signed Distance Fields, the Eikonal property can be either encouraged via a loss [Gropp et al.

2020] or imposed via 1-Lipschitz networks [Coiffier and Béthune 2024]. The quality of the approximation also depends on the activation function as evidenced by SIREN [Sitzmann et al. 2020].

If these representations rely on a latent code, morphings can be obtained by interpolating within the latent space, assuming that the latent space has been structured to reflect plausible deformations [Atzmon et al. 2021]. LipMLP [Liu et al. 2022] also focuses on deforming an implicit surface with respect to a parameter, which can be either the latent code or time, by minimizing the Lipschitz constant of the evolution. Following the work of Mehta et al. [2022], NISE [Novello et al. 2023] proposes to learn a neural implicit function that evolves continuously over time and satisfies the level set equation (LSE) with a hand-crafted velocity field for interpolation. The implicit function coincides with a signed distance function at the initial and final times. NISE's main drawback lies in the need of a handcrafted vector field. To improve the deformation, it is interesting to enforce volume preservation, as a way to regularize intermediate shapes. To do so, Implicit Neural Shape Deformation (INSD) [Sang et al. 2025] penalizes the divergence of a learned vector field, whereas ADADIV [Buonomo et al. 2025] introduces the notion of adaptive divergence to ensure volume preservation, while preserving the Eikonal property as much as possible. INSD requires a dense set of correspondences between the source and the target surfaces, whereas ADADIV operates without landmarks. In these approaches, the trajectories of points are not explicit, even though it is possible to integrate the velocity vector field for a given point, as is done for the set of landmarks in INSD. Finally, Yang et al. [2021] propose to combine neural fields with a neural network modeling a direct mapping for solving shape editing problems. Their invertible network, based on iResNet [Behrmann et al. 2019], produces a direct diffeomorphism without intermediate shapes. On the contrary, we combine implicit neural fields with flows, instead of diffeomorphism, allowing to provide continuous intermediate shapes. On a quite different topic, flows have been used in the space of diffeomorphisms for matching images [Beg et al. 2005].

## 3 Flows and Shape deformation

### 3.1 Background on flows

For completeness, we start by recalling some important concepts on mathematical flows, We refer the reader to [Berger and Gostiaux 2012] for a more comprehensive introduction to these notions. First, let us state that flows are intrinsically linked with vector fields as they are (under mild hypotheses) the unique solution of an ODE linking the motion speed with a vector field. For clarity, we defer this discussion to Appendix A, and focus here on the induced trajectories.

Definition 1. *Let $d \in \mathbb{N}$, a mapping $\phi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is a flow iff:*

(1) $\forall x \in \mathbb{R}^d, \phi(x, 0) = x$,
(2) $\forall x \in \mathbb{R}^d, \forall t, s \in \mathbb{R}, \phi(x, t + s) = \phi(\phi(x, s), t)$.

Flows are invertible, in the sense that the current position of a point uniquely determines its location $t$ time units earlier. We write this inverse $\phi^{-1}(x, t)$ with $\phi^{-1}(\phi(x, t), t) = x$. Using the group law (property 2 of definition 1), it is easy to see that: $\phi^{-1}(x, t) = \phi(x, -t)$.

A direct consequence of this invertibility is that two trajectories cannot cross, and even simple deformations such as the combined translation and rotation pictured in Figure 2 cannot be represented as a flow in the ambient space.

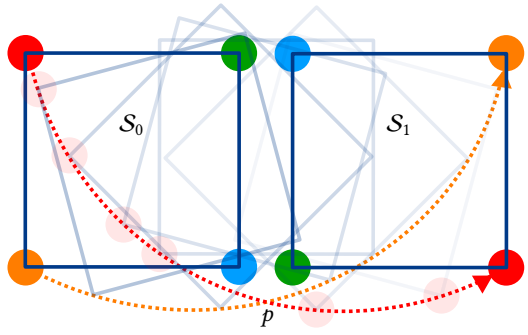


Fig. 2. A simple transformation case in $\mathbb{R}^2$ (simultaneous rotation and translation of $\mathcal{S}_0$ into $\mathcal{S}_1$) induces crossing trajectories. Both red and orange vertices pass through $p$, although at different times, yet their subsequent trajectories differ. Hence, it is impossible to represent the transformation by a flow in $\mathbb{R}^2$. The solution is to lift the problem in $\mathbb{R}^3$ adding the time as the third dimension.

To alleviate this issue, the problem can be lifted in higher dimension by defining the higher dimensional flow $\phi : \mathbb{R}^{d+1} \times \mathbb{R} \to \mathbb{R}^d \times \mathbb{R}$:

$$\phi((x_0, t_0), t) = (x(t + t_0), t + t_0). \tag{1}$$

With $x$ a smooth function from $\mathbb{R}$ to $\mathbb{R}^d$ such that $x(t_0) = x_0$ (it is a solution to a nonstationary ODE, see Appendix A). One can show easily that such a $\phi$ is indeed a flow in $\mathbb{R}^{d+1}$ by definition 1. Intuitively, this lifting amounts to encoding the point $x_0$ with the starting time $t_0$ and the duration of the advection by the flow $t$. This ensures that trajectories will never cross in $\mathbb{R}^{d+1}$. In the remainder of this paper, we will always use the lifted flow.

### 3.2 Shape deformation with flows

Given a closed shape $\mathcal{S}$ defined implicitly as the 0 level set of a Lipschitz function $g : \mathbb{R}^d \to \mathbb{R}$, the deformation $\mathcal{S}(t)$ of the shape by a flow $\phi$ can be represented by a spatio-temporal implicit function $f(x, t)$. Writing the level set preservation property, ensures that $\mathcal{S}(t)$ remains consistent throughout the deformation:

$$\forall x_0 \in \mathbb{R}^d, \forall t \in \mathbb{R}^+, \quad f(\phi((x_0, 0), t), t) = g(x_0). \tag{2}$$

Defining $y \in \mathbb{R}^d$ as $(y, 0) = \phi^{-1}((x, t), t) = \phi((x, t), -t)$, one can write $f(x, t) = f(\phi((y, 0), t), t) = g(y) = g(\phi^{-1}((x, t), t)) = g(\phi((x, t), -t))$, leading to the following definition:

Definition 2. *Let $\phi$ be a flow (defined in Equation 1), the spatio-temporal implicit function representing the deformation of $\mathcal{S}$ through $\phi$, and preserving level sets, is:*

$$f(x, t) = g(\phi^{-1}((x, t), t)) = g(\phi((x, t), -t)). \tag{3}$$

It is important to note that even if $g$ is a signed distance field, the spatio-temporal implicit function $f$ defined above does not necessarily retain the signed distance field property. However, if both $g$ and $\phi$ are spatially Lipschitz, $f$ is Lipschitz in space at any time. This is the case if the Jacobian of the flow is uniformly bounded.

## 4 Explicit Flow Computation

To compute the flow deforming a shape into another, many state-of-the-art methods first estimate the vector field associated to the deformation. Then they rely on numerical integration schemes (DOPRI, Runge Kutta [Dormand and Prince 1980]) for advecting the shape's points. As a result, the accuracy of the points trajectories depends on the numerical accuracy of the scheme. Achieving higher accuracy requires a high number of steps which increases the computational time significantly. Instead, we propose to compute the flow explicitly, by parameterizing it as a neural network with a specific architecture ensuring that it is a flow (i.e. it satisfies Definition 1).

### 4.1 Time-parameterized space diffeomorphisms

Our architecture builds upon a backbone that generates a diffeomorphism $\psi$ defined on $\mathbb{R}^d$, such as realNVP [Dinh et al. 2017], GLOW [Kingma and Dhariwal 2018] or VPNet [Zhu et al. 2022]. We modify the backbone by introducing a time parameterization, resulting in a continuous set of time-parameterized mappings $(\psi_t)_t$, such that, for any $t$, $\psi_t$ is guaranteed to be a smooth diffeomorphism of $\mathbb{R}^d$. We then build our flow by composing two particular diffeormorphisms of this set

$$\phi((x_0, t_0), t) = (\psi_{t+t_0}(\psi_{t_0}^{-1}(x_0)), t + t_0). \tag{4}$$

This construction bears some resemblance with Spanbauer and Sciarappa [2020]'s definition of a neural group action. Indeed, a flow on $\mathbb{R}^{d+1}$ is a group action of the additive group of real numbers $\mathbb{R}$ on $\mathbb{R}^{d+1}$. But their neural group action definition only targets finite groups (typically $\mathbb{Z}/k\mathbb{Z}$), building a specific diffeomorphism for each element of the finite group, which cannot be done for infinite uncountable groups such as $\mathbb{R}$.

Theorem 1. *The mapping $\phi$ defined by Equation 4 is a flow defined on the spatio-temporal space $\mathbb{R}^{d+1}$.*

Proof. One simply needs to check the properties of Definition 1.

(1) $\phi((x_0, t_0), 0) = (x_0, t_0)$ :

$$\phi((x_0, t_0), 0) = (\psi_{t_0}(\psi_{t_0}^{-1}(x_0)), t_0 + 0) = (x_0, t_0) \tag{5}$$

(2) $\phi(\phi((x_0, t_0), s), t) = \phi((x_0, t_0), s + t)$ :

$$\begin{aligned}\phi(\phi((x_0, t_0), s), t) &= \phi((\psi_{s+t_0}(\psi_{t_0}^{-1}(x_0)), s + t_0), t) \\ &= (\psi_{t+s+t_0}(\psi_{s+t_0}^{-1}(\psi_{s+t_0}(\psi_{t_0}^{-1}(x_0)))), t + s + t_0) \\ &= (\psi_{t+s+t_0}(\psi_{t_0}^{-1}(x_0)), t + s + t_0) \\ &= \phi((x_0, t_0), s + t)\end{aligned} \tag{6}$$

□

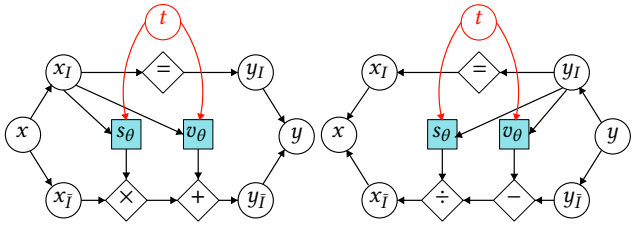


Fig. 3. Our coupling layers: direct and inverse transformations. Compared to the usual RealNVP layers the modifications are represented in red (time parameterization of the scaling and translation layers).

### 4.2 Time-parameterized coupling layers

Since our flow computation involves composing a diffeomorphism and the inverse of another diffeomorphism, efficiency is enhanced if the diffeomorphic network's design ensures that the inverse computation remains as low-cost as the forward pass. This is the case for networks based on coupling layers such as RealNVP [Dinh et al. 2017] or VPNet [MacDonald et al. 2021], but not for networks based on residual layers such as invertible ResNet [Behrmann et al. 2019], which require expensive fixed point iterations for inverse computation.

From now on we adopt the RealNVP architecture [Dinh et al. 2017] as our backbone. The idea is to transform only part of the coordinates using a transformation computed from the remaining coordinates to maintain invertibility. In $\mathbb{R}^d$, let us divide the coordinates in two blocks $x_I, x_{\bar{I}}$, the first block is fed into two small neural networks, called hereafter the *scaling and translation networks*. The translation network produces a translation vector $v_\theta(x_I)$. The scaling network produces a vector $\tilde{s}_\theta(x_I)$ yielding a scaling vector $s_\theta(x_I) = \exp \tilde{s}_\theta(x_I)$ with positive coefficients needed to ensure invertibility. $v_\theta(x_I)$ and $s_\theta(x_I)$ are then used transform $x_{\bar{I}}$. More formally:

$$\tilde{\psi}(x) = \begin{pmatrix} x_I \\ s_\theta(x_I) \odot x_{\bar{I}} + v_\theta(x_I). \end{pmatrix} \tag{7}$$

where $s_\theta(x_I) \odot x_{\bar{I}}$ is the Hadamard product between the two vectors. The next layer keeps $x_{\bar{I}}$ fixed, computes $s_\theta(x_{\bar{I}})$ and $v_\theta(x_{\bar{I}})$ and applies them to $x_I$. For $d > 2$, one must choose how to partition the coefficients into distinct subsets. For $d = 3$, there are 6 possible ways to split the coefficients into two non-empty subsets. Consequently, each layer will be divided into 6 sublayers, with each sublayer transforming only a subset of the coordinates. Since each sublayer can be trivially inverted, the whole layer and, in the end, the whole network are invertible. Furthermore, the inverse can be queried at the exact same computational cost as the forward computation. If the activation functions of the networks are smooth, the network is guaranteed to produce a diffeomorphism.

To build our time-parameterized set of space diffeomorphisms $(\tilde{\psi}_t)_t$, we introduce the time into the layers, as a parameter for the translation and scaling networks:

$$\tilde{\psi}_t(x) = \begin{pmatrix} x_I \\ s_\theta(x_I, t) \odot x_{\bar{I}} + v_\theta(x_I, t) \end{pmatrix}. \tag{8}$$

This is illustrated on Figure 3. This change does not alter the invertibility property, that is, for any $(x, t)$ one can compute $\tilde{\psi}_t^{-1}(x)$ as:

$$\tilde{\psi}_t^{-1}(x) = \begin{pmatrix} x_I \\ \frac{x_{\bar{I}} - v_\theta(x_I, t)}{s_\theta(x_I, t)} \end{pmatrix}. \tag{9}$$

where the division is done coefficient by coefficient. $\tilde{\psi}_t$ defines one coupling layer of our architecture. Several such layers are cascaded for better expressivity, which maintains the invertibility property, yielding a final time-parameterized mapping $\psi_t$. Flow $\phi$ follows by using the composition of Equation 4.

From now on, we refer to our method and its neural architecture as **EFIS** (Explicit Flow for Implicit Surfaces). We now prove a universality result ensuring that an EFIS architecture can indeed construct a flow between two diffeomorphic shapes. However, this result does not guarantee that any flow can be approximated, which is why it is referred to as a restricted universality theorem.

Theorem 2 (Restricted Universality). *Given two shapes $\mathcal{S}_0$ and $\mathcal{S}_1$ in $\mathbb{R}^d$ that are diffeomorphic, there exists a network following the EFIS architecture that provides a flow between $\mathcal{S}_0$ and $\mathcal{S}_1$.*

Proof. Given any two $\mathbb{R}^d$ shapes with the same topology, by the universality theorem in [Draxler et al. 2024] there exists a RealNVP architecture producing a diffeomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$. Let us consider a particular layer of this RealNVP instance, and let $s(x)$, $v(x)$ be the resulting subnetworks. A flow can be constructed by adding the following time parameterization of the layer $s(x, t) = exp(tlog(s(x)))$ and $v(x, t) = tv(x)$. The time-and-space parameterized MLPs of our architecture can approximate those expressions of $s(x, t)$ and $v(x, t)$ at any arbitrary precision. This ensures the existence of an EFIS flow mapping $\mathcal{S}_0$ to $\mathcal{S}_1$ and matching the diffeomorphism at $t = 1$. In practice this hard-coded approximation lacks expressivity, and we rather let the network optimize for the parameterization. □

For our experiments, unless explicitly specified otherwise, we use 12 sequential coupling layers (Figure 3), with $I$ successively equal to $x; y; z; xy; yz; xz$ (with the sequence repeated two times). Each translation and scaling networks are small MLPs with two fully-connected 128 neurons wide hidden layers (See section 9 for an ablation of the architecture), and Sine activation functions [Sitzmann et al. 2020].

## 5 Optimization

Our construction provides a way to parameterize a flow with a neural network, hence allowing to look for solutions of typical geometry processing problems as flows. In this section, we define the losses that drive our optimization. Losses are expressed in the continuous setting, but are in practice computed by Monte Carlo integration.

Let $g_0$ (resp. $g_1$) be a signed implicit representation of $\mathcal{S}_0$ (resp. $\mathcal{S}_1$). Without loss of generality, we assume that the surfaces of these 3D shapes correspond to the 0 level set of $g_0$ and $g_1$.

*Dirichlet Loss.* The first loss ensures that the points of the source shape are mapped onto the target shape and vice versa. The proximity of the transported points to the shape surface is measured by

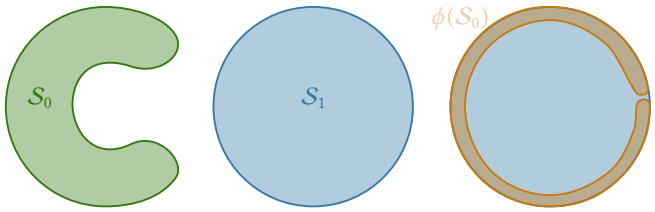


Fig. 4. Issue arising when training the flow only on surface points to deform $\mathcal{S}_0$ into $\mathcal{S}_1$. Both the blue and orange shapes are critical points of $\mathcal{L}_D$

$g_0$ and $g_1$, and we have the following relationships:

$$\begin{aligned} g_1(\phi((x_0, 0), 1)) &= g_0(x_0) = 0, \quad x_0 \in \partial\mathcal{S}_0 \\ g_0(\phi((y_1, 1), -1)) &= g_1(y_1) = 0, \quad y_1 \in \partial\mathcal{S}_1. \end{aligned} \tag{10}$$

This translates the following Dirichlet loss:

$$\mathcal{L}_D = \int_{x \in \partial\mathcal{S}_0} \|g_1(\phi((x, 0), 1))\|_2^2 dx + \int_{y \in \partial\mathcal{S}_1} \|g_0(\phi((y, 1), -1))\|_2^2 dy. \tag{11}$$

*Neumann Loss.* Normals of the deformed shape should match the target shape normals. The target normals can either be the explicit normals of the input shapes or computed as the gradient of the implicit representations. The deformed normals can be computed using the gradient of the deformed implicit function, leading to:

$$\begin{aligned} \mathcal{L}_N = &\int_{x \in \partial\mathcal{S}_0} 1 - \text{CosSim}\left(n_x, \nabla(g_1(\phi((x, 0), 1)))\right) dx \\ &+ \int_{y \in \partial\mathcal{S}_1} 1 - \text{CosSim}\left(n_y, \nabla(g_0(\phi((y, 1), -1)))\right) dy. \end{aligned} \tag{12}$$

where $n_x$ denotes the normal of a surface point $x$, $\nabla$ is the gradient relative to the space coordinates (and not the time), and CosSim denotes the cosine similarity measuring the similarity of two vectors. This loss drives the model to learn smoother surface interpolations.

*Interior/exterior Loss.* Constraining the flow only on surface points is not enough, as shown on Figure 4. We therefore add a volumetric constraint to guide the flow by penalizing sign changes between $g_0$ at a point and $g_1$ at the advected point. This can be cast as an inside-outside classification task for which a binary cross-entropy loss (BCE) is particularly well-suited:

$$\begin{aligned} \mathcal{L}_S = &\int_{x \in \mathbb{R}^d} \text{BCE}(\text{sign}(g_1(\phi((x, 0), 1))), \text{sign}(g_0(x))) dx \\ &+ \int_{x \in \mathbb{R}^d} \text{BCE}(\text{sign}(g_0(\phi((x, 1), -1))), \text{sign}(g_1(x))) dx. \end{aligned} \tag{13}$$

Where the sign function is replaced by the smooth positive function $\sigma(x) = sigmoid(5x)$ to avoid numerical issues.

The interior/exterior loss could be extended over time by forcing the signs of $g_1(\phi((x, t), 1 - t))$ and $g_0(\phi((x, t), -t))$ to coincide for any $t$. However, in practice, this overconstrains the system without improving the results.

*Landmark loss.* If some point correspondences between the shapes are available, a loss based on the distance between the advected points and their correspondences can be leveraged. Let $C = \{(x_0, x_1) | \ x_0 \in \partial\mathcal{S}_0, x_1 \in \partial\mathcal{S}_1\}$, represent the set of known corresponding point pairs.

$$\mathcal{L}_{lm} = \sum_{(x_0,x_1)\in C} \|\phi((x_0, 0), 1) - x_1\|_2^2 + \|\phi((x_1, 1), -1) - x_0\|_2^2. \quad (14)$$

## 5.1 Regularizations

Flows optimized using the losses above do not always provide physically interesting intermediate shapes, since they rely only on times 0 and 1. To regularize the flow, we propose 3 losses based on intermediate times. These losses are defined in the ambient space and do not depend on the position of the shape, which allows us to express them directly in terms of the diffeomorphism $\psi_t$ and not the flow itself. This saves a few neural network evaluations, that we explain below for each of the regularizations individually.

*Isometry regularization.* The first regularization term encourages the flow to be distance-preserving.

$$\mathcal{L}_{iso} = \int_{t\in[0,1]} \int_{x\in\mathbb{R}^d} \|J_{\psi_t}^T(x) J_{\psi_t}(x) - I\|_2^2 dx dt. \quad (15)$$

Having an as isometric as possible flow reduces spurious motions and helps the intermediate shapes retain the features of the interpolated shapes. Since the isometry property is preserved throughout the composition of isometric functions, $\phi$ is an isometry if $\psi_t$ is an isometry.

Since enforcing distance preservation across the entire volume would result only in rigid transformations, this loss should theoretically be applied specifically to the surface. To avoid using surface integrals, we opt for volume integrals but increase the sampling density of points near the surface, by applying the flow to points sampled on the initial surface.

*Volume preservation.* Volume variation are locally given by the Jacobian of the flow. This leads to the following criterion penalizing local volume change:

$$\mathcal{L}_{vol} = \int_{t\in[0,1]} \int_{x\in\mathbb{R}^d} \|\log\det(J_{\psi_t})(x)\|_2^2 dx dt. \quad (16)$$

Since $\det(J_\phi((x_0, t_0), t)) = \det(J_{\psi_{t+t_0}}(\psi_{t_0}^{-1}(x_0)))\det(J_{\psi_{t_0}^{-1}}(x_0))$, the volume variation of $\phi$ is controlled by the volume variation of $\psi_t$. Setting $\forall (x, t) \in \mathbb{R}^4$: $\det(J_{\psi_t}(x)) = 1$ is a sufficient condition for $\phi$ to be volume preserving.

RealNVP[Dinh et al. 2017] models are composed of coupling layers. A coupling layer preserves a part of the input to ensure invertibility ($\psi_t(x)_I = x_I$) and the other part is modified by an element-wise transformation. More precisely, the Jacobian of a coupling layer is a triangular matrix whose log-determinant is $\sum_{j\in I} \log(s_\theta(x_I, t)_j)$. Since $\log\det(AB) = \log\det(A) + \log\det(B)$ and $J_{f\circ g} = J_f(g)J_g$, the log-determinant of the whole model can be computed by summing the log-determinant of each coupling layer. Volume preservation is theoretically redundant with the previous isometry regularization, but we observed in practice faster convergence when combining the two.

Interestingly, in practice, it is more effective to use a non-volume-preserving architecture such as RealNVP [Dinh et al. 2017] combined with a loss function that promotes volume preservation, rather than relying on a volume-preserving architecture (such as VPNet [Dinh et al. 2017]). This is consistent with the observation of [Draxler et al. 2024] that volume preserving coupling architectures are not universal approximators.

*Time regularization.* To promote smooth motions in time, we introduce a penalty on the rate of shape deformation

$$\mathcal{L}_T = \int_{t\in[0,1]} \int_{x\in\mathbb{R}^d} \|\partial_t\psi_t(x)\|_2^2 dx dt. \quad (17)$$

The local velocity of the flow is given by $\partial_t\phi((x_0, t_0), t) = \partial_t[\psi_{t+t_0} \circ \psi_{t_0}^{-1}(x_0)] = \partial_t[\psi_{t+t_0}] \circ \psi_{t_0}^{-1}(x_0)$. When this quantity is integrated over space, the change of variable formula allows us to replace $\psi_{t_0}^{-1}(x_0)$ by the determinant of its Jacobian, which is close to 1 because of the volume preservation regularization.

An advantage of our time-parameterized coupling layers is that derivatives with respect to the time, for a fixed position $x$, are obtained efficiently through a forward differentiation pass. The association of the Dirichlet loss and both the time and volume regularizations is close to the formulation of the incompressible Benamou-Brenier transport [Benamou and Brenier 2000], a link worth exploring but beyond the scope of this paper.

## 5.2 Forward and backward blending

For interpolating between two shapes given implicitly by $g_0$ and $g_1$, either the deformation of $g_0$ with $\phi$ or the deformation of $g_1$ with $\phi^{-1}$ may be used, as they are theoretically equivalent when the Dirichlet constraints are satisfied. More precisely, let us define $f_0$ and $f_1$ as:

$$f_0(x, t) = g_0(\phi((x, t), -t)) \quad (18)$$

$$f_1(x, t) = g_1(\phi((x, t), 1 - t)). \quad (19)$$

It is easy to show that $\forall t \in [0, 1], f_0(x, t) = f_1(x, t)$ (see appendix B). In practice $f_0$ and $f_1$ may yield different results due to the training error. This small error can be mitigated to improve the intermediate shapes, by averaging the implicit functions: $\forall x \in \mathbb{R}^d, t \in [0, 1], f(x, t) = (1-t)f_0(x, t)+tf_1(x, t)$. The spatio-temporal implicit function is then guaranteed to match the input implicit functions at times 0 and 1. Notice that while this blending allows for more visually pleasant results, it can produce some minor topological noise on very thin structures. However, no flow associated with this blended deformation is explicitly available, and it might not even exist, so this trick should only be used to improve the rendered intermediate shapes.

# 6 Application to Shape Interpolation

*Input Data.* We begin by applying our method to interpolate between shapes that exhibit varying levels of detail and features. Our experiments use pre-aligned shape meshes from the FAUST dataset [Bogo et al. 2014], with feet positions matched for alignment (see Appendix D for the naming convention), as well as shapes from the THINGI10K dataset [Zhou and Jacobson 2016]. The input implicit surfaces are neural SDFs represented using SIREN networks

Table 1. Computation times for the experiments of Figures 5 and 7.

| Method | EFIS (Ours) | Nise | lipmlp | ADADIV |
|---|---|---|---|---|
| falcon-witch | 8139*s* | 3329*s* | 2576*s* | 9279*s* |
| armadillo-blob | 6045*s* | 2304*s* | 6757*s* | 5294*s* |
| bunny-fandisk | 23238*s* | 10570*s* | 6160*s* | 24425*s* |
| owls-fandisk | 18336*s* | 8256*s* | 5091*s* | 19055*s* |
| submarine-horse | 15287*s* | 8163*s* | 4518*s* | 17746*s* |
| Faust1-5 | 1533*s* | 1335*s* | 3113*s* | 3329*s* |

(6 layers, 128 neurons per layer). While we adopt neural SDFs in this paper, *any other* Lipschitz implicit representation could be used, as long as it can be queried continuously in space and differentiated on the surface to compute the Neumann loss (see section 9).

Our experiments were run on two different types of NVIDIA GPU (RTX 4070 and TESLA V100). Computational times are measured on the TESLA V100.

### 6.1 Shape interpolation without landmarks

Our first application is interpolation between any two arbitrary shapes, without relying on semantic correspondences. Figure 5 shows examples of such morphings: the deformations are smooth and do not produce topological artifacts. Additional results and comparisons can be found in the accompanying video. For this application, we used the following loss weights: $\lambda_D = 1e4$ (Dirichlet), $\lambda_N = 1e2$ (Neumann), $\lambda_S = 1e2$ (Interior/Exterior loss), $\lambda_{iso} = 1e1$ (isometry regularization), $\lambda_{vol} = 1$ (volume preservation) and $\lambda_T = 1e1$ (time regularization). Sampling for the Monte Carlo approximation of loss integrals is done uniformly in space and time in a bounding box of size $[-1, 1]^3 \times [0, 1]$. Surface points are sampled uniformly on the input shapes, provided as meshes in our experiments. To give more weight to the near-surface region when computing the regularization losses, we trace the trajectories of the input surface points, generating samples of the intermediate shapes.

Figure 6 shows a comparison with morphings obtained using recent baselines. For this comparison, we consider the vanilla versions of ADADIV, NISE, and LipMLP. For the NeuralFlow baseline, we adapt their architecture to neural implicit shape deformation and optimize it using our supervision and regularization losses. Both our EFIS method and NeuralFlow clearly outperform the other approaches. However, the trajectory produced by NeuralFlow is noticeably less smooth over time: the shape tends to remain close to the source and then undergoes a large deformation as $t$ approaches 1. This behavior is due to a characteristic of their time parameterization, which we describe in more detail in Section E. For arbitrary shapes the resulting morphings are on par with the most recent methods [Buonomo et al. 2025; Sang et al. 2025], whereas faster baselines [Liu et al. 2022; Novello et al. 2023] yield considerably less natural results. Table 1 reports the computation times for all methods. Figure 7 also compares our approach on a human shape (still without relying on semantic correspondences).

Our approach requires that the source and target shapes share the same topology but is not limited to genus 0, Figure 8 exemplifies this on higher genus shapes. However, the supervision is more difficult for higher genus shapes when topological features have too different sizes. In that case the training can fail in local minima similar to the one displayed in Figure 4. This problem could be mitigated by adding a topological feature detection and matching step, something that is beyond the scope of this work.

### 6.2 Shape interpolation with landmarks

Figures 9 and 10 illustrate the effect of incorporating landmarks when morphing shapes with semantic correspondences, using the landmark loss defined in Equation 14. In the simple case of a rotating human shape without landmarks, the transformation results in arms growing from the back of the figure, producing an uncanny effect. Landmarks help eliminate this issue. Figure 10 demonstrates that increasing the number of landmarks leads to more natural deformations, preventing unnatural stretching or excessive inflation of human limbs. For these experiments, we added the landmark loss with weight $\lambda_{lm} = 1e4$.

Figure 12 compares our method with approaches using semantic correspondences. On a simple case, all methods perform well, although INSD requires 50% of landmarks (point correspondences) to reach a result comparable to ours, which only uses 1%. Note that the method of Eisenberger et al. [2019] is actually solving a more complicated problem (estimating both correspondences and deformation) which explains the somewhat lower results.

For isometric deformations of surfaces, our approach enables the direct advection of the surface mesh over time, eliminating the need to remesh intermediate shapes. While this method may produce low-quality meshes for arbitrary shapes, due to triangle distortion during deformation, it is highly effective for the FAUST dataset. In this case, the input and target shapes are described by meshes with the same intrinsic geometry, hence EFIS can be used to advect the surface mesh directly, resulting in superior detail preservation. The results presented in figure 11 were obtained by using 1% of the input meshes vertices as landmarks for our flow optimization.

## 7 Morphing pairwise-to-any

We introduce a fast morphing approach to generate transformations between any two shapes of the same topological genus, provided that pairwise flows between each individual shape and a suitable canonical shape have been computed. Consequently, by computing $N$ morphings between each of the input shapes and a canonical shape, we can obtain *all* $N(N+1)/2$ morphings between any input shape pairs (Figure 13).

In this setting, we consider two shapes $\mathcal{S}_1$ and $\mathcal{S}_2$ and one *canonical* shape $\mathcal{S}_0$. This canonical shape should have the same topology as $\mathcal{S}_1$ and $\mathcal{S}_2$ and can be chosen arbitrarily, although this choice strongly impacts the results (see Figure 15). Let $\phi_1$ refer to the flow from $\mathcal{S}_0$ to $\mathcal{S}_1$ and $\phi_2$ to the flow from $\mathcal{S}_0$ to $\mathcal{S}_2$. We propose to compose these two flows using either:

- $\chi_{1\to2}(x,t) = \phi_1(\phi_2((x,1),-t),t)$
- $\chi_{2\to1}(x,t) = \phi_2(\phi_1((x,1),-t),t)$.

Note however that a composition of non-commutative flows is no longer a flow. But since the composition remains invertible for all time, the intermediate shapes are still well defined and the trajectories remain non-intersecting. The inverse of $\chi_{1\to2}$ does not coincide

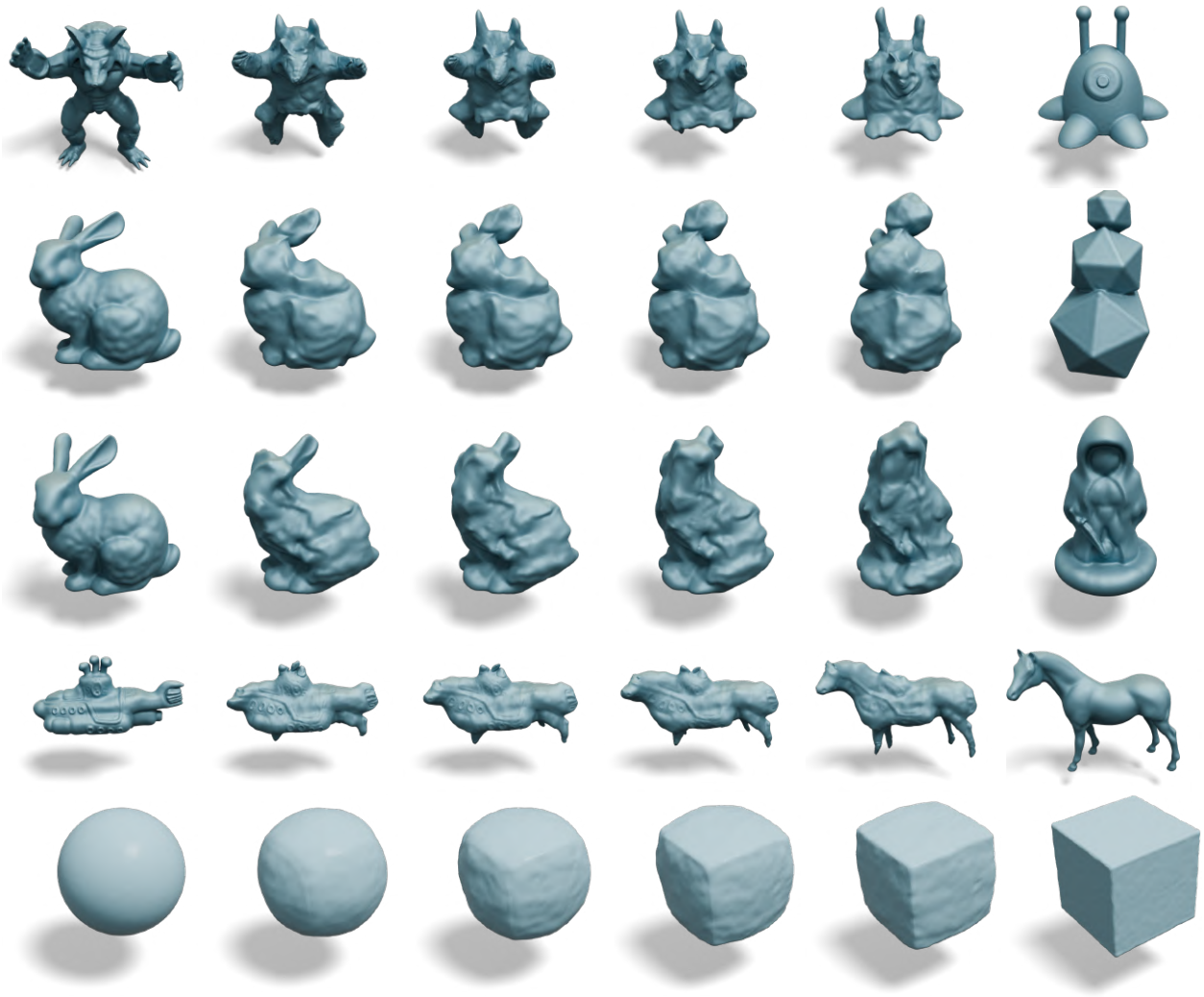

Fig. 5. Interpolation between arbitrary shapes without any semantic correspondences ($t = 0, 0.2, 0.4, 0.6, 0.8, 1$).

with $\chi_{2\to1}$, but in practice the results are very similar so we use either formula arbitrarily.

Figure 14 illustrates the results for various morphings between arbitrary shapes of genus 0 through a sphere. Although these morphings are less expressive than the direct ones, they are still reasonable solutions to the morphing problem. Figure 15 compares the morphings obtained through different canonical shapes: a sphere, a sphere with added noise and the Stanford Bunny. Although the canonical shape is not explicitly visible in the deformation, certain features, such as oscillations caused by the noise in the noisy sphere, can be observed. Figure 16 shows the pairwise-to-any morphings for various Faust shapes, with 1% landmarks, using a T-pose human shape as canonical shape. While the results are not on par with direct morphings obtained in the previous sections for human shape deformation, it can still capture a valid motion from source to target without passing through the canonical pose. Once again it is possible to advect the mesh of the canonical shape to get directly the deformed mesh (Figure 17).

For shapes sharing similar features in different positions (e.g., human poses), it is important that the canonical shape exhibits the same features. Otherwise, the resulting transformation may fail to preserve these features in the interpolated shapes, as shown in Fig 18.

## 8 Edition

In the editing framework, only the starting shape $\mathcal{S}$ is available. The target shape $\tilde{\mathcal{S}}$ is described solely by a set of landmarks $(x_0, x_1)$ such that $x_0 \in \partial\mathcal{S}$, $x_1 \in \partial\tilde{\mathcal{S}}$ and $\phi((x_0, 0), 1) = x_1$. The full deformation is defined by an additional assumption on how the residual shape follows the motion of the landmarks. A common assumption for a plausible deformation is the thin-shell energy. Yang et al. [2021] have proposed an implementation of this energy in the neural field

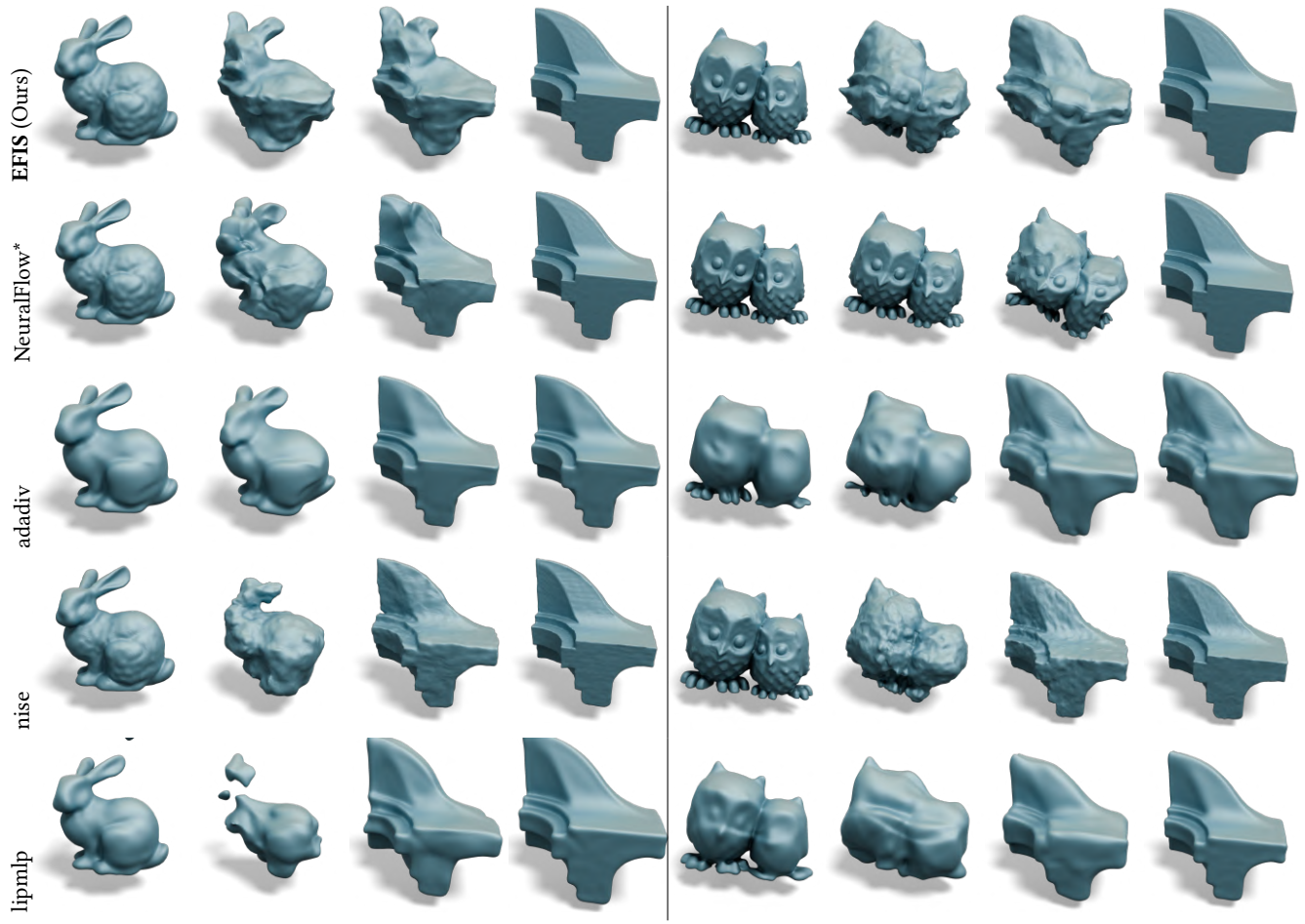


Fig. 6. Comparison with existing approaches. NeuralFlow* corresponds to the adaptation of the Neural Flows architecture to Lipschitz implicit surfaces and supervision with our losses. ADADIV, Nise, LipMLP are produced using the authors available implementation ($t = 0, 0.33, 0.661$).

framework (NFGP) for shape editing. Their implementation is based on a bending loss that penalizes curvature variation and a stretching loss that penalizes tangential stretching. We adapt their losses to our case (see Appendix C for more details). The vertices prescribed positions are incorporated into the landmark loss (Equation 14).

Figure 19 compares our results with NFGP and ARAP [Sorkine and Alexa 2007]. Our method exhibits nice detail preservation at a computational cost far lower than NFGP. ARAP is by far the fastest method (16 s) but leads to a more local deformation clearly depending on the high resolution of the mesh, while we benefit from the flow prior. Figure 20 shows the results obtained throughout the optimization steps of the flow. Our method provides a good result much faster than NFGP (24 min vs 300 min, framed in Figure 20), and the behavior is far less oscillatory.

Compared to ARAP and NFGP, using our flow for editing a mesh allows us to recover a smooth transition between the input and the edited shape (Figure 21). To do so, we include the bending and stretching losses for intermediate times (and not only for $t = 1$) with weights $\lambda_{bend} = 0.1$, $\lambda_{strech} = 10$ and the time regularization (Equation 17) with weight $\lambda_T = 10$. See also the accompanying video.

## 9 Analysis and Ablations

We compare our flow architecture with several other architectures in 2D on Figure 22. NODE, Neural Flows and our method exhibit better performances, but Neural Flows produces a morphing with far more inertia at the initial time, we conjecture that this is due to the $u(0) = 0$ condition in their time-parameterization (see appendix E). Neural Conjugate Flows delivers the worst result. In fact, this method was originally demonstrated for image morphing, and the author-provided implementation only works in even-dimensional spaces, hence we only compare it on 2d shape morphing, by replacing the images by Lipschitz implicit curves.

We evaluate our flow numerically in terms of invertibility error and group law error defined as:

$$E_{inv} = \int_{t\in[0,1]} \int_{x\in\mathbb{R}^d} \|\phi^{-1}(\phi(x,t),t) - x\|_2^2 dt dx. \tag{20}$$

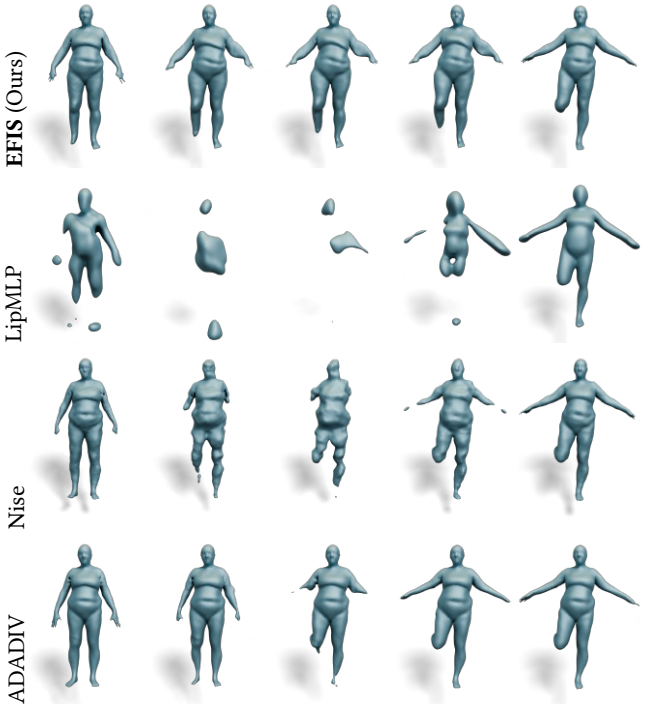


Fig. 7. Comparison with other methods on the Faust dataset, without using landmarks ($t = 0, 0.25, 0.5, 0.75, 1$)

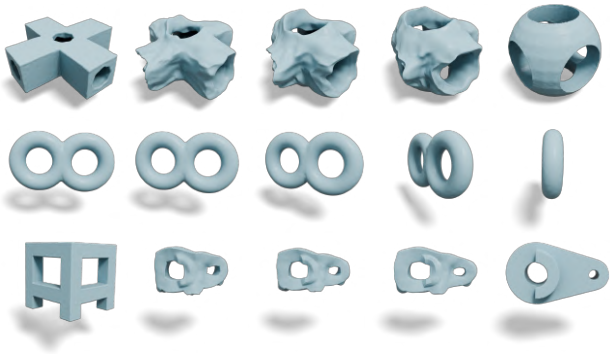

Fig. 8. Interpolations between shapes of higher genus ($t = 0, 0.25, 0.5, 0.75, 1$).

$$E_{group} = \int_{s,t,t+s\in[0,1]} \int_{x\in\mathbb{R}^d} \|\phi(\phi(x,t),s) - \phi(x,t+s)\|_2^2 dxdtds. \tag{21}$$

These errors are evaluated through Monte Carlo sampling in time and space and reported in Table 2. Our flow with RealNVP as a backbone exhibits the lower group law error and second lowest inversion error. In addition, it is also one of the fastest for forward computation and auto-differentiation. For all these methods we set a similar number of parameters reported in the table. By removing the need for an ODE solver, our inversion error is solely due to the numerical precision. As expected, Adjoint NODE [Chen et al. 2019] which is specifically designed to be memory efficient outperforms every other method by far in terms of memory consumption. However, it is also by far the worst in terms of errors and computation time, making it unusable. NODE being prohibitive in terms of memory cost, we only demonstrate it in 2D (Figure 22).

Figure 23 compares different architecture variations (number of layers) and choices for the coordinates split in the sublayers showing that our chosen architecture outperforms (blue curve) ablated

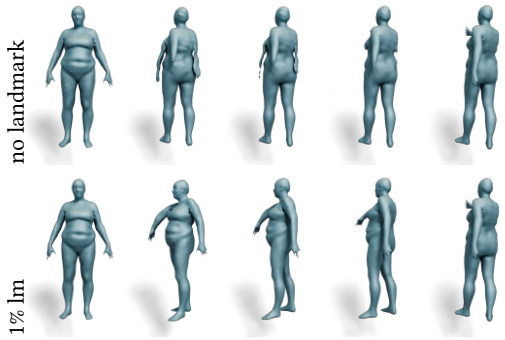


Fig. 9. Interpolation between shapes from the FAUST dataset without landmarks vs with using landmarks ($t = 0, 0.25, 0.5, 0.75, 1$).

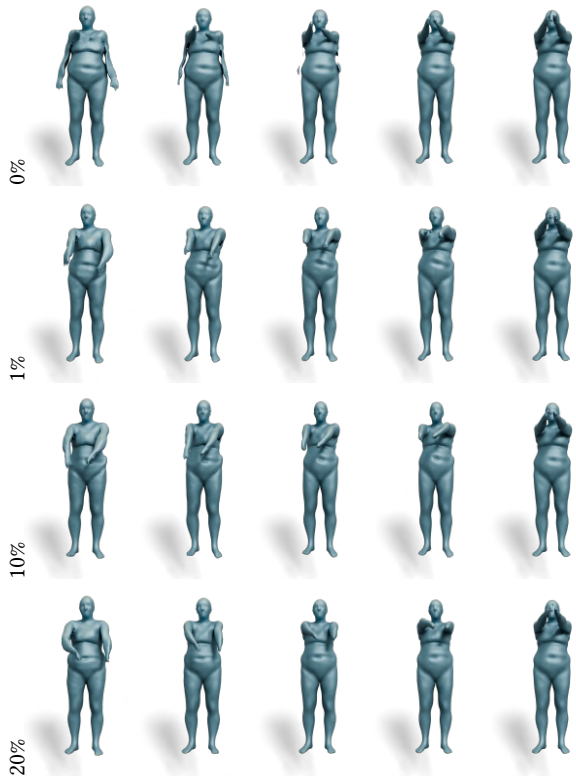


Fig. 10. Interpolation between several FAUST shapes with various amounts of landmarks, given in percentage of vertex numbers in the original mesh ($t = 0, 0.25, 0.5, 0.75, 1$).

Table 2. Evaluation of the error, computation times and memory cost (average and standard deviation) of our flow computation with RealNVP as a backbone compared to our flow with iResNet, Neural Flow (NF) [Biloš et al. 2021], Neural Conjugate Flow (NCF) [Bizzi et al. 2024], NODE and Adjoint-NODE [Chen et al. 2019] with models of similar size (best in bold, second best underlined). Our method exhibits the best trade-off between error, computation time and memory cost. Evaluations are averaged over 5000 inputs (networks are not trained).

| | **EFIS** (Ours) (RealNVP) | EFIS (iResNet) | NF | NCF | NODE | Adjoint -NODE |
|---|---|---|---|---|---|---|
| Nb param. | 421302 | 407862 | 421302 | 417948 | 413699 | 413699 |
| $E_{inv}$ (1e-8) | <u>11.46</u> \| 74.32 | 3348 \| 3152 | **4.295** \| 6.671 | 25.68 \| 131.2 | 22.08 \| 30.00 | 22.08 \| 30.00 |
| $E_{group}$ (1e-8) | **10.22** \| 28.51 | 418.6 \| 477.5 | 267.0 \| 351.2 | 28.31 \| 96.08 | 17.33 \| 23.07 | <u>17.33</u> \| 23.07 |
| Forward (ms) | <u>19.65</u> \| 6.41 | 130.1 \|31.43 | 36.15 \| 9.941 | **18.71** \| 5.907 | 26.58 \| 3.090 | 27.82 \| 4.471 |
| Auto-diff (ms) | <u>82.24</u> \| 6.722 | 167.7 \| 39.47 | 104.8 \| 12.85 | **80.03** \| 4.955 | 123.4\| 9.319 | 2022 \| 172.8 |
| Memory cost (MB) | 492.7 | 3088 | 487.4 | <u>484.3</u> | 2585 | **0.1538** |

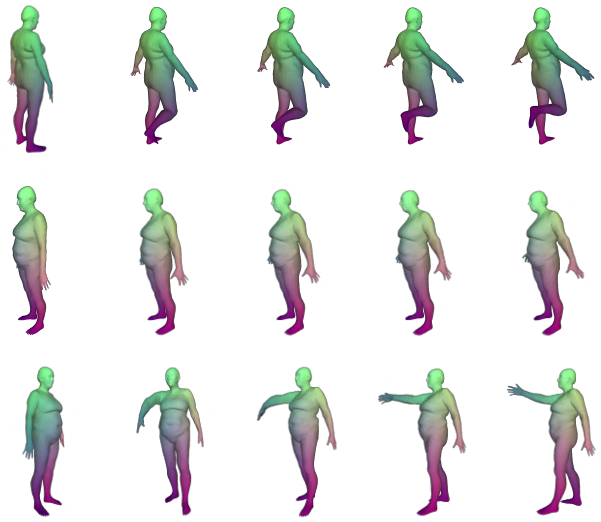

Fig. 11. Advection of the input mesh using EFIS. From top to bottom: Faust 1 to 3, Faust 1 to 5, Faust 1 to 7 ($t = 0, 0.25, 0.5, 0.75, 1$).

models. We further analyze the impact of our losses on Figure 24 for the blob-column experiment. One can see that without Dirichlet and Neumann losses the target shapes are not so well captured. Furthermore, without the sign loss, empty shells might appear. The temporal regularization is crucial to avoid surface oscillations, and isometry ensures smooth intermediate surfaces. Notice that we do not ablate the volume regularization, as in our experiments it does not alter the result much but without it, some optimizations do not converge. We do not use the blending trick of subsection 5.2 when displaying these results, to show the flow direct output.

So far, we have relied on neural implicit representations approximating Signed Distance Fields for the source and target shape. However, our approach is compatible with a wide range of Lipschitz implicit representations, provided they allow continuous spatial querying and surface differentiation. Figure 25 shows an interpolation between 2 shapes represented by Gaussian Radial Basis Functions with infinite support.

We evaluated the approximation quality of our method on the FAUST dataset since it provides ground truth correspondences for

Table 3. Average mean square error and deviation between target landmarks and advected landmarks

| | Isometry | Isometry + Landmarks | Bend.-Stretch. | Bend.+Stretch. + Landmarks | INSD |
|---|---|---|---|---|---|
| faust 1-2 | 0.89\|1.43 | 0.27\|0.31 | 1.82\|4.19 | 0.22\|0.25 | 1.40\|3.58 |
| faust 1-3 | 1.84\|5.25 | 0.34\|0.53 | 0.50\|0.77 | 0.21\|0.26 | 18.44\|42.37 |
| faust 1-4 | 24.19\|52.35 | 0.86\|1.77 | 63.33\|126.14 | 1.00\|2.40 | 4.99\|12.34 |
| faust 1-5 | 0.44\|1.15 | 0.19\|0.22 | 0.19\|0.20 | 0.18\|0.19 | 0.78\|1.87 |
| faust 1-6 | 51.65\|112.03 | 1.35\|2.49 | 52.62\|113.30 | 1.38\|2.06 | 3.53\|23.36 |
| faust 1-7 | 79.68\|87.53 | 0.33\|0.53 | 82.86\|98.23 | 0.20\|0.28 | 3.98\|12.90 |
| faust 1-8 | 11.38\|28.74 | 0.42\|1.13 | 59.70\|126.50 | 0.34\|1.00 | 23.95\|87.83 |
| faust 1-9 | 58.48\|116.00 | 0.44\|0.97 | 57.28\|115.54 | 1.27\|2.76 | 80.33\|146.4 |

Table 4. Average deviation to the groundtruth over time: Chamfer distance ($\times 1e3$) | Hausdorff Distance ($\times 1e2$) on the Bear dataset.

| | **EFIS** | ARAP | Nise | ADADIV | LipMLP | INSD |
|---|---|---|---|---|---|---|
| Chamfer | **4.34** | 7.29 | 37 | 63 | 73 | 51 |
| Hausdorff | **6.03** | 13.72 | 23 | 33 | 35 | 32 |

the complete meshes. We trained the model with 1% landmarks and measured the landmark loss on the whole set of correspondences after training. We compare our results with the one of INSD with the same number of landmarks (Table3). Due to the intrinsic regularity of our architecture, even landmarks not seen during training are well matched. On the other hand, INSD tends to overfit on the integration scheme and generalizes poorly to unseen landmarks.

To quantify the deviation of the interpolated shapes from the ground truth provided in an animated sequence, we use the running bear animation from the DeformingThings4D dataset [Li et al. 2021]. It consists in a sequence of meshes whose vertices' positions are tracked over time. Hence, the correspondences are fully provided. Nise and LipMLP were trained by using correspondences implicitly, *i.e.* using only the fact that corresponding points should remain on the 0 level set. ADADIV and INSD used correspondences both implicitly and explicitly, while ARAP and EFIS used only explicit correspondences. This corresponds to the best performing setting for each of the methods. The correspondences consist of a subset of 100 surface vertices from the deforming mesh, sampled every 5 time steps in the sequence. The errors were computed on the rest of the samples (for which the groundtruth deformation is known). The results are displayed in Table 4.

Finally, our shape interpolation task is highly dependent on the initial pose of the shapes. We compare on Figure 26 the results

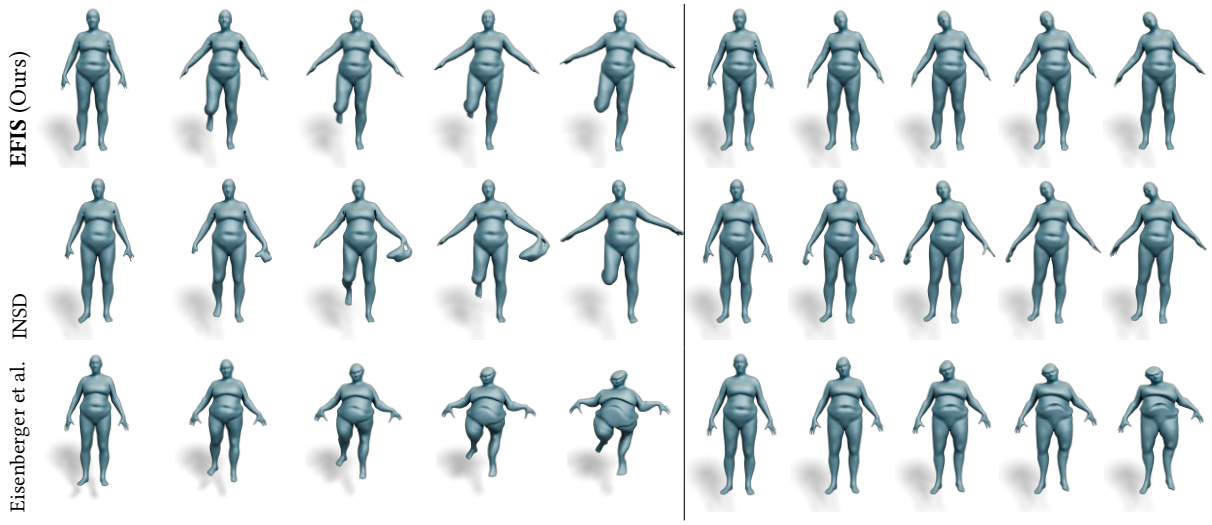


Fig. 12. Comparison on the Faust dataset with other methods leveraging shape semantics in the matching: INSD [Sang et al. 2025] (50% landmarks) and Eisenberger et al. [2019] who compute both deformation and correspondences jointly. Our result uses only 1% landmarks (50 landmarks) ($t = 0, 0.25, 0.5, 0.75, 1$).

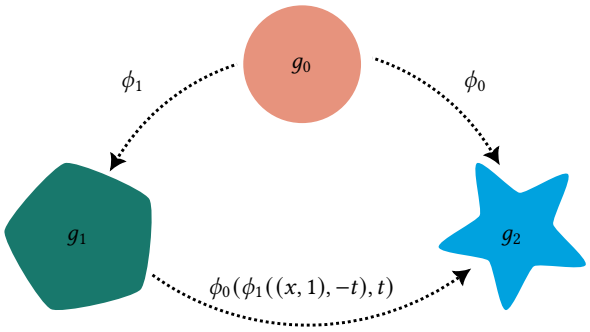


Fig. 13. Pairwise to any composition

obtained with the same initial shape, but in different poses. Our method tends to transform the features locally, but sparse landmarks could be used to guide the feature deformations.

## 10 Limitations

While our approach offers a mathematically grounded flow with applications in geometry processing, there remains several limitations. First, the expressivity of the flow relative to the number of layers is difficult to predict. Additionally, computational costs remain significant, though they are more manageable for smaller shapes, such as those in the FAUST dataset. A fundamental constraint inherent to the flow formalism is its inability to handle topological changes, it can only morph shapes with identical topologies and cannot generate new handles or holes, unlike purely implicit deformation methods. Finally, although it may be appealing to use this flow as a substitute for flow matching in the generative context, avoiding thereby Euler steps in generative processes, it is important to note that the true advantage of our flow lies in its intermediate steps, which are unnecessary in typical generative contexts.

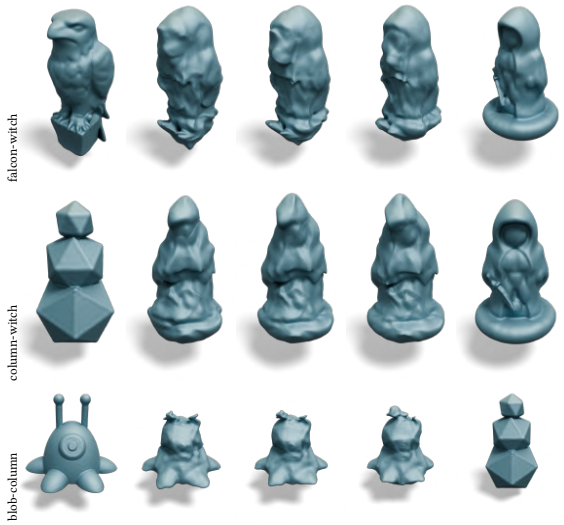


Fig. 14. Pairwise-to-any interpolation between arbitrary shapes without any landmarks with a sphere as the canonical shape ($t = 0, 0.25, 0.5, 0.75, 1$).

## 11 Conclusion

We presented a method for computing explicit mathematical flows, offering theoretical guarantees for deformation, with applications to shape deformation and editing. A promising direction for future research would be to explore improved combinations of flows for pairwise-to-any mappings, focusing on enhancing their regularity properties. Additionally, our flow framework holds potential for a broader range of geometry processing tasks, such as shape animation, trajectory computation, or solving general ODEs in cases where the system's temporal behavior is of interest.

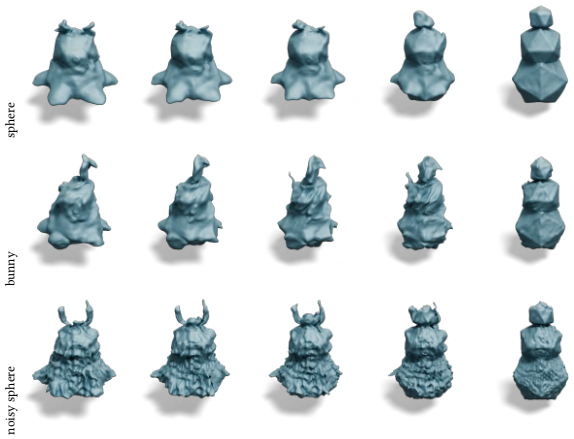


Fig. 15. Pairwise-to-any interpolation via various canonical shapes (a perfect sphere, a bunny and a noisy sphere) ($t = 0.17, 0.33, 0.5, 0.66, 0.83$)

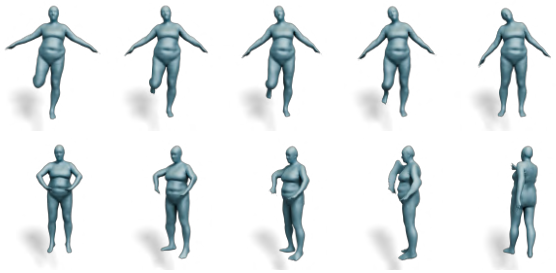

Fig. 16. Pairwise-to-any interpolation with a T-pose as canonical shape (faust1). First row: faust 3 to 5, 2nd row: faust 2 to 7 (t = 0, 0.25, 0.5, 0.75, 1).

Fig. 17. Advection of the input mesh using our flow in the pairwise-to-any case. First row: Faust 3 to 5, 2nd row: Faust 2 to 8 ($t = 0, 0.25, 0.5, 0.75, 1$).

## Acknowledgments

This work was partially funded by ANR-23-PEIA-0004 (PDE-AI). This project was provided with computing AI and storage resources by GENCI at IDRIS thanks to the grant 2025-AD010616975 on the supercomputer Jean Zay's V100 partition. We thank the Stanford Computer Graphics Laboratory for their publicly available 3D Scanning Repository (Armadillo and Bunny). The horse mesh is courtesy of Cyberware. The shapes from the Thingi10k dataset are provided under the following licenses: CC-BY-NC: owls (MOOSES), column (FRANCFALCO), submarine (CERBERUSS333); BY-NC-SA: blob (ZARQUON), falcon (COLINFIZGIG), tcross (MDB); BY-NC-ND: witch (DUTCHMOGUL); CC-BY-SA: initial (TARTURO), truggy (BARSPIN), tsphere (POLYMAKER) and the fandisk is from the aim@shape repository. We also thank the Max Planck Institute for the FAUST dataset and the authors of [Li et al. 2021] for the BEAR dataset.

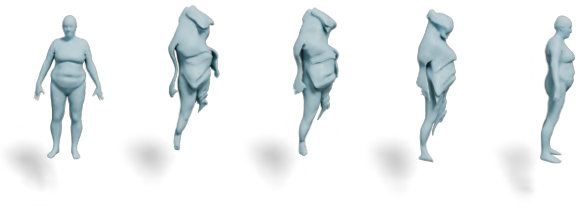

Fig. 18. Pairwise-to-any of a rotation of faust1 via a sphere ($t = 0, 0.25, 0.5, 0.75, 1$). We use 5 landmarks, one at the end of each limb and one on the head, matched to 5 coplanar sphere points.

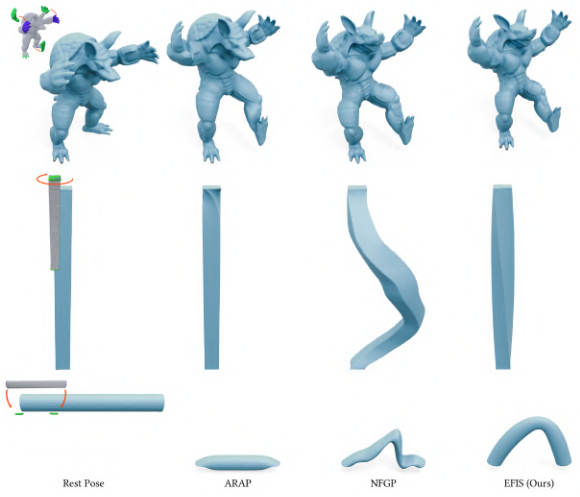


Fig. 19. Shape editing with our flow: a few vertices final positions are given. For NFGP, we use the code provided by[Yang et al. 2021] and the Open3D library for ARAP [Zhou et al. 2018]. Our method is able to handle larger deformations providing more intuitive and consistent deformation, at a lower computation time than NFGP.

## References

Matan Atzmon, David Novotny, Andrea Vedaldi, and Yaron Lipman. 2021. Augmenting Implicit Neural Shape Representations with Explicit Deformation Fields. *arXiv preprint arXiv:2108.08931* (2021).

Mirza Faisal Beg, Michael Miller, Alain Trouvé, and Laurent Younes. 2005. Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms. *International Journal of Computer Vision* 61 (02 2005), 139–157. doi:10.1023/B:VISI.0000043755.93987.aa

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. 2019. Invertible Residual Networks. arXiv:1811.00995 [cs.LG]

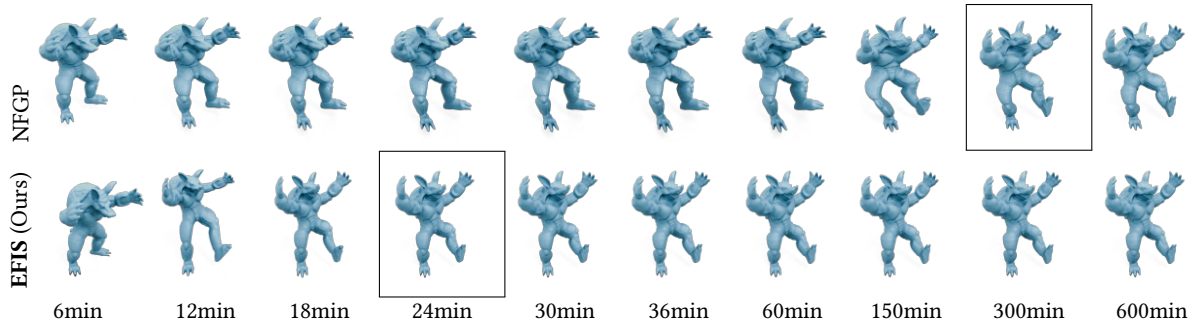


Fig. 20. Comparison of the optimization steps for our method and NFGP, for the shape editing task. We show the result obtained after various optimization times.

Fig. 21. Our method allows us to recover a smooth transition between the input and the edited shape ($t = 0, 0.17, 0.33, 0.5, 0.66, 0.83, 1$).

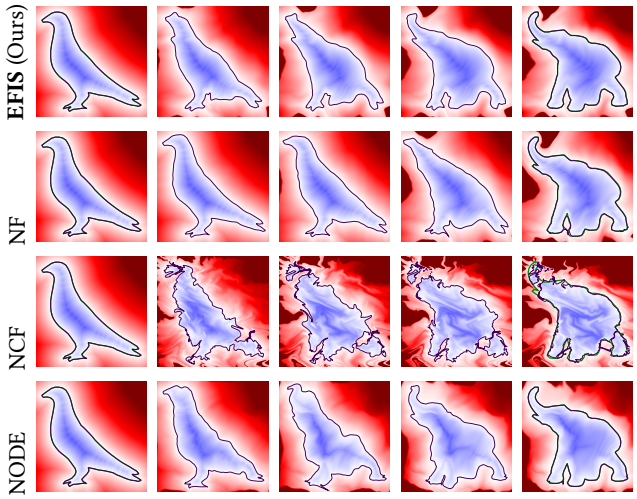


Fig. 22. Results for shape interpolation in 2D with various methods: EFIS (Ours), neural flows, neural conjugate flows and NODE. We display the advected implicit function value in addition to the 0 level set ($t = 0, 0.25, 0.5, 0.75, 1$).

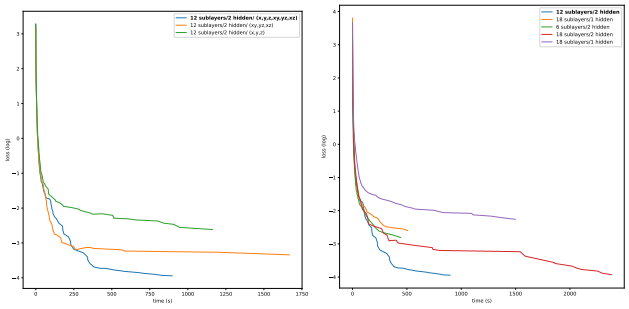

Fig. 23. Ablation of our architecture choices with respect to the number of sublayers and depth of the scaling and translation networks (left) and ways of splitting the coordinates for the sublayers (right). The training is stopped at convergence (Test case: bunny to column.)

Jean-David Benamou and Yann Brenier. 2000. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numer. Math.* 84, 3 (2000), 375–393.

Marcel Berger and Bernard Gostiaux. 2012. *Differential Geometry: Manifolds, Curves, and Surfaces: Manifolds, Curves, and Surfaces.* Vol. 115. Springer Science & Business Media.

Marin Biloš, Johanna Sommer, Syama Sundar Rangapuram, Tim Januschowski, and Stephan Günnemann. 2021. Neural Flows: Efficient Alternative to Neural ODEs. arXiv:2110.13040 [cs.LG]

Arthur Bizzi, Lucas Nissenbaum, and João M. Pereira. 2024. Neural Conjugate Flows: Physics-informed architectures with flow structure. arXiv:2411.08326 [cs.LG]

Jules Bloomenthal, Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, and Geoff Wyvill (Eds.). 1997. *Introduction to Implicit Surfaces.* Morgan Kaufmann, San Francisco, CA, USA.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).* IEEE, Piscataway, NJ, USA.

Camille Buonomo, Julie Digne, and Raphaëlle Chaine. 2025. Volume Preserving Neural Shape Morphing. *Computer Graphics Forum* 44, 5 (2025).

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2019. Neural Ordinary Differential Equations. arXiv:1806.07366 [cs.LG]

Guillaume Coiffier and Louis Béthune. 2024. 1-Lipschitz Neural Distance Fields. *Computer Graphics Forum* (2024). doi:10.1111/cgf.15128

Laurent Dinh, David Krueger, and Yoshua Bengio. 2015. NICE: Non-linear Independent Components Estimation. arXiv:1410.8516 [cs.LG]

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. arXiv:1605.08803 [cs.LG]

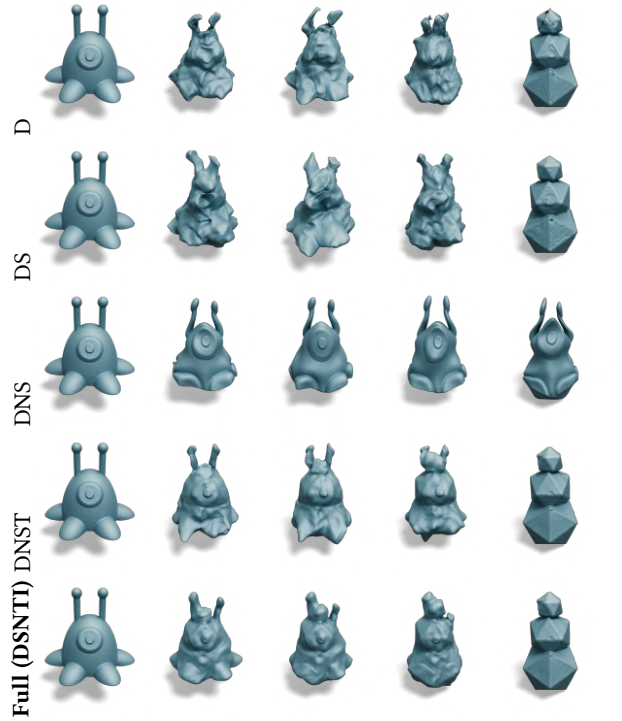


Fig. 24. Loss Ablation using various losses combination. D: Dirichlet, N: Neumann, S: sign, T: temporal regularization, I: isometry regularization ($t = 0, 0.25, 0.5, 0.75, 1$).

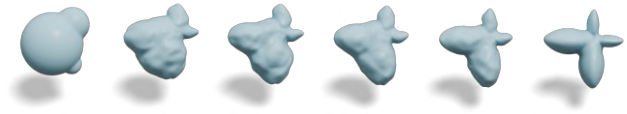

Fig. 25. Interpolation between shapes represented by Radial Basis Functions ($t = 0, 0.25, 0.5, 0.75, 1$).

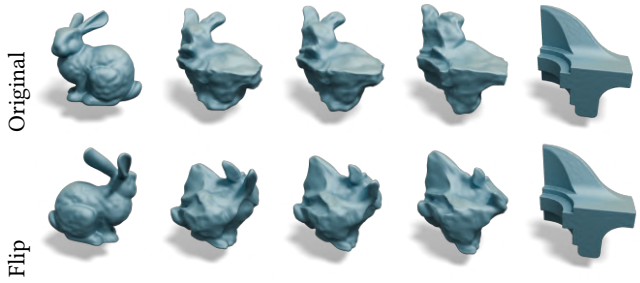


Fig. 26. Impact of the initial pose of the shapes ($t = 0, 0.25, 0.5, 0.75, 1$).

J.R. Dormand and P.J. Prince. 1980. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.* 6, 1 (1980).

Felix Draxler, Stefan Wahl, Christoph Schnörr, and Ullrich Köthe. 2024. On the universality of volume-preserving and coupling-based normalizing flows. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 461, 29 pages.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. 2019. Cubic-Spline Flows. arXiv:1906.02145 [stat.ML]

M. Eisenberger, Z. Lähner, and D. Cremers. 2019. Divergence-Free Shape Correspondence by Deformation. *Computer Graphics Forum* 38, 5 (2019), 1–12.

Jean Feydy, Benjamin Charlier, Francois-Xavier Vialard, and Gabriel Peyré. 2017. Optimal Transport for Diffeomorphic Registration. In *Medical Image Computing and Computer Assisted Intervention (MICCAI 2017), Lecture Notes in Computer Science*, Vol. 10433. Springer, 291–299. doi:10.1007/978-3-319-66182-7_34

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. 2020. How to train your neural ODE: the world of Jacobian and kinetic regularization. arXiv:2002.02798 [stat.ML]

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. 2018. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. arXiv:1810.01367 [cs.LG]

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. 2020. Implicit Geometric Regularization for Learning Shapes. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 3789–3799.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. 2019. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. arXiv:1902.00275 [cs.LG]

Priyank Jaini, Kira A. Selby, and Yaoliang Yu. 2019. Sum-of-Squares Polynomial Flow. arXiv:1905.02325 [cs.LG]

Diederik P. Kingma and Prafulla Dhariwal. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. arXiv:1807.03039 [stat.ML]

Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. 2021. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 11 (Nov. 2021), 3964–3979. doi:10.1109/tpami.2020.2992934

Bruno Lévy. 2015. A Numerical Algorithm for $L_2$ Semi-Discrete Optimal Transport in 3D. *ESAIM: Mathematical Modelling and Numerical Analysis* 49, 6 (2015), 1693–1715. doi:10.1051/m2an/2015055

Yang Li, Hikari Takehara, Takafumi Taketomi, Bo Zheng, and Matthias Niesner. 2021. 4DComplete: Non-Rigid Motion Estimation Beyond the Observable Surface . In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 12686–12696. doi:10.1109/ICCV48922.2021.01247

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. 2023. Flow Matching for Generative Modeling. arXiv:2210.02747 [cs.LG]

Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. 2022. Learning smooth neural functions via lipschitz regularization. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–13.

Gordon MacDonald, Andrew Godbout, Bryn Gillcash, and Stephanie Cairns. 2021. Volume-preserving Neural Networks. arXiv:1911.09576 [cs.LG]

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A Level Set Theory for Neural Implicit Evolution under Explicit Flows. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 711–729. doi:10.1007/978-3-031-20086-1_41

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4460–4470. doi:10.1109/CVPR.2019.00459

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. arXiv:1808.03856 [cs.LG]

Tiago Novello, Vinícius da Silva, Guilherme Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. 2023. Neural Implicit Surface Evolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 14279–14289.

Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. 2021. OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport. arXiv:2006.00104 [cs.LG]

Stanley Osher and Ronald P. Fedkiw. 2003. *Level Set Methods and Dynamic Implicit Surfaces*. Applied Mathematical Sciences, Vol. 153. Springer Science & Business Media, New York, NY, USA.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 165–174. doi:10.1109/CVPR.2019.00025

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. 2011. Wasserstein barycenter and its application to texture mixing. In *Proceedings of the Third International Conference on Scale Space and Variational Methods in Computer Vision* (Ein-Gedi, Israel) *(SSVM'11)*. 435–446.

Lu Sang, Zehranaz Canfes, Dongliang Cao, Florian Bernard, and Daniel Cremers. 2025. Implicit Neural Surface Deformation with Explicit Velocity Fields. In *ICLR*.

Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. arXiv:2006.09661 [cs.CV]

Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas J. Guibas. 2015. Convolutional Wasserstein Distances: Efficient Optimal Transportation on Geometric Domains. *ACM Transactions on Graphics* 34, 4 (2015), 66:1–66:11. doi:10.1145/2766963

Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *Proc. Symposium on Geometry Processing*.

Span Spanbauer and Luke Sciarappa. 2020. Neural Group Actions. arXiv:2010.03733 [stat.ML]

Jos Stam and Ryan Schmidt. 2011. On the velocity of an implicit surface. *ACM Trans. Graph.* 30, 3, Article 21 (May 2011), 7 pages. doi:10.1145/1966394.1966400

Michael Tao, Justin Solomon, and Adrian Butscher. 2016. Near-Isometric Level Set Tracking. *Computer Graphics Forum* 35, 5 (2016), 65–77. doi:10.1111/cgf.12964

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. 2024. Improving and generalizing flow-based generative models with minibatch optimal transport. arXiv:2302.00482 [cs.LG]

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural Fields in Visual Computing and Beyond. *Computer Graphics Forum* 41, 2 (2022), 641–676. doi:10.1111/cgf.14505

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. In *Advances in Neural Information Processing Systems*, Vol. 34. 22483–22497.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR]

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847* (2018).

Aiqing Zhu, Beibei Zhu, Jiawei Zhang, Yifa Tang, and Jian Liu. 2022. VP-Nets: Volume-preserving neural networks for learning source-free dynamics. arXiv:2204.13843 [cs.LG]

Zachary M. Ziegler and Alexander M. Rush. 2019. Latent Normalizing Flows for Discrete Sequences. arXiv:1901.10548 [stat.ML]

# Appendices

## A Link between flows, vector fields and ODEs

In this section, we link flows with ODEs and vector fields. While, in this paper, we do not consider the ODE directly and work solely with flows, it is worth linking our results to the ODE formulation for completeness. Let us consider a stationary vector field $\mathbf{V} : \mathbb{R}^d \to \mathbb{R}^d$ Lipschitz-continuous. Given such $\mathbf{V}$ and a given point $x_0 \in \mathbb{R}^d$ at time $t_0 \in \mathbb{R}$, we can define the Ordinary Differential Equation (ODE) for advecting point $x_0$ along the vector field $\mathbf{V}$:

$$\dot{x}(t) = \mathbf{V}(x(t)), \quad x(t_0) = x_0. \tag{A1}$$

The motion of a point through this ODE yields a flow.

Definition 3 (Flow of a vector field). *The flow $\phi$ induced by a vector field $\mathbf{V}$ is defined as the solution of Equation A1. This solution is unique as long as $\mathbf{V}$ is smooth. $\phi$ is mathematically a flow as defined in Definition 1.*

It follows directly that

$$\dot{\phi}(x_0, t) = \mathbf{V}(\phi(x_0, t)), \quad \phi(x_0, 0) = x_0. \tag{A2}$$

All of the above assumes that $\mathbf{V}$ is stationary. In particular, property 2 in Definition 1 does not hold if the advection vector field is non-stationary. However, one can extend it to a nonstationary vector field $\mathbf{V}(x, t)$ by lifting the problem to $\mathbb{R}^{d+1}$.

Let us rewrite Equation A1 for nonstationary vector fields:

$$\dot{x}(t) = \mathbf{V}(x, t), \quad x(t_0) = x_0. \tag{A3}$$

If $x$ is a solution of Equation A3, then we can define the higher dimensional flow $\phi : \mathbb{R}^{d+1} \times \mathbb{R} \to \mathbb{R}^d \times \mathbb{R}$:

$$\phi((x_0, t_0), t) = (x(t + t_0), t + t_0) \tag{A4}$$

One can show that $\phi((x_0, t_0), t)$ is the flow associated to the stationary $\mathbb{R}^{d+1}$ vector field:

$$\forall x \in \mathbb{R}^d, t \in \mathbb{R}, \quad \mathbf{W}((x, t)) = \begin{pmatrix} \mathbf{V}(x, t) \\ 1 \end{pmatrix}. \tag{A5}$$

Adding the time as a variable is a standard way to *stationarize* vector fields. Intuitively, one can think as $(x_0, t_0)$ as the position and time at which to start a flow and parameter $t$ as the duration of the advection by the flow.

## B Equality proof for implicit function definitions

We prove the assertion in subsection 5.2 that $f_1(x, t) = f_0(x, t)$.

Proof.

$$\begin{aligned} f_1(x, t) &= g_1(\phi((x, t), 1 - t)) \\ &= g_1(\phi(\phi((x, t), -t), 1)) \\ &= g_1(\phi((y, 0), 1)) \ \text{ with } (y, 0) = \phi((x, t), -t) \\ &= g_0(y) \\ &= g_0(\phi((x, t), -t)) \\ &= f_0(x, t) \end{aligned} \tag{A6}$$

Where we use Equation 10 for going from line 3 to line 4. Hence both $f_0$ and $f_1$ express an implicit solution of the morphing problem guided by flow $\phi$. □

## C Edition losses

In this section we define for completeness the losses used for edition, that are directly adapted from [Yang et al. 2021].

Let $D(x) = \phi((x, 1), -1)$ be the map between the final volume and the source volume and $g$ the signed distance field of the starting surface. The target shape $\tilde{\mathcal{S}}$'s implicit representation is given by $\tilde{g}(x) = g(\phi((x, 1), -1)) = g(D(x))$. Since bending and stretching should only be penalized on the surface, we define $P_g(x) = (I_d - n_g(x)n_g^T)(x)$ where $n_g(x) = \frac{\nabla g(x)}{\|\nabla g(x)\|}$, the projection operator onto $\mathcal{S}$, and similarly $P_{\tilde{g}}$ the projection operator onto $\tilde{\mathcal{S}}$.

*Bending loss.* The bending loss measures the curvature difference between the input surface and the edited surface. Information about the curvature of the surface is given by the Hessian of the implicit representation.

$$\mathcal{L}_{bend} = \int_{x \in \tilde{\mathcal{S}}} \|P_{\tilde{g}}(H_{\tilde{g}} - J_D^T H_g(D(x)) J_D) P_{\tilde{g}}\|_F^2 dx. \tag{A7}$$

Where $[H_g]_{ij}(x) = \partial_{ij} g(x)$ is the Hessian of $g$ and $[J_D]_{ij}(x) = \partial_i D_j(x)$ is the Jacobian of $D$.

*Stretching loss.* The stretching loss measures the infinitesimal surface changes induced by $D$. In practice, it is similar to the isometry loss of Equation 15 but applied only on the tangential space of the surface.

$$\mathcal{L}_{strech} = \int_{x \in \tilde{\mathcal{S}}} \|P_{\tilde{g}}(I_d - J_D^T J_D) P_{\tilde{g}}\|_F^2 dx. \tag{A8}$$

*Losses computation.* The above losses must be computed on the edited surface $\tilde{\mathcal{S}}$. Sampling $\tilde{\mathcal{S}}$ is costly since it is only known via its implicit representation that is not a signed distance field. Since the deformation is invertible, Yang et al. [Yang et al. 2021] propose to use the change of variable formula with $x = D^{-1}(y)$

$$\int_{x\in\tilde{\mathcal{S}}} \mathcal{L}(x)dx = \int_{y\in\mathcal{S}} \mathcal{L}(D^{-1}(y))w(D^{-1}(y))dy \tag{A9}$$

With $w(x) = |det(J_D(x)P_{\tilde{g}}(x) + n_g(D(x))n_{\tilde{g}}(x))|_F^{-2}$. We refer the reader to the section 5.3 of [Yang et al. 2021] for a more complete understanding of the implicit surface editing framework.

## D FAUST dataset naming convention

We selected a few human shapes from the FAUST_r dataset, which we renamed for readability as follows:

- Faust1: tr_reg_080
- Faust2: tr_reg_081
- Faust3: tr_reg_084
- Faust4: tr_reg_087
- Faust5: tr_reg_085
- Faust6: tr_reg_083
- Faust7: tr_reg_088
- Faust8: tr_reg_089
- Faust9: tr_reg_086

## E More details on Neural Flows [Biloš et al. 2021]

Neural Flows[Biloš et al. 2021] introduce a direct deformation of a probability distribution with a single pass through a neural network. By parameterizing a normalizing flow [Kobyzev et al. 2021] in time, Bilos et al. obtain continuous non-intersecting trajectories without relying on an integration scheme:

$$F(x,t) = \begin{pmatrix} x_I \\ \exp(s_\theta(x_I,t)\sigma(t))x_{\bar{I}} + \sigma(t)v_\theta(x_I,t) \end{pmatrix} \tag{A10}$$

Where $\sigma$ is such that $\sigma(0) = 0$ to ensure that $F(x,0) = x$. Since $F$ is invertible for all $t$, the trajectories cannot intersect at the same time: $F(x_1,t) = F(x_2,t) \Rightarrow x_1 = x_2$. But such a solution is not a flow as it does not verify the group law of flows $F(F(x,s),t) \neq F(x,s+t)$. In fact, $F$ corresponds to the solution of non-stationary ODE and as such, a reparameterization is necessary to obtain a flow. Moreover, the amplitude of the deformation is proportional to $\sigma(t)$. While this is not a limitation in the distribution matching framework, it is ill-suited for shape interpolation, which is our context. Since most of the deformation happens near $t = 1$, the optimization is non-symmetric. The morphing is biased towards the shape at time $t = 0$ as shown in our experiments (Figures 6, 22). In comparison, our optimization is fully symmetric, yielding an interpolation much smoother in time.