

Neural Field Processing on Surfaces

WEIZHOU LIU, Inria / École polytechnique, France

XINGYU NI*, Inria, France

MATHIEU DESBRUN, Inria / École polytechnique, France

JIONG CHEN, Inria, France

We introduce a unified framework for neural field processing on surfaces that combines implicit neural representations with the Closest Point Method (CPM), enabling intrinsic surface computation to be carried out in ambient Cartesian coordinates in a narrow band around the surface – with no explicit parameterization and no topology-dependent discretization. The framework rests on three technical contributions. First, we adapt CPM to the neural field setting: by extending field values constantly along surface normals, intrinsic differential operators reduce to their Cartesian counterparts and can be evaluated directly via a coordinate-based network. Second, we leverage the Hodge decomposition to encode surface vector fields entirely through neural scalar potentials, so that a single architecture handles scalar and vector surface fields alike, without frame-field choices or per-element discretization. Third, for near-degenerate geometries such as thin structures with closely spaced parallel faces, we lift the network input to the normal bundle to suppress leakage between sheets that are close in space but far apart along the surface. Under this formulation, a broad range of geometry processing tasks including surface PDE solving, Laplacian eigenfunction computation, reduced fluid simulation, and cross-field design for quad remeshing all reduce to variational problems solved by neural field optimization. Allowing boundary conditions or shape parameters as additional network inputs extends the benefit further: one network generalizes across an entire family of problem instances, replacing costly per-instance recomputation with fast feed-forward queries. Experimentally, we demonstrate accuracy competitive with FEM, a $30\times$ speedup over neural competitors in eigenfunction computation, and superior cross-field quality compared to NeurCross.

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

Additional Key Words and Phrases: Geometry Processing, Neural Fields, Manifold PDEs, Closest Point Method, Hodge Decomposition, Cross Fields

ACM Reference Format:

Weizhou Liu, Xingyu Ni, Mathieu Desbrun, and Jiong Chen. 2026. Neural Field Processing on Surfaces. In *SIGGRAPH Asia 2026 Conference Papers (SA Conference Papers '26)*, December 01–04, 2026, Kuala Lumpur, Malaysia. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3829340.3842300>

1 Introduction

The past decade has seen remarkable advances in deep learning, particularly in natural language processing and generative modeling. These developments have prompted renewed interest in revisiting

*Corresponding author.

Authors' Contact Information: Weizhou Liu, weizhou.liu@inria.fr, Inria / École polytechnique, Palaiseau, France; Xingyu Ni, xingyu.ni@inria.fr, Inria, Palaiseau, France; Mathieu Desbrun, mathieu.desbrun@inria.fr, Inria / École polytechnique, Palaiseau, France; Jiong Chen, jiong.chen@inria.fr, Inria, Palaiseau, France.



This work is licensed under a Creative Commons Attribution 4.0 International License. SA Conference Papers '26, Kuala Lumpur, Malaysia
© 2026 Copyright held by the owner/authors(s).
ACM ISBN 979-8-4007-2842-6/2026/12
<https://doi.org/10.1145/3829340.3842300>

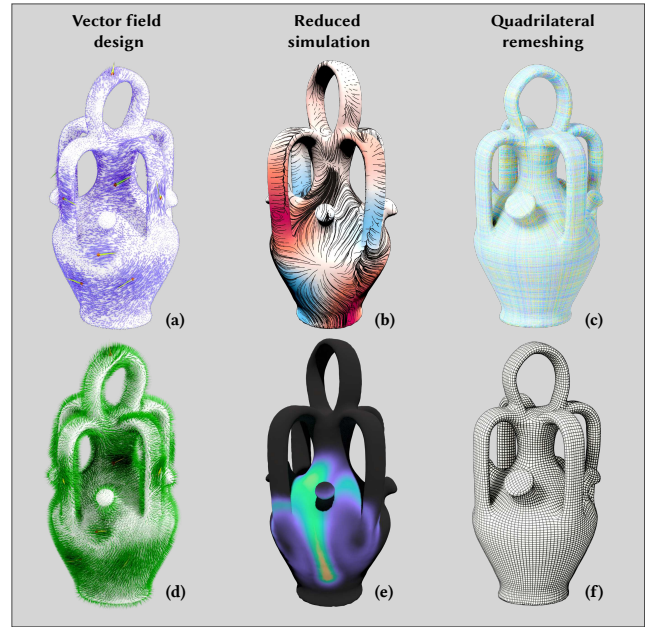


Fig. 1. **Neural surface fields.** Our approach supports a broad range of neural fields on surfaces: vector fields (a) for surface pattern design (d), divergence-free vector fields (b) for Laplacian eigenfluid simulation (e), and cross-frame fields (c) for quad-remeshing (f). By leveraging the Hodge decomposition to encode these various fields solely through neural scalar potentials, our approach provides a unified representation that enables generalization across varying boundary conditions and shape spaces.

classical problems grounded in PDEs and optimization through learning. For example, neural fields paired with volume rendering have achieved striking results in novel-view synthesis [Mildenhall et al. 2021; Müller et al. 2022], while related approaches have been successfully applied to tasks such as implicit surface reconstruction [Wang et al. 2021], scene deformation [Park et al. 2021], and fluid simulation [Deng et al. 2023]. These successes suggest a shift away from traditional discrete representations toward neural ones, motivated by their ability to model fields as differentiable functions in both supervised and unsupervised learning frameworks.

Surface field modeling is inherently more challenging than its volumetric equivalent. Classical methods, such as the finite element method or discrete exterior calculus, define operators on discrete meshes or point clouds [Desbrun et al. 2005; Sharp and Crane 2020]. However, continuous neural models often struggle to remain faithful to the underlying manifold when faced with complex or near-degenerate geometry. Moreover, there is a distinct lack of a unified representation-agnostic framework capable of addressing diverse surface processing tasks without resorting to ad-hoc design choices.

Motivated by these challenges, we introduce a simple, streamlined approach for processing neural surface fields. By combining the Closest Point Method (CPM) with neural fields, we compute intrinsic differential operators directly through ambient Cartesian derivatives in a narrow band around the surface, further lifting the network input to the normal bundle to enhance robustness on near-degenerate geometries. Although initially formulated for scalar fields, we demonstrate that our method seamlessly extends to vector-field learning via Hodge decomposition. Finally, conditioning the network on task or shape parameters allows a single model to generalize across diverse boundary conditions and shape spaces, significantly reducing the recomputation overhead typically required by traditional numerical solvers.

2 Related Work

Before presenting our approach, we briefly review a number of related works to motivate and contextualize it.

Neural Representation for Shapes. Recent advances in neural implicit representations have significantly reshaped geometry processing. Unlike traditional explicit structures (e.g., point clouds, voxels, and meshes), these methods represent geometry as continuous, differentiable functions, allowing flexible topology and direct integration with optimization. This shift was catalyzed in part by Occupancy Networks [Mescheder et al. 2019] and DeepSDF [Park et al. 2019], which use multilayer perceptrons (MLPs) to model continuous geometry from large 3D datasets, learning strong shape priors for reconstruction and generation. The introduction of Neural Radiance Fields [Mildenhall et al. 2021] further advanced the field by combining implicit representations with volume rendering, enabling high-quality surface reconstruction from multi-view images — a task that has historically challenged traditional methods — and achieving impressive results in large-scale scene reconstruction [Yariv et al. 2020, 2021; Wang et al. 2021; Li et al. 2023a].

Beyond volume rendering, neural surfaces have also been integrated with classical geometry processing. A common strategy is to overfit a neural network to a given shape and then define differential operators (e.g., gradient, Laplacian, etc.) on the learned representation for downstream tasks. In practice, the network can remain fixed as an intermediate mapping that transforms geometry into a canonical domain (e.g., disks [Williamson and Mitra 2025]) to simplify processing, or it can be iteratively refit to accommodate evolving shapes [Yang et al. 2021; Mehta et al. 2022; Buonomo et al. 2025]. Note that Buonomo et al. [2025] also apply a Hodge-inspired decomposition to their neural velocity field, advecting implicit surfaces for volume-preserving shape morphing.

Neural Representation for PDE Solutions. An alternative way to integrate neural representations into geometry processing is to encode the solutions of PDEs rather than the geometry itself. In this setting, the geometry merely defines the computational domain and boundary conditions. Our work follows this approach, which has gained momentum since the introduction of Physics Informed Neural Networks (PINNs) [Raissi et al. 2019], which incorporate governing equations directly into the loss function and enable mesh-free solution without requiring simulation data. Since then, implicit neural representations have been widely explored to improve classical

numerical methods along two main directions. First, they improve the trade-off between accuracy and memory usage, for instance in generalized barycentric coordinates [Dodik et al. 2023], topology optimization [Zehnder et al. 2021], and broader simulation settings [Deng et al. 2023; Chen et al. 2023; Liu et al. 2025; Chang et al. 2023]. Second, they enable generalization across varying initial conditions and PDE parameters, reducing the need for case-by-case recomputation, as demonstrated in eigenvector computation across shape spaces [Chang et al. 2025a] and neural operators [Li et al. 2021].

However, most progress has been focused on volumetric domains, while neural methods for surface-based PDEs remain less explored. Recently, Wang et al. [2025] represented the stream function on surfaces to solve fluid simulation via CPM, achieving favorable accuracy-storage trade-offs compared to classical approaches. Also building on CPM, Welschinger et al. [2026] learned local solution updates for Poisson and heat equations conditioned on geometry. Their method fits within the operator-learning paradigm, predicting solutions directly from inputs under supervision from large datasets, whereas we focus on the representational power of neural fields. In quad meshing, NeurCross [Dong et al. 2025] jointly optimizes a neural SDF and a cross field on the surface; however, these cross fields are still discretized into per-element frames, limiting global continuity, and the associated U-Nets introduce memory and computational costs that scale poorly with mesh resolution.

Closest Point Method. The Closest Point Method, introduced by Ruuth and Merriman [2008], provides a straightforward way to extend surface-defined functions into the surrounding space by assigning to each point in the embedding domain the value of its nearest point on the surface. This construction transforms PDEs on manifolds into equivalent volumetric PDEs in Cartesian coordinates, enabling the use of standard numerical solvers across different surface representations. An implicit formulation was later developed by MacDonald and Ruuth [2010], improving stability and allowing larger time steps while maintaining the accuracy of the discretization. The method’s theoretical justification was subsequently established by März and Macdonald [2012], who rigorously showed that intrinsic differential operators can be replaced by their Cartesian counterparts under this embedding. In computer graphics, CPM has been further extended and adapted to a range of applications. Kim et al. [2013], for example, used it for wave simulation on deforming liquid surfaces, effectively enhancing coarse volumetric simulations. King et al. [2024] generalized the framework to support interior boundary conditions and both open and closed surfaces of arbitrary codimension for geometry processing tasks. More recently, Sugimoto et al. [2024] combined CPM with a Monte-Carlo approach, computing solutions through random walks and projections, thereby bypassing explicit spatial discretization. The underlying Walk-on-Spheres estimator, however, applies only to linear scalar PDEs, precluding the nonlinear, vector- and cross-field processing tasks that we target.

3 Recap of the Closest Point Method

This section provides a brief overview of the Closest Point Method (CPM); see the work of King et al. [2024] for more detail.

Closest Point Projection. The core idea of CPM is to reformulate a problem posed on a surface $\mathcal{M}$ embedded in $\mathbb{R}^3$ as an equivalent problem over a tubular neighborhood $\mathcal{N}(\mathcal{M}) \subset \mathbb{R}^3$ surrounding $\mathcal{M}$. The key ingredient of CPM is the closest point projection π :

$$\pi(\mathbf{x}) = \arg \min_{\mathbf{y} \in \mathcal{M}} \|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x} \in \mathcal{N}(\mathcal{M}),$$

which assigns to each point $\mathbf{x} \in \mathcal{N}$ the nearest point on $\mathcal{M}$. This map is not unique in general, since multiple surface points may be equidistant from $\mathbf{x}$. Yet, uniqueness can be ensured by requiring the tubular radius to be smaller than the local feature size of $\mathcal{M}$ at the projected points [Sugimoto et al. 2024].

Closest Point Extension. The projection π provides a natural way to extend fields defined on $\mathcal{M}$ to the full neighborhood $\mathcal{N}$. Given a surface scalar field f or a vector field $\mathbf{v}$, their extensions to $\mathcal{N}$, denoted by $\tilde{f}$ and $\tilde{\mathbf{v}}$, respectively, are defined pointwise by

$$\tilde{f}(\mathbf{x}) := f(\pi(\mathbf{x})), \quad \tilde{\mathbf{v}}(\mathbf{x}) := \mathbf{v}(\pi(\mathbf{x})), \quad \forall \mathbf{x} \in \mathcal{N}(\mathcal{M}).$$

Note that $\tilde{\mathbf{v}}(\mathbf{x}) \in \mathcal{T}_{\pi(\mathbf{x})}\mathcal{M} \subset \mathbb{R}^3$, where $\mathcal{T}_{\pi(\mathbf{x})}\mathcal{M}$ denotes the tangent space of $\mathcal{M}$ at $\pi(\mathbf{x})$. Since the closest point projection maps each $\mathbf{x}$ to its nearest point on $\mathcal{M}$, the extended fields are essentially constant along directions normal to the surface (see inset). This also equivalently leads to the first-order conditions, that is, for $\mathbf{x} \in \mathcal{N}$,

$$\nabla \tilde{f}(\mathbf{x}) \cdot \tilde{\mathbf{n}}(\mathbf{x}) = 0, \quad \nabla \tilde{\mathbf{v}}(\mathbf{x}) \tilde{\mathbf{n}}(\mathbf{x}) = \mathbf{0}, \quad (1)$$

where $\nabla \tilde{\mathbf{v}}$ denotes the Jacobian matrix of $\tilde{\mathbf{v}}$, and $\tilde{\mathbf{n}}(\mathbf{x}) = \mathbf{n}(\pi(\mathbf{x}))$ is the *extended* normal field in $\mathcal{N}$.

Closest-Point Calculus. The constant-along-normal property allows the evaluation of intrinsic differential operators on the input manifold through standard Cartesian operators on the extended fields within $\mathcal{N}$, without needing explicit projection. Specifically, for a scalar function f and a vector field $\mathbf{v}$, we have

$$\begin{aligned} \text{grad}_{\mathcal{M}} f &\equiv \nabla \tilde{f}, & \text{rot}_{\mathcal{M}} f &\equiv \tilde{\mathbf{n}} \times \nabla \tilde{f}, \\ \text{curl}_{\mathcal{M}} \mathbf{v} &\equiv \tilde{\mathbf{n}} \cdot (\nabla \times \tilde{\mathbf{v}}), & \text{div}_{\mathcal{M}} \mathbf{v} &\equiv \nabla \cdot \tilde{\mathbf{v}}, \end{aligned} \quad (2)$$

where $\tilde{\mathbf{n}}$ coincides with the surface normal, and all quantities are evaluated on $\mathcal{M}$. Thus, the CPM extension offers a canonical method to realize intrinsic differential operators in $\mathbb{R}^3$ which can also be described using the language of DEC [Li et al. 2023b]. It is precisely this equivalence that enables the application of neural representations to surface-based geometry processing.

4 Method

We now present our method for processing surface fields with neural networks. Each network is trained to represent a mapping from a point in the tubular neighborhood $\mathcal{N}(\mathcal{M})$ of the surface $\mathcal{M}$ to a scalar value. It can be used either directly to model the extension of a surface-based scalar field, or indirectly (via Hodge decomposition) to represent the extension of a surface vector field, see Sec. 5. For simplicity, we omit the tilde notation that distinguishes the extended field $\tilde{f}$ from its surface counterpart f (and similarly for $\mathbf{n}$), and use f_θ to denote the neural representation of f , where θ denotes the trainable parameters. We also temporarily ignore the network inputs that encode the configuration space — those associated with

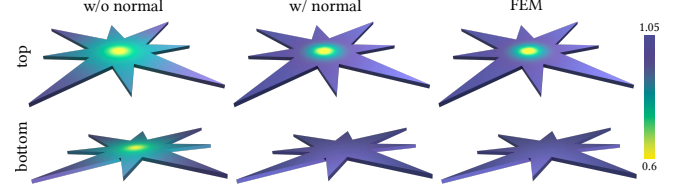


Fig. 2. **Input lifting.** Using only point coordinates $\mathbf{x}$ as input, our neural solver for a screened Poisson equation on this thin structure suffers from spurious source “leakage” across opposite surfaces, and the relative L_2 error to the ground-truth FEM solution is 23.5% in this case. Incorporating point normals $\mathbf{n}$ suppresses the artifact, reducing the relative error to 1.36%.

boundary conditions or shape spaces — and defer the discussion of this generalization capability to Sec. 5.4.

4.1 Learning Surface Fields under CPM Constraints

For a given surface $\mathcal{M}$, many geometry processing problems can be formulated as the minimization of an energy functional defined over a field on the surface. Without loss of generality, we consider a single scalar field f defined on $\mathcal{M}$ — extensions to a vector field can be handled in an analogous manner using the potentials of a Hodge decomposition [Crane et al. 2013]. The problems studied in this work can be expressed in the following unified form:

$$\min_{\theta} \mathcal{E}_{\mathcal{M}}[f_{\theta}(\mathbf{x}), \nabla f_{\theta}(\mathbf{x}), \nabla^2 f_{\theta}(\mathbf{x})], \quad \text{s.t. } \mathcal{C}_{\mathcal{M}}[f_{\theta}(\mathbf{x}), \nabla f_{\theta}(\mathbf{x})] = \mathbf{0},$$

where $\mathcal{E}_{\mathcal{M}}$, $\mathcal{C}_{\mathcal{M}}$ denote the energy functional and the collection of equality constraints defined on the surface, respectively, arising from specific applications. Note that Cartesian gradients and higher-order derivatives are used throughout, with the requisite surface differential operators evaluated according to CPM-induced Eq. (2).

However, computing surface differential operators via Cartesian derivatives requires the function to be properly extended within the tubular region so that it satisfies the constant-along-normal condition in Eq. (1). We thus introduce the following CPM loss:

$$\mathcal{L}_{\text{cp}}[f_{\theta}(\mathbf{x})] = \int_{\mathcal{N}} (\mathbf{n}(\mathbf{x}) \cdot \nabla f_{\theta}(\mathbf{x}))^2 d\tau, \quad (3)$$

which is added to the energy functional $\mathcal{E}_{\mathcal{M}}$ during optimization.

Note that we adopt a first-order condition here (enforcing the *constant-along normal* condition) rather than a zeroth-order one, that would merely penalize the discrepancy $|f_{\theta}(\mathbf{x}) - f_{\theta}(\pi(\mathbf{x}))|^2$ between the function value at a point and its closest-point projection. Because most energy functionals depend on derivatives of the surface field rather than its actual values, enforcing the first-order condition directly promotes consistency between surface differential operators and their Cartesian counterparts. While constraining the extended function values is feasible in practice, it enforces the equivalence of the differential operators only indirectly. This can reduce the efficacy of CPM, much as finite-difference approximations of gradients introduce inaccuracies.

4.2 Lifting the Input to the Normal Bundle

The CPM provides a tool to restrict Euclidean differential operators to the underlying geometry, but for neural representations, this approach alone is sometimes insufficient. As illustrated in Fig. 2, a scalar source diffusing from a flat face should decay according

to the geodesic distance. However, the diffusion “leaks” across the ambient space to adjacent, oppositely oriented faces, even though the true geodesic distance between these regions is large. This failure may stem from the spectral bias of neural networks [Rahaman et al. 2019], which inherently favors globally smooth solutions, and consequently, the network struggles to distinguish between two close, parallel faces, smoothing over the narrow gap between them.

To resolve these near-degenerate cases, we need to introduce spatial non-smoothness to the neural field. Inspired by Chang et al. [2025b], who lifted spatial coordinates into higher dimensions using a winding number field to inject discontinuities, we analogously utilize the *surface normal field* $\mathbf{n}(\mathbf{x})$ as an indicator of geometry. By lifting the input via the map $L: \mathbf{x} \mapsto (\mathbf{x}, \mathbf{n}(\mathbf{x}))$ (thus, to the higher-dimensional space that is the *normal bundle* of the input surface), we allow the network to capture surface-adapted non-smoothness within the tubular region. Our neural field is thus defined as

$$f_\theta : \mathcal{N}(\mathcal{M}) \times \mathbb{S}^2 \rightarrow \mathbb{R}, \quad (4)$$

where $\mathbb{S}^2$ denotes the unit sphere in which normal vectors reside. To ensure scale invariance across inputs, we always normalize the input surface to fit in $[-1, 1]^3$ prior to training.

Because the normal $\mathbf{n}$ is a function of $\mathbf{x}$, spatial gradients of the field must be evaluated via the chain rule, yielding:

$$\nabla_{\mathbf{x}} f_\theta(\mathbf{x}, \mathbf{n}(\mathbf{x})) = \partial_{x_i} f_\theta + (\partial_{x_i} n_j) \partial_{n_j} f_\theta, \quad i, j \in \{1, 2, 3\}, \quad (5)$$

where we used Einstein summation convention, i.e., repeated indices are summed over their full range, while x_i and n_j denote the i^{th} and j^{th} components of $\mathbf{x}$ and $\mathbf{n}$, respectively. Terms in green are network-related, thus evaluated via auto differentiation and updated during optimization, while terms in purple represent fixed geometric properties such as the extended shape operator $\partial_{x_i} n_j$. For vector field processing, we also need the Hessian of the scalar function, computed via a second application of the chain rule:

$$\begin{aligned} \nabla_{\mathbf{x}\mathbf{x}}^2 f_\theta(\mathbf{x}, \mathbf{n}(\mathbf{x})) &= \partial_{x_k} \partial_{x_i} f_\theta + \partial_{x_k} n_l (\partial_{n_l} \partial_{x_i} f_\theta) + (\partial_{x_k} \partial_{x_i} n_j) \partial_{n_j} f_\theta \\ &\quad + \partial_{x_i} n_j (\partial_{x_k} \partial_{n_j} f_\theta + \partial_{x_k} n_l (\partial_{n_l} \partial_{n_j} f_\theta)), \quad i, j, k, l \in \{1, 2, 3\}. \end{aligned} \quad (6)$$

The Laplacian then follows as the trace of the Hessian, that is to say, $\nabla \cdot \nabla f_\theta = \text{tr}(\nabla_{\mathbf{x}\mathbf{x}}^2 f_\theta)$.

4.3 Extending the Surface Normal Field

Both the lifting map and the CPM constraint require extending the surface normal into the tubular neighborhood $\mathcal{N}(\mathcal{M})$. While these normals can be computed on-the-fly via closest-point projection, doing so during network training is computationally expensive. To improve efficiency, we follow [King et al. 2024] and tessellate $\mathcal{N}(\mathcal{M})$ with a regular grid. We precompute the extended normals $\{\mathbf{n}(\mathbf{x}_p)\}_p$ at each grid node $\mathbf{x}_p$, allowing the normal $\mathbf{n}(\mathbf{x})$ at an arbitrary point to be efficiently evaluated via trilinear interpolation.

To ensure smoothness in the derivative fields, we avoid direct differentiation of the trilinear basis functions, as this results in discontinuities across cell boundaries, hindering learning efficiency. Instead, we precompute the Jacobian at each node $\mathbf{x}_p$ using central

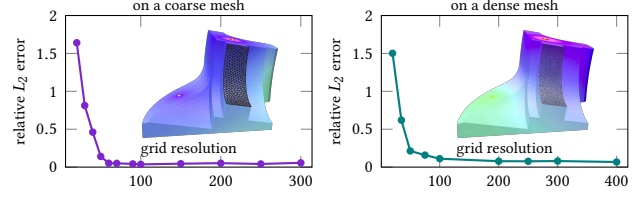
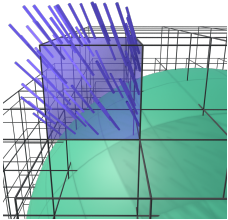


Fig. 3. **Narrow-band resolution.** The grid spacing of the narrow band affects the accuracy of the normal extension, and hence of the extrinsic approximation of the surface differential operators. Here, we solve the screened Poisson equation with point sources at varying grid resolutions and measure the L_2 error relative to an FEM reference. A coarse grid introduces significant errors, obviously. Under grid refinement, the error decreases consistently and plateaus once the spacing drops below the mesh edge length (2/50 left, 2/100 right). We therefore adopt a conservative resolution of $500 \times 500 \times 500$ throughout our experiments.

differences along each axis direction:

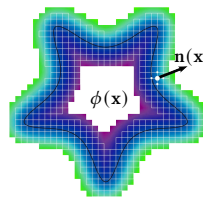
$$\partial_{x_i} \mathbf{n}(\mathbf{x}_p) = \frac{\mathbf{n}(\mathbf{x}_p + h\mathbf{e}_i) - \mathbf{n}(\mathbf{x}_p - h\mathbf{e}_i)}{2h}, \quad i \in \{1, 2, 3\}, \quad (7)$$

where h is the grid spacing and $\mathbf{e}_i \in \mathbb{R}^3$ is the i^{th} standard basis vector. We apply this scheme analogously for second-order derivatives, storing the resulting third-order tensors on the grid. During training, these precomputed nodal values are trilinearly interpolated and combined with the auto-differentiated derivatives of f_θ to evaluate the spatial gradient (Eq. (5)) and Hessian (Eq. (6)) of our CPM-extended neural field. Note that for surfaces with sharp features, this approach inherently smooths out the underlying geometric discontinuities. Although the resulting field does not perfectly adhere to these sharp features, refining the discretization within $\mathcal{N}$ effectively mitigates the surface approximation error in practice. Fig. 3 illustrates the asymptotic behavior of the neural field under grid refinement.

4.4 Implementation Details

Next we provide implementation details — note that our code will be released at <https://gitlab.inria.fr/geomerix/public/NeuralCPM>.

Narrow-Band Signed Distance Field. To extend the normal field from the surface into the tubular region, a natural approach would be to locate each grid point’s closest point on the surface and query the normal there. Instead, we adopt a narrow-band signed distance function, which not only leads to a simpler implementation using existing libraries, but also renders the method agnostic to the choice of input surface representation. Concretely, given an input surface (triangular meshes in our experiments), we begin by constructing a narrow-band signed distance field (SDF) using fVDB [Williams et al. 2024]. The resulting sparse voxels discretize the tubular region, and



the extended normals follow naturally via $\mathbf{n}(\mathbf{x}) = \nabla \phi$. In practice, we evaluate this gradient using a central difference scheme at each grid vertex (see inset) and interpolate trilinearly throughout the narrow band. The discrete derivatives are then obtained through Eq. (7), as well as higher-order ones. In all experiments, we use a grid resolution of 500^3 and a narrow band of 3 voxels on each side of the surface. A very coarse grid (e.g., 20^3 over $[-1, 1]^3$) may not represent the underlying surface faithfully and



Fig. 4. **Solving screened Poisson equation.** As the prescribed sources diffuse across the entire surface, our neural approach (left) achieves a low approximation error compared to standard FEM-based methods (middle). The pointwise error (right) is computed as $|u_i^{\text{ours}} - u_i^{\text{fem}}| / \|u^{\text{fem}}\|_{\infty}$.

incurs large errors, as Fig. 3 shows. We thus adopt this conservative resolution to keep the CPM extension reliable.

Monte-Carlo Integration. The integrals over the surface and the tubular neighborhood are both approximated via Monte-Carlo sampling. For surface integrals, quadratures are drawn randomly over each triangle through barycentric interpolation, with the number of quadratures per face allocated proportionally to its area. For the ambient narrow band, we first sample the surface again and then perturb these quadratures by adding Gaussian noise drawn from $N(0, hI_{3 \times 3})$, where h denotes the voxel size. In both cases, the integral is finally approximated as the expectation of the integrand over the corresponding sample set.

Neural Network and Optimizer. Throughout all experiments, we employ a shallow SIREN network [Sitzmann et al. 2020] consisting of five hidden layers with 128 neurons each. The input is a concatenation of the point coordinates $\mathbf{x}$ and its associated normal $\mathbf{n}$, forming a vector in $\mathbb{R}^6$, while the output is a single scalar. Each component of coordinate $\mathbf{x}$ is further mapped through a cosine positional encoding using frequency bands up to 2^5 . The network is optimized using Adam [Kingma and Ba 2017].

5 Applications and Results

This section first presents several applications of our surface fields, followed by a demonstration of the neural network’s generalization capability (Sec. 5.4) — arguably the most valuable feature of neural representations. Experimental details and additional visual and quantitative results are provided in the supplemental material.

5.1 Solving Dirichlet Problems on Surfaces

Computing smooth fields from sparse constraints underpins a wide range of graphics applications. Here we present two such examples: solving screened Poisson equations and designing vector fields.

Screened Poisson Equations on Surfaces. As a first application, we show how to solve the screened Poisson equation on a surface, which has found applications in tasks such as surface filtering [Chuang and Kazhdan 2011], gradient filtering [Tojo and Umetani 2025], and geodesic distance computation [Belyaev and Fayolle 2015]. Given a set of curves Γ defined on a surface with Dirichlet boundary data $\bar{u}$, the screened equation can be formulated as a minimization problem:

$$\min_u \int_{\mathcal{M}} (\|\text{grad}_{\mathcal{M}} u\|^2 + u^2) d\sigma + \lambda_C \int_{\Gamma} (u(\mathbf{x}) - \bar{u}(\mathbf{x}))^2 ds + \lambda_{\text{cp}} \mathcal{L}_{\text{cp}}[u], \quad (8)$$

where the screening factor is assumed to be 1, and the Dirichlet constraints are enforced via a penalty function with weight λ_C .

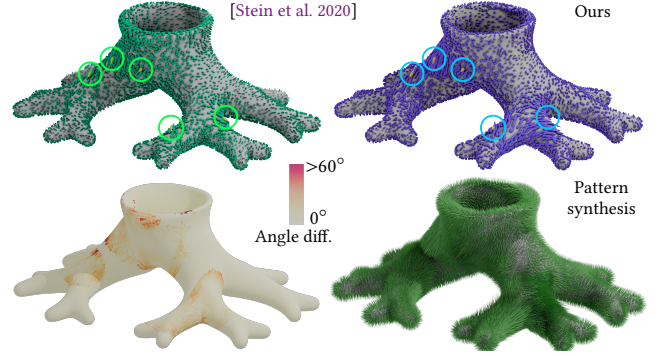


Fig. 5. **Vector field design.** We synthesize a smooth vector field constrained by given anchor vectors (marked in circles), yielding results consistent with an FEM baseline [Stein et al. 2020]. Angular difference is computed after filtering out small vectors with magnitudes below 20% of the maximum. This synthesized field can be used for surface pattern generation — for instance, blending the surface normal $\mathbf{n}(\mathbf{x})$ with the neural vector field $\mathbf{u}(\mathbf{x})$ mimics grass growth over the plant (bottom right).

In Fig. 4, we show an example of the screened Poisson equation with four prescribed point constraints ($\bar{u} = 1$), and compare our result against an FEM implementation based on libigl [Jacobson et al. 2018]. On this Stanford bunny surface, our method agrees with the reference FEM solution to within a relative L_2 error of 1.82%.

Vector Field Design. Vector field design is a fundamental problem in geometry processing, serving as a building block for many applications [Vaxman et al. 2016; Stein et al. 2020]. Typically, this problem is formulated as the minimization of a Dirichlet energy of a tangential vector field $\mathbf{u}$ on a surface, subject to sparse Dirichlet constraints defined at a set of anchor points $\mathcal{A} = \{\mathbf{x}_i \in \mathcal{M} \mid \mathbf{u}(\mathbf{x}_i) = \bar{\mathbf{u}}_i\}$ across the surface. Here, we adopt the *Hodge-Dirichlet energy* $\mathcal{E}_{\mathcal{D}}[\mathbf{u}] = \int_{\mathcal{M}} (\|\text{div}_{\mathcal{M}} \mathbf{u}\|^2 + \|\text{curl}_{\mathcal{M}} \mathbf{u}\|^2) d\sigma$, i.e., the quadratic form of the Hodge Laplacian (see supplemental material, Sec. B).

Directly learning surface vector fields with a neural network is fraught with difficulties, as comparing vectors to compute derivatives requires a choice of frame field. Fortunately, the *Hodge decomposition* offers a straightforward remedy: on a simply connected genus-0 surface for instance, any vector field can be fully parameterized through two scalar potentials as $\mathbf{u} = \text{grad}_{\mathcal{M}} f + \text{rot}_{\mathcal{M}} g$, where the harmonic component vanishes in this case. Since $\text{div}_{\mathcal{M}} \text{rot}_{\mathcal{M}} g = 0$ and $\text{curl}_{\mathcal{M}} \text{grad}_{\mathcal{M}} f = 0$, the two potentials further decouple in the Hodge-Dirichlet energy, yielding

$$\mathcal{E}_{\mathcal{D}}[f, g] = \int_{\mathcal{M}} (\|\text{div}_{\mathcal{M}} \text{grad}_{\mathcal{M}} f\|^2 + \|\text{curl}_{\mathcal{M}} \text{rot}_{\mathcal{M}} g\|^2) d\sigma, \quad (9)$$

where the surface differential operators are evaluated via Eq. (2). This decomposition naturally motivates a dual-network architecture, in which two separate networks — each with its own learnable parameters — encode the scalar potential f and the stream function g , respectively. The full optimization objective combines the Hodge-Dirichlet energy with an anchor constraint penalty, CPM losses, and a gauge-fixing term that eliminates the trivial constant kernel shared by both potentials, i.e.,

$$\min_{f, g} \mathcal{E}_{\mathcal{D}}[f, g] + \lambda_C \mathcal{L}_C[f, g] + \lambda_{\text{cp}} (\mathcal{L}_{\text{cp}}[f] + \mathcal{L}_{\text{cp}}[g]) + \lambda_{\mathcal{G}} \mathcal{L}_{\mathcal{G}}[f, g], \quad (10)$$

where $\mathcal{L}_C = \sum_{\mathbf{x}_i \in \mathcal{A}} \|\text{grad}_{\mathcal{M}} f(\mathbf{x}_i) + \text{rot}_{\mathcal{M}} g(\mathbf{x}_i) - \bar{\mathbf{u}}_i\|^2$ enforces the anchor constraints and $\mathcal{L}_G = (\int_{\mathcal{M}} f \, d\sigma)^2 + (\int_{\mathcal{M}} g \, d\sigma)^2$ is the gauge regularizer that pins the mean of each potential to zero.

In Fig. 5, we compare our method with the FEM-based method of Stein et al. [2020], which discretizes the covariant-derivative (Bochner) Dirichlet energy. Our Hodge energy and theirs differ by a curvature term, so exact agreement is not expected; nonetheless, we visualize the angular discrepancy between the two resulting vector fields and find our result visually almost indistinguishable from the FEM one. We further demonstrate how this field can be used for surface pattern synthesis. In addition, Fig. 13 shows instantly generated vector fields with varied anchor orientations as a form of generalization.

5.2 Computing Eigenfunctions on Surfaces

Eigenfunctions are indispensable for shape analysis and reduced simulation. Here, we present our neural approach for computing scalar and vector Laplacian eigenfunctions on surfaces.

Scalar Laplacian Eigenfunctions. Chang et al. [2025a] used neural networks to compute Laplacian eigenfunctions for volumetric domains, demonstrating that the results can generalize to new geometries within a shape space. With the CPM, their method can be readily extended to surface Laplacian eigenfunctions. Similarly, each eigenfunction is parameterized by an independent network ϕ_i with its own dedicated parameters θ_i . During each training epoch, these networks update sequentially, each minimizing an objective consisting of a Rayleigh quotient ($\mathcal{L}_{\text{RQ}}$), orthogonality and normalization penalties ($\mathcal{L}_{\perp}$ and $\mathcal{L}_{\mathbb{I}}$, respectively), and a CPM loss:

$$\min_{\phi_i} \underbrace{\frac{\int_{\mathcal{M}} \|\nabla \phi_i\|^2 \, d\sigma}{\int_{\mathcal{M}} \phi_i^2 \, d\sigma}}_{\mathcal{L}_{\text{RQ}}[\phi_i]} + \underbrace{\lambda_{\perp} \frac{\sum_{j=0}^{i-1} \left(\int_{\mathcal{M}} \phi_i \bar{\phi}_j \, d\sigma \right)^2}{\int_{\mathcal{M}} \phi_i^2 \, d\sigma}}_{\mathcal{L}_{\perp}[\phi_i, \{\bar{\phi}_j\}_j]} + \underbrace{\lambda_{\mathbb{I}} \left[\log \left(\int_{\mathcal{M}} \phi_i^2 \, d\sigma \right) \right]^2}_{\mathcal{L}_{\mathbb{I}}[\phi_i]} + \lambda_{\text{cp}} \mathcal{L}_{\text{cp}}[\phi_i], \quad (11)$$

where $\bar{\phi}_j$ are previously updated and frozen eigenfunctions, and $\lambda_{\perp}$, $\lambda_{\mathbb{I}}$, λ_{cp} are penalty weights. Following Chang et al. [2025a], we

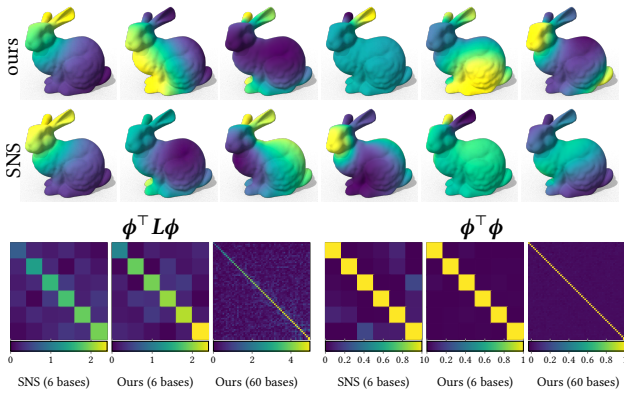


Fig. 6. **Scalar Laplacian eigenfunctions.** SNS [Williamson and Mitra 2025] takes about 6 hours for the input spherical embedding and 12 hours to compute 6 eigenfunctions; our method completes the same task in only 30 minutes. Visualizing the reduced matrices $\phi^T L \phi$, and $\phi^T \phi$, where L is the FEM-based Laplacian matrix and ϕ are values sampled at mesh vertices, shows that our results achieve better Laplacian- and L_2 -orthogonality.

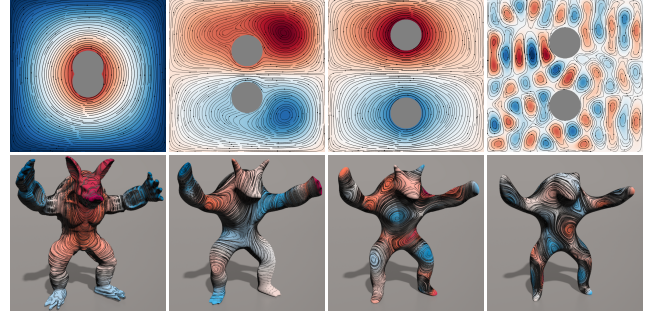


Fig. 7. **Divergence-free vector Laplacian eigenfunctions.** We visualize different-frequency eigenfunctions of the stream function with obstacles (top) and 3D closed surfaces (bottom). The streamlines of the corresponding divergence-free vector fields are also displayed for comparison purposes.

project ϕ_i out of the subspace spanned by $\{\bar{\phi}_j\}_{j=0}^{i-1}$ before gradient computation to enforce orthogonality constraints more strictly. Once updated, ϕ_i is then cached at quadrature points and normalized as $\bar{\phi}_i$ for the next eigenfunction. The first eigenfunction, ϕ_0 , is fixed as an all-ones vector. Causal sorting [Chang et al. 2025a] is also adopted to ensure mode consistency across shape space. Training details and pseudocode are provided in the supplemental material.

In Fig. 6, we compare our method with Spherical Neural Surfaces (SNS) [Williamson and Mitra 2025]. Although SNS is designed to provide a differentiable shape representation while ours focuses on encoding surface fields, both are applicable to eigenfunction computation. Technically, SNS first overfits a given spherical embedding and then solves for eigenfunctions one by one. In contrast, we optimize several eigenfunctions together, delivering higher accuracy in a fraction of the time — a factor around 30 $\times$ on the bunny example.

Divergence-Free Vector Laplacian Eigenfunctions. Divergence-free eigenfunctions are essential for reduced fluid simulations on surfaces as they can be used to define a smooth reduced basis of incompressible vector fields. In addition to mutual Laplacian- and mass-orthogonality which we discussed above, they must also satisfy the divergence-free condition, $\text{div}_{\mathcal{M}} \Phi_i = 0$. By Hodge decomposition on a surface homeomorphic to a sphere — where no non-trivial harmonic component exists — any such vector field can be completely parameterized by a single scalar stream function g , such that $\Phi = \text{rot}_{\mathcal{M}} g$. This parameterization is divergence-free by construction, and the Rayleigh quotient is then expressed as

$$\mathcal{L}_{\text{RQ}}[\Phi] = \frac{\int_{\mathcal{M}} \|\text{grad}_{\mathcal{M}} \Phi\|^2 \, d\sigma}{\int_{\mathcal{M}} \|\Phi\|^2 \, d\sigma} = \frac{\int_{\mathcal{M}} \|\text{curl}_{\mathcal{M}} \text{rot}_{\mathcal{M}} g\|^2 \, d\sigma}{\int_{\mathcal{M}} \|\text{rot}_{\mathcal{M}} g\|^2 \, d\sigma}. \quad (12)$$

Consequently, the total optimization objective for computing each eigenfunction $\Phi_i = \text{rot}_{\mathcal{M}} g_i$ is formulated as

$$\min_{g_i} \mathcal{L}_{\text{RQ}}[\Phi_i] + \lambda_{\perp} \mathcal{L}_{\perp}[\Phi_i, \{\bar{\Phi}_j\}_j] + \lambda_{\mathbb{I}} \mathcal{L}_{\mathbb{I}}[\Phi_i] + \lambda_{\text{cp}} \mathcal{L}_{\text{cp}}[g_i] + \lambda_{\mathcal{G}} \mathcal{L}_{\mathcal{G}}[g_i], \quad (13)$$

where the gauge penalty, $\mathcal{L}_{\mathcal{G}}[g] = (\int_{\mathcal{M}} g \, d\sigma)^2$ is again employed to eliminate the trivial constant kernel of the stream function. The remaining terms and the overall training procedure closely mirror the scalar Laplacian eigenfunction optimization defined in Eq. (11).

The above formulation extends naturally to simply connected, disk-like surfaces, where we need to ensure a unique solution to the

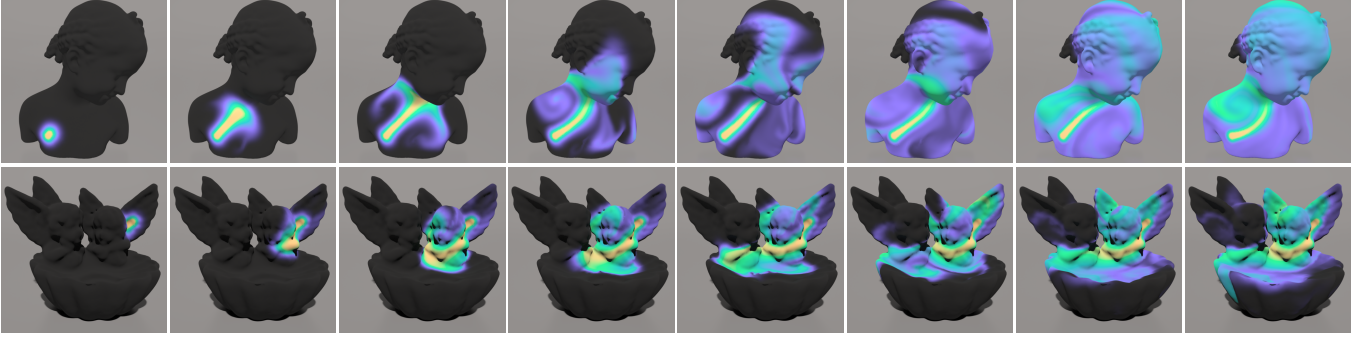


Fig. 8. **Laplacian eigenfluid simulation.** We learn forty eigenfunctions of the stream function to simulate incompressible fluids on the two sphere-like surfaces. A semi-Lagrangian time integrator [Liu et al. 2025] is applied with a time step of 5×10^{-4} seconds.

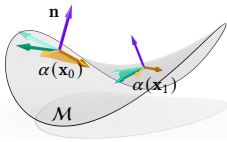
Hodge decomposition by prescribing boundary conditions. Specifically, we choose a free-slip condition for fluid simulations by incorporating the penalty term $\mathcal{L}_{bc}[g] = \int_{\partial\mathcal{M}} \|(\mathbf{n} \times \mathbf{t}) \cdot \text{rot}_{\mathcal{M}} g\|^2 ds$, where $\mathbf{t}$ is the unit tangent to the boundary curve and $\mathbf{n}$ is the surface normal. This term, weighted by a relatively large λ_{bc} , enforces the desired behaviors of the vector field at the boundary. Fig. 7 illustrates the stream eigenfunctions and corresponding streamlines of the divergence-free vector fields across various surfaces.

Laplacian Eigenfluid Simulation. Divergence-free vector Laplacian eigenfunctions offer a highly efficient basis for simulating complex fluids [De Witt et al. 2012; Liu et al. 2015; Cui et al. 2018]. Approximating the velocity with low-frequency eigenfunctions, $\mathbf{v}(\mathbf{x}, t) = \sum_i \alpha_i(t) \Phi_i(\mathbf{x})$, reduces the system to a small set of time-dependent coordinates $\{\alpha_i(t)\}_i$, significantly improving computational performance. To time-integrate this reduced system, we advect both the velocity and a scalar density field (used for visualization) via a semi-Lagrangian scheme [Liu et al. 2025]. Although the spectral variational integrator [Liu et al. 2015] can yield better energy-preserving behavior, it requires a more involved implicit scheme to remain stable at practical time step sizes. We therefore opt for the semi-Lagrangian integrator due to its simplicity and robustness. In Fig. 8, we show two examples of eigenfluid simulation on a closed surface using 40 modes each.

5.3 Generating Cross Fields on Surfaces

Rooted in the spherical harmonic (SH) representation of octahedral fields [Huang et al. 2011], feature-aligned surface cross fields can be designed by leveraging an L_p smoothness energy to reinforce crease alignment [Zhang et al. 2020]. To mimic their formulation, we train a continuous neural cross field using CPM, contrasting with the discrete representation used in NeurCross [Dong et al. 2025].

To optimize the octahedral field with a scalar-valued network, we begin by building a local reference frame $\mathbf{F}(\mathbf{x}) = [\mathbf{t}(\mathbf{x}), \mathbf{b}(\mathbf{x}), \mathbf{n}(\mathbf{x})]$ on the surface. The axis $\mathbf{n}$ is aligned with the surface normal, $\mathbf{t}$ is chosen perpendicular to $\mathbf{n}$, and $\mathbf{b} = \mathbf{n} \times \mathbf{t}$ completes the frame (see Alg. 1 in the supplement). The construction of a cross field thus reduces to optimizing a rotation angle field $\alpha: \mathcal{M} \rightarrow \mathbb{S}^1$ that rotates the tangent axes about the surface normal (see inset), and the final optimized frame $\mathbf{F}[\alpha]$ is then recovered from this angle field.



Using the SH representation, optimization combines a smoothness term, a curvature-direction alignment term, and a CPM loss:

$$\min_{\alpha} \int_{\mathcal{M}} \|\text{grad}_{\mathcal{M}} \mathbf{q}\|^2 d\sigma + \lambda_{\text{align}} \mathcal{L}_{\text{align}}[\mathbf{F}] + \lambda_{\text{cp}} \mathcal{L}_{\text{cp}}[\alpha], \quad (14)$$

where $\mathbf{q}(\mathbf{F}(\alpha)) \in \mathbb{R}^9$ is the vector field of SH coefficients at the fourth band, following [Huang et al. 2011; Zhang et al. 2020]. For the Dirichlet energy term, although automatic differentiation can compute Cartesian gradients as in other applications, we found the resulting optimization to be numerically unstable, likely due to the inherently underdetermined mapping from SH coefficients to octahedral frames [Palmer et al. 2020]. We instead approximate the Dirichlet energy through discrete covariant derivatives:

$$\int_{\mathcal{M}} \|\text{grad}_{\mathcal{M}} \mathbf{q}\|^2 d\sigma \approx \frac{\text{area}(\mathcal{M})}{N} \sum_i \frac{1}{|\mathcal{NB}(i)|} \sum_{j \in \mathcal{NB}(i)} \|\mathbf{q}(\mathbf{F}_i) - \mathbf{q}(\mathbf{R}_{ij}\mathbf{F}_j)\|^2. \quad (15)$$

The samples are randomly drawn on each facet using barycentric coordinates. For each sample i on a facet, its neighborhood $\mathcal{NB}(i)$ consists of samples from the facet’s one-ring neighbors. Here $\mathbf{R}_{ij}$ is the minimal rotation satisfying $\mathbf{R}_{ij}\mathbf{n}_j = \mathbf{n}_i$. While CPM also permits approximating the Dirichlet energy via a direct graph Laplacian over embedding-space samples, we found this to yield suboptimal results: for a *normal-aligned* octahedral field, the Dirichlet energy density of the SH field, $\|\nabla \mathbf{q}(\mathbf{F})\|^2$, decomposes into principal-curvature terms and a tangential twist term [Zhang et al. 2020]. If the frames were kept exactly normal-aligned, only the twist term would contribute to the optimization. However, because the normal-aligned property is not preserved by the gradient flow, descent directions can tilt the underlying frames off the surface normal; the curvature terms then cease to be constant and contribute gradients of their own. Consequently, the optimization sacrifices tangent-frame smoothness

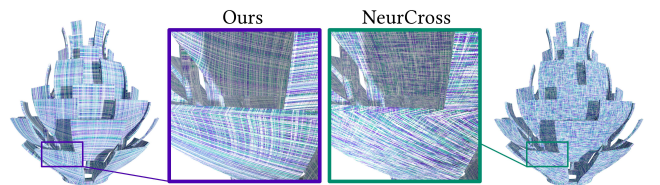


Fig. 9. **Comparisons with NeurCross near features.** Our approach enhances curvature alignment by reweighting the alignment energy based on principal curvature difference [Zhuang et al. 2014], typically leading to better feature preservation compared to NeurCross.

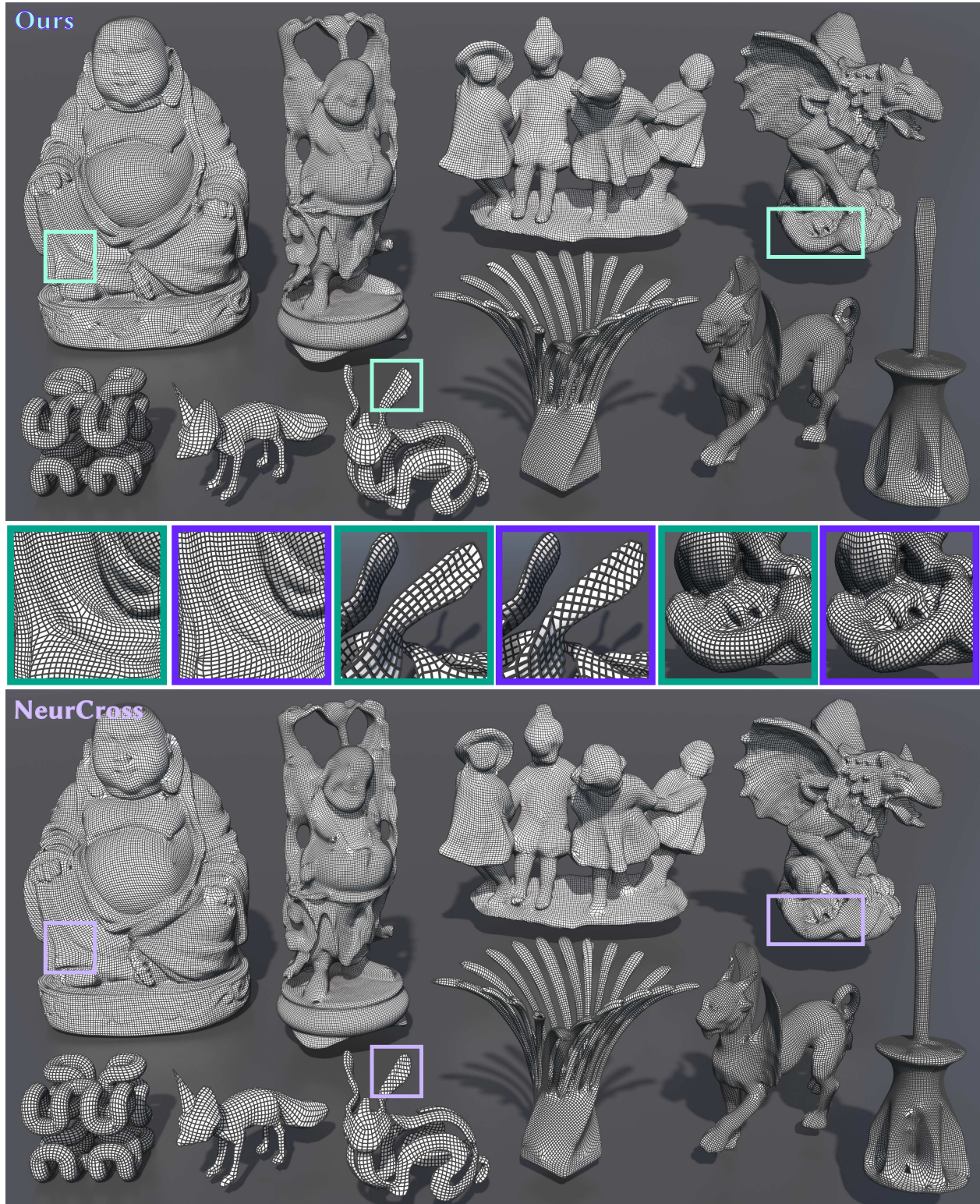


Fig. 10. **Comparison with NeurCross** [Dong et al. 2025]. On these ten challenging models, our method outperforms NeurCross on average across five key metrics (area distortion, angle distortion, singularity count, Chamfer distance, and Jacobian ratio), while using only 1.5% of its parameter count and running 5.3× faster. Detailed per-model statistics (Table S1) and streamline visualization of the cross fields (Figure S5) are provided in the supplemental material.

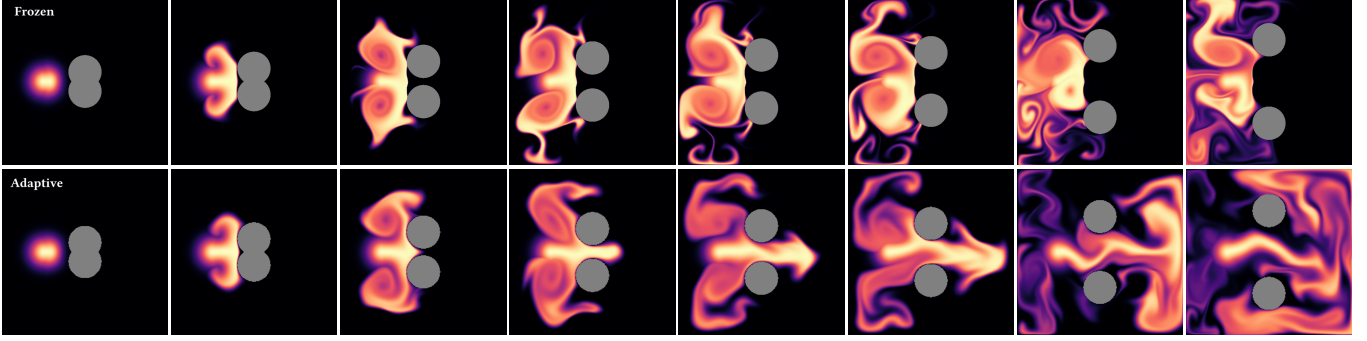


Fig. 11. **Laplacian eigenfluids generalized to moving obstacles.** Here, the control parameter defines the distance between the centers of two circular obstacles. After training, freezing the eigenfunctions to the initial obstacle configuration fails to correctly advect the velocity and density fields as obstacles separate (top); dynamically updating them at each time step resolves the issue (bottom). Eighty eigenfunctions are used in this example.

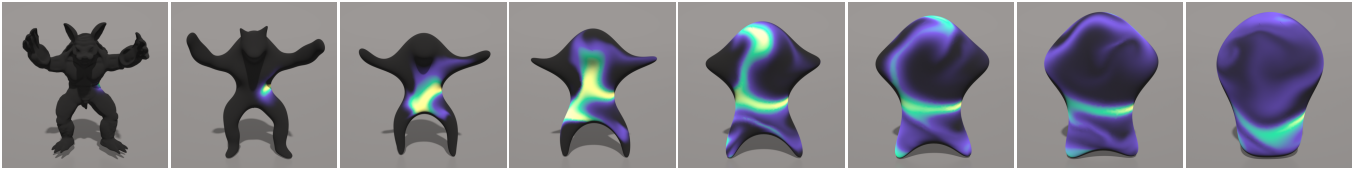


Fig. 12. **Laplacian eigenfluids generalized to deforming surfaces.** Here, the control parameter is the time during the surface’s mean-curvature flow, which we compute via the released code from Kazhdan et al. [2012]. As the surface deforms, our network generates continuously adapting eigenfunctions to ensure a temporally coherent eigenfluid simulation. This example uses sixty eigenfunctions only.

in order to suppress spurious variations in the normal component (see supplemental material, Sec. C).

In Eq. (14), the curvature direction alignment energy is written

$$\mathcal{L}_{\text{align}}[\mathbf{F}] = \int_{\mathcal{M}} \|\kappa_1\| - \|\kappa_2\| \|\nabla_{\mathbf{r}} \mathcal{P}_{\mathbf{F}(\mathbf{x})}(\mathbf{d}) - 4\mathbf{d}\|^2 d\sigma, \quad (16)$$

where $\mathbf{d}$ denotes the principal curvature direction, derived from the Hessian of the SDF. The weighting term, based on the difference between the two principal curvatures (κ_1, κ_2) adopts the idea of Zhuang et al. [2014] to strengthen the alignment in regions of strongly anisotropic curvatures. The polynomial $\mathcal{P}_{\mathbf{F}(\mathbf{x})}(\mathbf{r})$ takes a vector $\mathbf{r} \in \mathbb{R}^3$ as input and is conditioned on a frame $\mathbf{F}(\mathbf{x})$. Using the Einstein summation convention, it is expressed as $\mathcal{P}_{\mathbf{F}}(\mathbf{r}) = T_{ijkl}(\mathbf{F})r_i r_j r_k r_l$ over indices $i, j, k, l \in \{1, 2, 3\}$. Here, T_{ijkl} is an orthogonally decomposable (ODECO) tensor [Zhang et al. 2020] constructed from the frame: $T_{ijkl}(\mathbf{F}) = t_i t_j t_k t_l + b_i b_j b_k b_l + n_i n_j n_k n_l$. The rationale for this formulation follows from the result of Robeva [2016], which established that a unit vector $\mathbf{r}$ is an eigenvector of $\mathbf{T}(\mathbf{F})$ if and only if it satisfies $\nabla_{\mathbf{r}} \mathcal{P}_{\mathbf{F}}(\mathbf{r}) = 4\mathbf{r}$. Consequently, minimizing $\mathcal{L}_{\text{align}}$ encourages one of the frame axes to align with the principal curvature directions of the surface.

When the input surface contains sharp features, we additionally sample quadrature points along those feature curves and include their contribution to the alignment energy. Once the surface cross field has been obtained, we compute a global parameterization based on face samples of the cross field and the input mesh using libigl [Jacobson et al. 2018], and extract the final quadrilateral mesh with libQEx [Ebke et al. 2013]. Fig. 10 compares our method with NeurCross [Dong et al. 2025] on ten challenging models, and our approach achieves superior performance on average across five key



Fig. 13. **Smooth vector fields across anchor orientations.** The learned vector fields are parameterized by rotation angles of the constraint vectors. Network queries produce smoothly varying surface vector fields as the anchor vectors rotate (about 0.3s per query), which can be used for the rapid design of fur structures as shown here.

metrics with far fewer network parameters. We refer readers to the supplemental material for detailed per-model statistics.

5.4 Generalization

Our neural formulation naturally supports conditional surface-field learning. Given a vector $\mathbf{c} = [c_1, \dots, c_n] \in C$ representing n control parameters, we redefine the neural surface field as $f_{\theta}: \mathcal{N}(\mathcal{M}) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$, where C is the admissible set of the control parameters. Rather than training an individual network for each problem instance, we train a single model that covers C . During each training epoch, a new sample of $\mathbf{c}$ is concatenated with the point coordinates $\mathbf{x}$ and its normal $\mathbf{n}$. The network is then trained to learn a family of solutions continuously, enabling smooth interpolation between varied configurations. Consequently, the costly recomputation typical of traditional methods is streamlined by network querying.

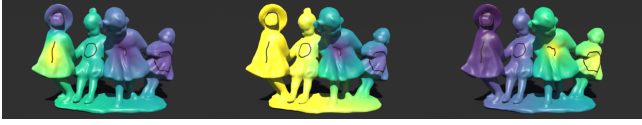


Fig. 14. **Screened Poisson solutions across source values.** The trained network can instantly generate the solution to the screened Poisson equations under varying source values.

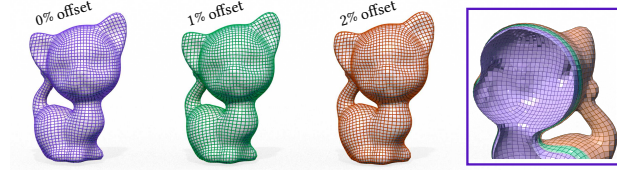


Fig. 15. **Cross fields across offset surfaces.** When taking the control parameter c as the percentage of the offset distance relative to the bounding box, the network can produce a coherent frame field for a reasonable intermediate instance, from which we extract a quadrilateral mesh.

Boundary Conditions. The control vector c can parameterize constraint values for surface field design. For instance, in vector field design, the vector c denotes the rotation angles prescribed for the anchor vectors. This allows the network’s output field to vary smoothly as anchor vectors rotate, enabling rapid synthesis of surface patterns like animal fur (see Fig. 13). For the screened Poisson problem, the vector c represents values of the sources, allowing the network to generate a series of diffusion fields under varying source data for shading the surface (Fig. 14).

Shape Spaces. We also use c to encode variations in the underlying geometry. In this setting, the network represents a conditional surface field over a family of related geometries rather than a single fixed surface. As the geometry varies with c , the learned fields change continuously, which enables amortized learning of geometry-dependent surface fields across a shape space. We demonstrate this capability across several applications. For evolving surfaces in a mean-curvature flow (MCF) sequence (Fig. 12), we learn adaptive, divergence-free vector Laplacian eigenfunctions to drive the reduced eigenfluid simulation with dynamically updated bases. On a planar surface with moving obstacles (Fig. 11), our method learns obstacle-aware bases for efficient online updates, ensuring that density fields are properly advected through the obstacles. Finally, by interpreting c as an offset distance from an input surface, we extend cross-field generation and the resulting quad-meshing to several offset surfaces (Fig. 15).

Quantitative analysis. Fig. 16 provides a quantitative evaluation of generalization performance. Measured against the FEM references, our neural solver achieves average relative L_2 errors of 2.239% on the training samples, 2.256% on the interpolation samples, and 3.358% on the extrapolation samples. The interpolation error is nearly indistinguishable from the training error, indicating that interpolation-based generalization preserves solution accuracy while eliminating the cost of rerunning the optimization. Under extrapolation, the error grows only moderately, reflecting a reasonable degree of robustness beyond the training regime.

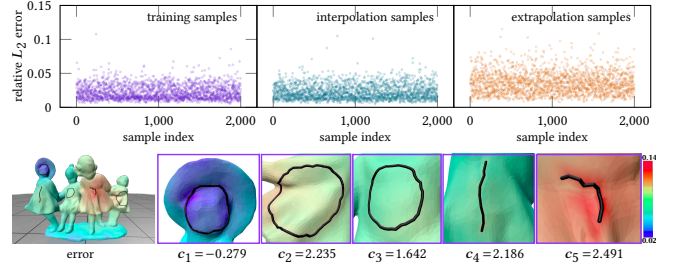


Fig. 16. **Generalization analysis.** The 5D control vector of Fig. 14 encodes the values of the five curve sources, confined to $[-3, 2]^5$ during training. The top row reports the relative L_2 error of our neural solution against an FEM reference on three sample sets: randomly retrieved training samples, interpolation samples newly drawn from $[-3, 2]^5$, and extrapolation samples from $[-3.5, 2.5]^5 \setminus [-3, 2]^5$. Interpolation essentially reproduces the training error distribution, whereas extrapolation degrades mildly, as expected. The bottom row visualizes the pointwise error $|u_i^{\text{ours}} - u_i^{\text{fem}}| / \|u^{\text{fem}}\|_\infty$ for the worst extrapolation sample, with close-ups showing it concentrates near the sources that fall furthest outside the training range.

6 Conclusion and Future Work

We presented a simple CPM-based framework for processing neural surface fields. Our approach leverages lightweight networks to achieve stronger performance than neural baselines without requiring specialized surface discretizations or complex parameterizations. By operating in an embedding space, our method supports robust geometry processing and physical simulation on complex surfaces, seamlessly applicable to diverse geometries and tasks.

Our method motivates avenues for future research. First, lifting the input to the normal bundle does not completely resolve neural networks’ spectral bias. While the lifting is effective for close planes with opposite normals, leakage still occurs between close faces sharing the same normal (see inset). Second, our vector field formulation is currently limited to genus-zero surfaces, where the harmonic components simply vanish. Extending it to higher-genus surfaces would require precomputing harmonic basis vectors and tracking their contributions explicitly, e.g., time-stepping harmonic coefficients in fluid simulation [Yin et al. 2023]. Third, we currently extend surface geometry using a narrow-band SDF. We also experimented with a neural SDF but found it impractical due to slow setup and Jacobian computation as well as a large memory footprint. Finally, shape co-optimization for inverse design would also be a valuable direction to explore.

Acknowledgments

We thank anonymous reviewers for their constructive feedback, and Wei Li for generously sharing his GPU resources for this project. XN also acknowledges the CGVU Lab led by Prof. Taku Komura at the University of Hong Kong for additional computing resources. MD acknowledges the support of the MediTwin consortium (funded by the French government as part of France 3030), of an Inria Chair, and funding from Synopsys and Adobe Research.

References

- Alexander G. Belyaev and Pierre-Alain Fayolle. 2015. On Variational and PDE-Based Distance Function Approximations. *Computer Graphics Forum* 34, 8 (2015), 104–118. doi:10.1111/cgf.12611
- Camille Buonomo, Julie Digne 1, and Raphaëlle Chaine. 2025. Volume Preserving Neural Shape Morphing. *Computer Graphics Forum* 44, 5 (2025), e70196. doi:10.1111/cgf.70196
- Yue Chang, Otman Benckroun, Maurizio M. Chiaramonte, Peter Yichen Chen, and Eitan Grinspun. 2025a. Shape Space Spectra. *ACM Trans. Graph.* 44, 4, Article 121 (2025). doi:10.1145/3731148
- Yue Chang, Peter Yichen Chen, Zhecheng Wang, Maurizio M. Chiaramonte, Kevin Carlberg, and Eitan Grinspun. 2023. LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields. In *SIGGRAPH Asia 2023 Conference Papers*. Article 111. doi:10.1145/3610548.3618158
- Yue Chang, Mengfei Liu, Zhecheng Wang, Peter Yichen Chen, and Eitan Grinspun. 2025b. Lifting the Winding Number: Precise Discontinuities in Neural Fields for Physics Simulation. In *Proceedings of ACM SIGGRAPH Conference*. Article 25. doi:10.1145/3721238.3730597
- Honglin Chen, Rundui Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2023. Implicit Neural Spatial Representations for Time-dependent PDEs. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*. 5162–5177.
- Ming Chuang and Michael Kazhdan. 2011. Interactive and anisotropic geometry processing using the screened Poisson equation. In *ACM SIGGRAPH 2011 papers (SIGGRAPH '11)*. 1–10. doi:10.1145/1964921.1964952
- Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. 2013. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*. Article 7. doi:10.1145/2504435.2504442
- Qiaodong Cui, Pradeep Sen, and Theodore Kim. 2018. Scalable laplacian eigenfluids. *ACM Trans. Graph.* 37, 4 (2018). doi:10.1145/3197517.3201352
- Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using Laplacian eigenfunctions. *ACM Trans. Graph.* 31, 1 (2012). doi:10.1145/2077341.2077351
- Yitong Deng, Hong-Xing Yu, Diyang Zhang, Jiajun Wu, and Bo Zhu. 2023. Fluid Simulation on Neural Flow Maps. *ACM Trans. Graph.* 42, 6, Article 248 (2023). doi:10.1145/3618392
- Mathieu Desbrun, Anil N. Hirani, Melvin Leok, and Jerrold E. Marsden. 2005. Discrete Exterior Calculus. arXiv:math/0508341 [math.DG] <https://arxiv.org/abs/math/0508341>
- Ana Dodik, Oded Stein, Vincent Sitzmann, and Justin Solomon. 2023. Variational Barycentric Coordinates. *ACM Trans. Graph.* 42, 6, Article 255 (2023). doi:10.1145/3618403
- Qiujie Dong, Huibiao Wen, Rui Xu, Shuangmin Chen, Jiaran Zhou, Shiqing Xin, Changhe Tu, Taku Komura, and Wenping Wang. 2025. NeurCross: A Neural Approach to Computing Cross Fields for Quad Mesh Generation. *ACM Trans. Graph.* 44, 4, Article 108 (2025). doi:10.1145/3731159
- Hans-Christian Ebke, David Bommes, Marcel Campen, and Leif Kobbelt. 2013. QEx: Robust Quad Mesh Extraction. *ACM Trans. Graph.* 32, 6, Article 168 (2013). doi:10.1145/2508363.2508372
- Jin Huang, Yiyang Tong, Hongyu Wei, and Hujun Bao. 2011. Boundary aligned smooth 3D cross-frame field. *ACM Trans. Graph.* 30, 6 (2011). doi:10.1145/2070781.2024177
- Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. <https://libigl.github.io/>.
- Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can Mean-Curvature Flow be Modified to be Non-singular? *Computer Graphics Forum* 31, 5 (2012), 1745–1754. doi:10.1111/j.1467-8659.2012.03179.x
- Theodore Kim, Jerry Tessendorf, and Nils Thürey. 2013. Closest point turbulence for liquid surfaces. *ACM Trans. Graph.* 32, 2, Article 15 (2013). doi:10.1145/2451236.2451241
- Nathan King, Haozhe Su, Mridul Aanjaneya, Steven Ruuth, and Christopher Batty. 2024. A Closest Point Method for PDEs on Manifolds with Interior Boundary Conditions for Geometry Processing. *ACM Trans. Graph.* 43, 5, Article 159 (2024). doi:10.1145/3673652
- Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] <https://arxiv.org/abs/1412.6980>
- Mica Li, Michael Owens, Juheng Wu, Grace Yang, and Albert Chern. 2023b. Closest Point Exterior Calculus. In *SIGGRAPH Asia 2023 Posters*. Article 9. doi:10.1145/3610542.3626143
- Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhat-tacharya, Andrew Stuart, and Anima Anandkumar. 2021. Fourier Neural Operator for Parametric Partial Differential Equations. arXiv:2010.08895 [cs.LG] <https://arxiv.org/abs/2010.08895>
- Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023a. Neuralangelo: High-fidelity neural surface reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 8456–8465.
- Beibei Liu, Gemma Mason, Julian Hodgson, Yiyang Tong, and Mathieu Desbrun. 2015. Model-reduced variational fluid simulation. *ACM Trans. Graph.* 34, 6 (2015). doi:10.1145/2816795.2818130
- Yibo Liu, Zhixin Fang, Sune Darkner, Noam Aigerman, Kenny Erleben, Paul Kry, and Tesse Schneider. 2025. Neural Kinematic Bases for Fluids. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. Article 28. doi:10.1145/3757377.3763925
- Colin B. MacDonald and Steven J. Ruuth. 2010. The Implicit Closest Point Method for the Numerical Solution of Partial Differential Equations on Surfaces. *SIAM Journal on Scientific Computing* 31, 6 (2010), 4330–4350. doi:10.1137/08074003
- Thomas März and Colin B. Macdonald. 2012. Calculus on Surfaces with General Closest Point Functions. *SIAM J. Numer. Anal.* 50, 6 (2012), 3303–3328. doi:10.1137/120865537
- Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A Level Set Theory for Neural Implicit Evolution Under Explicit Flows. In *Computer Vision – ECCV 2022*. 711–729.
- 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*. 4460–4470.
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106. doi:10.1145/3503250
- Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* 41, 4, Article 102 (2022). doi:10.1145/3528223.3530127
- David Palmer, David Bommes, and Justin Solomon. 2020. Algebraic Representations for Volumetric Frame Fields. *ACM Trans. Graph.* 39, 2 (2020). doi:10.1145/3366786
- 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*. 165–174.
- Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. 2021. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*. 5865–5874.
- Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. 2019. On the Spectral Bias of Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. 5301–5310.
- M. Raissi, P. Perdikaris, and G.E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707. doi:10.1016/j.jcp.2018.10.045
- Elina Robeva. 2016. Orthogonal Decomposition of Symmetric Tensors. *SIAM J. Matrix Anal. Appl.* 37, 1 (2016), 86–102. doi:10.1137/140989340
- Steven J Ruuth and Barry Merriman. 2008. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.* 227, 3 (2008), 1943–1961. doi:10.1016/j.jcp.2007.10.009
- Nicholas Sharp and Keenan Crane. 2020. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum* 39, 5 (2020), 69–80. doi:10.1111/cgf.14069
- Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems*, Vol. 33. 7462–7473.
- Oded Stein, Max Wardetzky, Alec Jacobson, and Eitan Grinspun. 2020. A Simple Discretization of the Vector Dirichlet Energy. *Computer Graphics Forum* 39, 5 (2020), 81–92. doi:10.1111/cgf.14070
- Ryusuke Sugimoto, Nathan King, Toshiya Hachisuka, and Christopher Batty. 2024. Projected Walk on Spheres: A Monte Carlo Closest Point Method for Surface PDEs. In *SIGGRAPH Asia 2024 Conference Papers*. Article 8. doi:10.1145/3680528.3687599
- Kenji Tojo and Nobuyuki Umetani. 2025. GreenCloud: Volumetric Gradient Filtering via Regularized Green's Functions. *Computer Graphics Forum* 44, 5 (2025), e70207. doi:10.1111/cgf.70207
- Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* 35, 2 (2016), 545–572. doi:10.1111/cgf.12864
- Haoliang Wang, Tao Yu, Hui Qiao, and Qionghai Dai. 2025. Neural Fluid Simulation on Geometric Surfaces. In *International Conference on Learning Representations*, Vol. 2025. 99091–99118.
- Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems*, Vol. 34. 27171–27183.
- Lilian Welschinger, Yilin Liu, Zican Wang, and Niloy Mitra. 2026. Learning to Solve PDEs on Neural Shape Representations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- Francis Williams, Jiahui Huang, Jonathan Swartz, Gergely Klar, Vijay Thakkar, Matthew Cong, Xuanchi Ren, Ruilong Li, Clement Fuji-Tsang, Sanja Fidler, Eftychios Sfakias,

- and Ken Museth. 2024. FVDB: A Deep-Learning Framework for Sparse, Large Scale, and High Performance Spatial Intelligence. *ACM Trans. Graph.* 43, 4, Article 133 (2024). doi:10.1145/3658226
- Romy Williamson and Niloy J. Mitra. 2025. Neural Geometry Processing via Spherical Neural Surfaces. *Computer Graphics Forum* 44, 2 (2025), e70021. doi:10.1111/cgf.70021
- 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.
- Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume Rendering of Neural Implicit Surfaces. In *Advances in Neural Information Processing Systems*, Vol. 34. 4805–4815.
- Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. 2020. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. In *Advances in Neural Information Processing Systems*, Vol. 33. 2492–2502.
- Hang Yin, Mohammad Sina Nabizadeh, Baichuan Wu, Stephanie Wang, and Albert Chern. 2023. Fluid Cohomology. *ACM Trans. Graph.* 42, 4, Article 126 (2023). doi:10.1145/3592402
- Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. 2021. NTopo: Mesh-free Topology Optimization using Implicit Neural Representations. In *Advances in Neural Information Processing Systems*, Vol. 34. 10368–10381.
- Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. 2020. Octahedral Frames for Feature-Aligned Cross Fields. *ACM Trans. Graph.* 39, 3, Article 25 (2020). doi:10.1145/3374209
- Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. 2014. Anisotropic geodesics for live-wire mesh segmentation. *Comput. Graph. Forum* 33, 7 (2014), 111–120. doi:10.1111/cgf.12479

Neural Field Processing on Surfaces — *Supplementary Material*

WEIZHOU LIU, Inria / École polytechnique, France

XINGYU NI, Inria, France

MATHIEU DESBRUN, Inria / École polytechnique, France

JIONG CHEN, Inria, France

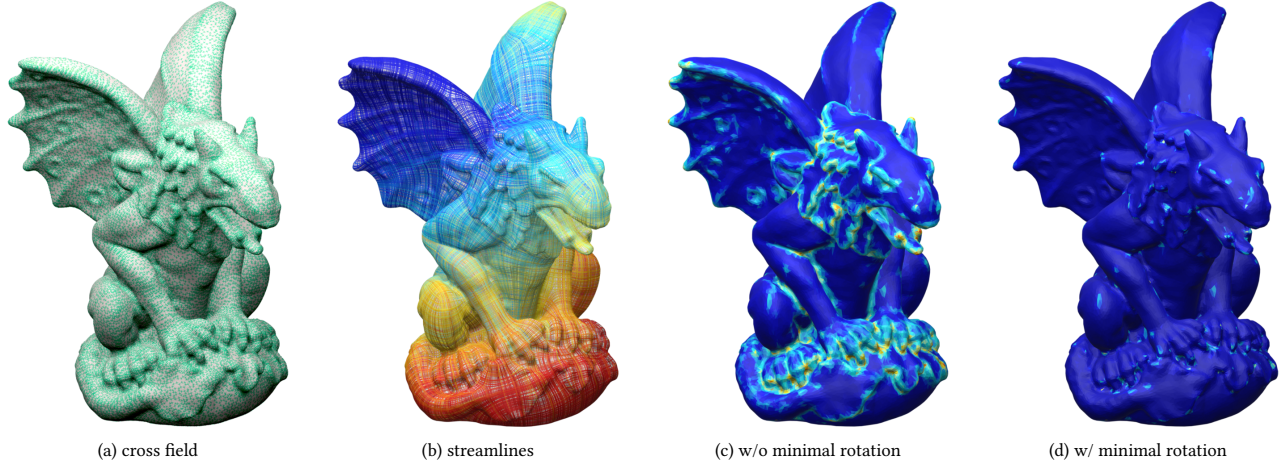


Fig. S1. **Effect of the normal minimal rotation.** Given a high-quality cross field (a) and its streamlines (b), the Dirichlet energy computed without normal minimal rotation remains large in curved regions (c), causing the optimization to keep penalizing high-curvature areas in practice. With normal minimal rotation (d), the Dirichlet energy removes the influence of surface curvature.

A Surface Differential Operators via Differential Forms

While we define manifold differential operators via their Cartesian counterparts through Eq. (2) in the main paper, they can also be defined using differential forms, independent of coordinate frame choices.

Let $(\mathcal{M}, g)$ be an oriented two-dimensional Riemannian manifold. We denote by $C^\infty(\mathcal{M})$ the space of smooth scalar fields and by $\mathfrak{X}(\mathcal{M})$ the space of smooth vector fields on $\mathcal{M}$. Let $\Omega^k(\mathcal{M})$ be the space of smooth differential k -forms. The metric g induces two musical isomorphisms

$$\flat : \mathfrak{X}(\mathcal{M}) \rightarrow \Omega^1(\mathcal{M}), \quad \sharp : \Omega^1(\mathcal{M}) \rightarrow \mathfrak{X}(\mathcal{M}).$$

Moreover, we write d for the exterior derivative and $\star$ for the Hodge star operator associated with the Riemannian metric g and the chosen orientation.

We define the basic first-order differential operators intrinsically as follows. For a smooth function $f \in C^\infty(\mathcal{M})$, its gradient is

$$\text{grad}_{\mathcal{M}} f := (df)^\sharp,$$

while the vector field obtained by rotating its gradient by $\pi/2$ is defined by

$$\text{rot}_{\mathcal{M}} f := (\star df)^\sharp.$$

For a smooth vector field $X \in \mathfrak{X}(\mathcal{M})$, its scalar curl is defined by

$$\text{curl}_{\mathcal{M}} X := \star d(X^\flat),$$

and its divergence is defined by

$$\text{div}_{\mathcal{M}} X := \star d \star (X^\flat).$$

These operators are entirely determined by the exterior derivative d and the Hodge star $\star$, and hence depend only on the intrinsic geometry of $(\mathcal{M}, g)$.

Equivalently, the above operators can be summarized as

$$\begin{aligned} \text{grad}_{\mathcal{M}} &= \sharp \circ d, \\ \text{rot}_{\mathcal{M}} &= \sharp \circ \star \circ d, \\ \text{curl}_{\mathcal{M}} &= \star \circ d \circ \flat, \\ \text{div}_{\mathcal{M}} &= \star \circ d \circ \star \circ \flat. \end{aligned}$$

Since $d^2 = 0$ and $\star^2 = -\text{Id}$ on 1-forms, these intrinsic definitions satisfy classical vector calculus identities, such as

$$\text{curl}_{\mathcal{M}}(\text{grad}_{\mathcal{M}} f) = 0, \quad \text{div}_{\mathcal{M}}(\text{rot}_{\mathcal{M}} f) = 0,$$

for all $f \in C^\infty(\mathcal{M})$.

B Hodge Decomposition of the Vector Dirichlet Energy

Let $\mathbf{u}$ be a tangential vector field on a closed oriented surface $\mathcal{M}$, and let $\alpha = \mathbf{u}^\flat$ be its corresponding one-form. By the Hodge decomposition, α splits uniquely into exact, co-exact, and harmonic components. We define the vector Dirichlet energy used in this work as the quadratic form of the *Hodge Laplacian* $\Delta_H = d\delta + \delta d$,

$$\mathcal{E}_{\mathcal{D}}[\mathbf{u}] := \langle \alpha, \Delta_H \alpha \rangle_{\mathcal{M}}.$$

Table S1. **Comparisons between our method and NeurCross.** We report multiple evaluation metrics for both methods on various models, together with their average values. The angle distortion is calculated in degrees. Better results are highlighted in bold. We also report the training time (s). For our method, the reported time includes both sparse-grid SDF construction (first term) and network training (second term), while for NeurCross it includes only network training. Both methods are trained for 10K epochs. Our method has an average of 67,073 learnable parameters, whereas NeurCross has an average of 4,374,082 learnable parameters.

Model (#faces)	NeurCross						Ours					
	Area ↓	Angle(°) ↓	#Sings ↓	CD ↓	JR ↑	time (s) ↓	Area ↓	Angle(°) ↓	#Sings ↓	CD ↓	JR ↑	time (s) ↓
BUNNYPEEL (15K)	1.09	11.87	125	238.60	0.88	1872.13	1.46	13.21	141	241.63	0.85	26.25 + 811.20
FENNECFox (12K)	1.12	12.64	88	185.18	0.86	1633.01	1.22	12.73	77	185.69	0.86	14.30 + 788.24
HILB64THICK (56K)	1.92	11.65	242	262.28	0.87	6650.34	1.20	8.88	256	254.52	0.89	76.71 + 886.29
VASE (36K)	0.27	12.11	170	134.94	0.94	4341.47	0.25	13.01	130	134.21	0.94	16.90 + 854.79
BUDDHA (50K)	0.11	12.12	158	108.88	0.95	5900.07	0.09	12.75	147	108.65	0.94	23.72 + 884.17
DC (45K)	0.15	13.79	239	107.30	0.92	5307.17	0.12	11.12	175	106.53	0.94	23.77 + 884.57
FELINE (20K)	0.21	13.85	160	123.42	0.92	2417.27	0.18	12.70	138	123.82	0.93	14.72 + 814.10
GARGOYLE (60K)	0.21	15.86	346	115.67	0.90	7146.95	0.17	13.68	244	115.15	0.92	22.15 + 929.42
HAPPYBUDDHA (50K)	0.06	13.04	285	75.04	0.93	5890.55	0.05	11.02	201	74.27	0.95	11.11 + 902.18
SCREWDRIVER (54K)	0.12	11.27	71	65.39	0.94	6453.19	0.13	7.40	24	65.86	0.96	6.3 + 892.04
Average	0.53	12.82	188.40	141.67	0.91	4761.22	0.49	11.65	153.30	141.03	0.92	23.60 + 864.70

Since $\mathcal{M}$ is closed, integration by parts produces no boundary terms, and

$$\langle \alpha, \Delta_H \alpha \rangle_{\mathcal{M}} = \langle \alpha, (d\delta + \delta d)\alpha \rangle_{\mathcal{M}} = \|\delta\alpha\|_{\mathcal{M}}^2 + \|d\alpha\|_{\mathcal{M}}^2.$$

For a one-form on a two-dimensional surface, $\delta\alpha = -\text{div}_{\mathcal{M}}\mathbf{u}$ and $\star d\alpha = \text{curl}_{\mathcal{M}}\mathbf{u}$; as the Hodge star preserves the norm, $\|d\alpha\| = \|\star d\alpha\|$. The energy therefore decomposes as

$$\mathcal{E}_{\mathcal{D}}[\mathbf{u}] = \int_{\mathcal{M}} \left(\|\text{div}_{\mathcal{M}}\mathbf{u}\|^2 + \|\text{curl}_{\mathcal{M}}\mathbf{u}\|^2 \right) d\sigma, \quad (\text{S1})$$

which is exactly Eq. (9) of the main paper once $\mathbf{u}$ is written through the potentials $\mathbf{u} = \text{grad}_{\mathcal{M}}f + \text{rot}_{\mathcal{M}}g$. For the harmonic component h one has $dh = 0$ and $\delta h = 0$, so it does not contribute to the energy; on the genus-zero surfaces we consider, no non-trivial harmonic component exists in the first place.

C Tangential-Twist-Only Smoothness Energy

Zhang et al. [2020] showed that for a *normal-aligned* octahedral field $\mathbf{F}$ with the SH representation $\mathbf{q}$, the Dirichlet energy density decomposes pointwise as

$$\|\nabla \mathbf{q}(\mathbf{F})\|^2 = \kappa_1^2 + \kappa_2^2 + w,$$

where κ_1, κ_2 are the principal curvatures and w measures the tangential twist of the frame. The curvature terms are determined entirely by the variation of the surface normal and are therefore fixed by the surface geometry; only the twist term depends on the frame. If the optimization were constrained to normal-aligned fields, the curvature terms would be constants and contribute no gradient, and minimizing the Dirichlet energy would amount to smoothing the in-plane rotation of the frames. However, the gradient flow does not preserve the normal alignment: descent directions can tilt the frames away from the surface normal, at which point the curvature terms become frame-dependent and contribute gradients as well. The optimization can then trade tangential smoothness for a reduction in normal variations, particularly in high-curvature regions.

To remedy this issue, Zhang et al. [2020] use a penalty function to strengthen normal alignment. Given our angle-based representation, we constrain the normal alignment *explicitly* in SH representation, leading to the covariant derivative based energy defined by Eq. (15) in the main paper. Our approach directly factors the normal variations out of the smoothness energy, and therefore avoids over-penalizing high-curvature regions which would otherwise hinder the minimization of tangential twist.

In Fig. S1, we show this effect on a high-quality input cross field. Without normal minimal rotation, the smoothness energy remains large in high-curvature regions. With normal minimal rotation, the curvature contribution is removed, and the energy better reflects the intrinsic twist of the frame field.

D Quantitative Comparison with NeurCross on Quadrilateral Meshing

We ran NeurCross using the authors' open-source implementation¹. Details regarding our optimization are provided in Sec. E. To ensure a fair comparison, both methods were evaluated using a fixed set of parameters, and all experiments were conducted on a single NVIDIA RTX 4090 GPU.

Evaluation metrics. We assess the quality of the extracted quadrilateral meshes using five complementary metrics: area distortion (Area), angle distortion (Angle), number of singularities (#Sings), Chamfer distance (CD), and Jacobian Ratio (JR). Area distortion evaluates the uniformity of element sizes by computing the standard deviation of quadrilateral face areas, scaled by 10^4 . Angle distortion measures the deviation of quad corners from right angles, defined as

$$\text{Angle} = \sqrt{\frac{1}{N} \sum_{i=1}^N (\beta_i - 90^\circ)^2},$$

where β_i is a corner angle (in degrees) and N is the total number of quad angles. #Sings counts the number of extraordinary vertices, i.e.,

¹<https://github.com/QiuJieDong/NeurCross>



Fig. S2. **Comparison with NeurCross on memory consumption.** For the approximately 80K-triangle model shown on the left, NeurCross runs out of memory, while our method still produces a high-quality cross field. We visualize the generated quadrilateral mesh on the right.

those whose valence differs from the regular quad-mesh valence. CD measures the geometric discrepancy between the input surface and the generated quad mesh using the bidirectional Chamfer distance with the L_1 norm, scaled by 10^4 . JR characterizes the local regularity of each quadrilateral element by comparing the smallest and largest Jacobian determinants evaluated at its corners. Lower Area, Angle, #Sings, and CD values, together with a higher JR value, indicate better quadrilateral mesh quality.

In Table S1, we quantitatively compare our method with NeurCross on five metrics across ten challenging models. Our method achieves better average performance. In particular, our approach only requires a lightweight SIREN network, whereas NeurCross uses both a SIREN and a U-Net, resulting in a larger parameter count and higher memory consumption. As a result, NeurCross can exceed the memory limit of an NVIDIA RTX 4090 GPU on high-resolution models. Fig. S2 shows such an example with approximately 80K triangles: NeurCross runs out of memory, while our method still produces high-quality results.

E Implementation Details

In this section, we elaborate on implementation details and parameter settings of our method for different applications.

E.1 Initializing Local Reference Frames for Cross-Field Learning

For learning a surface cross field parameterized by an angle field, we initialize a deterministic normal-aligned local reference frame from the surface normal. Following the notation in the main text, we denote the local frame as $F(\mathbf{x}) = [\mathbf{t}(\mathbf{x}), \mathbf{b}(\mathbf{x}), \mathbf{n}(\mathbf{x})]$. At a surface point $\mathbf{x}$ with its normal $\mathbf{n} = (n_x, n_y, n_z)$, we first select the coordinate axis least aligned with $\mathbf{n}$. We then construct the first tangent vector $\mathbf{t}$ by setting the selected component to zero and swapping/negating the remaining two components, so that $\mathbf{t} \perp \mathbf{n}$. The second tangent vector is defined as $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. After normalization, $(\mathbf{t}, \mathbf{b}, \mathbf{n})$ forms

an orthonormal local reference frame. This axis-swap construction avoids the pole singularity of Rodrigues-based alignment, with only measure-zero gauge discontinuities where two components of $|\mathbf{n}|$ are equal. The full procedure is summarized in Alg. 1.

E.2 Network Training with Different Boundary Conditions

Screened Poisson Equation. Assume there are n sources on the surface, and the prescribed Dirichlet values on these sources lie in the range $[a, b]$. We define the control vector as $\mathbf{c} = (c_1, c_2, \dots, c_n) \in [a, b]^n$, where each c_i controls the source value on the corresponding source Γ_i . The training algorithm is summarized in Alg. 2.

ALGORITHM 1: Constructing local reference frames.

Input : Surface sample points $\{\mathbf{x}_\ell\}_{\ell=1}^N$ with normals $\{\mathbf{n}_\ell\}_{\ell=1}^N$, where $\mathbf{n}_\ell \in \mathbb{R}^3$.
Output : A normal-aligned local reference frame $F_\ell = [\mathbf{t}_\ell, \mathbf{b}_\ell, \mathbf{n}_\ell]$ for each sample point $\mathbf{x}_\ell$.

```

1 for  $\ell = 1$  to  $N$  do
2   Normalize the normal:
       
$$\mathbf{n}_\ell \leftarrow \frac{\mathbf{n}_\ell}{\|\mathbf{n}_\ell\|};$$

3   Select the coordinate axis least aligned with  $\mathbf{n}_\ell$ :
       
$$i = \arg \min_{a \in \{x, y, z\}} |(\mathbf{n}_\ell)_a|;$$

4   Let  $j, k$  be the remaining two coordinate axes in ascending order;
5   Initialize  $\mathbf{t}_\ell = 0$  and set
       
$$(\mathbf{t}_\ell)_i = 0, \quad (\mathbf{t}_\ell)_j = (\mathbf{n}_\ell)_k, \quad (\mathbf{t}_\ell)_k = -(\mathbf{n}_\ell)_j;$$

6   Construct the normalized tangent vectors as follows:
       
$$\mathbf{t}_\ell \leftarrow \frac{\mathbf{t}_\ell}{\|\mathbf{t}_\ell\|}, \quad \mathbf{b}_\ell \leftarrow \frac{\mathbf{n}_\ell \times \mathbf{t}_\ell}{\|\mathbf{n}_\ell \times \mathbf{t}_\ell\|};$$

7 return  $\{F_\ell = [\mathbf{t}_\ell, \mathbf{b}_\ell, \mathbf{n}_\ell]\}_{\ell=1}^N;$ 

```

ALGORITHM 2: Training the screened Poisson solver with varied constraint values.

Input : A surface $\mathcal{M}$ represented by a triangular mesh; a set of sources $\{\Gamma_i\}_{i=1}^n$ on $\mathcal{M}$, where each Γ_i is either a point or a curve; the lower and upper bounds $[a, b]$ of the Dirichlet boundary values and the shape-control domain $C = [a, b]^n$; the number of training epochs N ; the number of surface samples N_M ; and the number of narrow-band samples N_N .

Output : A SIREN neural network $f_\theta : \mathcal{N}(\mathcal{M}) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$ that, given a surface point $\mathbf{x}$, its normal $\mathbf{n}$, and a control vector $\mathbf{c} = (c_1, c_2, \dots, c_n)$ with $c_i \in [a, b]$, outputs the screened Poisson solution corresponding to the Dirichlet values encoded by $\mathbf{c}$.

- 1 **Initialization**: Build a sparse grid around $\mathcal{M}$. At each grid vertex, evaluate the SDF value and compute derivatives of arbitrary order using finite differences. Initialize f_θ with SIREN initialization;
- 2 **for** $I = 1$ **to** N **do**
- 3 Randomly sample a control vector $\mathbf{c} \in C$;
- 4 Sample N_M points $X = \{\mathbf{x}_i\}$ on the surface and N_N points $Y = \{\mathbf{y}_i\}$ in the narrow band;
- 5 Query normals and SDF derivatives for the sampled points;
- 6 Compute the loss $\mathcal{L}(f_\theta, X, Y, \mathbf{c})$ according to Sec. 5.1, Eq. (8);
- 7 Update the network parameters by back-propagation;
- 8 **return** f_θ ;

Vector Field Design. Assume that we are given n tangential constraint vectors $\{\mathbf{u}_i\}_{i=1}^n$ on the surface. We define the control vector as $\mathbf{c} = (c_1, c_2, \dots, c_n) \in [0, 2\pi]^n$, where each c_i represents the rotation angle of the corresponding constraint vector within its tangent plane. Thus, $\mathbf{c}$ parameterizes the constraint directions. The training algorithm is summarized in Alg. 3.

E.3 Network Training within Shape Spaces

Div-Free Laplacian Modes in Shape Spaces. We treat $\mathbf{c}$ as a control parameter that indexes a family of shapes $\mathcal{M}_c$, such as the time t along a surface deformation process. This allows our method to learn Laplacian modes that vary continuously over the shape space. In the main paper, $\mathbf{c} \in \mathbb{Z}^+$ denotes the deformation step of a mean-curvature-flow sequence, and our method learns Laplacian modes along the evolving surface. For 2D domains with moving obstacles, $\mathbf{c} \in [0, 0.5]$ denotes the distance between the centers of the two circular obstacles. Our method can even capture newly emerging modes after topological changes, enabling plausible density advection. The training algorithm is summarized in Alg. 4.

Cross Fields on Offset Surfaces. By treating the parameter c as the surface offset distance, our method learns a family of frame fields that vary smoothly across offset surfaces. The training algorithm is summarized in Alg. 5.

E.4 Advection on Generalized Surfaces

After learning the divergence-free Laplacian modes on shape space

$$\Phi_j(\cdot; c) = \text{rot}_{\mathcal{M}_c} h_{\theta_j}(\cdot; c), \quad j = 1, \dots, k,$$

ALGORITHM 3: Training the potential with varying anchor orientations for vector field design.

Input : A surface $\mathcal{M}$ represented by a triangular mesh; a set of tangential constraint vectors $\{\mathbf{u}_i\}_{i=1}^n$ on $\mathcal{M}$; The shape-control domain $C = [0, 2\pi]^n$; the number of training epochs N ; the number of surface samples N_M ; and the number of narrow-band samples N_N .

Output : Two SIREN neural networks, $f_\theta : \mathcal{N}(\mathcal{M}) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$ and $g_\theta : \mathcal{N}(\mathcal{M}) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$, representing the potential function and stream function, respectively. Given a surface point $\mathbf{x}$, its normal $\mathbf{n}$, and a control vector $\mathbf{c} = (c_1, c_2, \dots, c_n)$ with $c_i \in [0, 2\pi]$, the networks produce a surface vector field that varies smoothly with the constraint directions encoded by $\mathbf{c}$.

- 1 **Initialization**: Build a sparse grid around $\mathcal{M}$. At each grid vertex, evaluate the SDF value and compute derivatives of arbitrary order using finite differences. Initialize f_θ and g_θ with SIREN initialization;
- 2 **for** $I = 1$ **to** N **do**
- 3 Randomly sample a control vector $\mathbf{c} \in C$;
- 4 Sample N_M points $X = \{\mathbf{x}_i\}$ on the surface and N_N points $Y = \{\mathbf{y}_i\}$ in the narrow band;
- 5 Query normals and SDF derivatives for the sampled points;
- 6 Compute the loss $\mathcal{L}(f_\theta, g_\theta, X, Y, \mathbf{c})$ according to Sec. 5.1, Eq. (10);
- 7 Update the network parameters by back-propagation;
- 8 **return** f_θ, g_θ ;

we use them to advect a density field on a sequence of generalized surfaces $\{\mathcal{M}_{c_m}\}_{m=0}^{T-1}$. All meshes in the sequence share the same connectivity and vertex correspondence. For each surface $\mathcal{M}_{c_m}$, we precompute the mode values $\Phi_j(\mathbf{x}_i^m; c_m)$ at mesh vertices, the lumped vertex areas A_i^m , and the Gram matrix

$$G_{pq}^m = \sum_i A_i^m \langle \Phi_p(\mathbf{x}_i^m; c_m), \Phi_q(\mathbf{x}_i^m; c_m) \rangle.$$

We initialize the density as a Gaussian blob centered at $\mathbf{x}_s \in \mathcal{M}_{c_0}$, i.e.,

$$\rho_i^0 = \exp\left(-\frac{\|\mathbf{x}_i^0 - \mathbf{x}_s\|^2}{2r^2}\right),$$

and construct a *target* tangent velocity u_{tar} of the form

$$u_{\text{tar}}(\mathbf{x}_i^0) = s \exp\left(-\frac{\|\mathbf{x}_i^0 - \mathbf{x}_s\|^2}{2r^2}\right) P_{T_{\mathbf{x}_i^0} \mathcal{M}_{c_0}} d,$$

where $d \in \mathbb{R}^3$ is the prescribed direction, which is first projected onto the tangent plane at the source point and normalized; $P_{T_{\mathbf{x}_i^0} \mathcal{M}_{c_0}} = I - n_i^0(n_i^0)^T$ denotes the orthogonal projection onto the tangent plane at $\mathbf{x}_i^0$, and $s > 0$ is a scalar controlling the speed of the injected flow. The initial coefficient vector is obtained by the weighted least-squares projection

$$\alpha^0 = \arg \min_{\alpha} \sum_i A_i^0 \left\| \sum_{j=1}^k \alpha_j \Phi_j(\mathbf{x}_i^0; c_0) - u_{\text{tar}}(\mathbf{x}_i^0) \right\|^2.$$

ALGORITHM 4: Training the stream function within a shape space for div-free Laplacian modes.

Input : A family of surfaces $\{\mathcal{M}_c\}_{c \in C}$ represented by triangular meshes; the shape-control domain C ; the number of modes k ; the number of training epochs N ; the number of surface samples N_M ; and the number of narrow-band samples N_N .

Output : A set of SIREN neural networks $\{h_{\theta_j}\}_{j=1}^k$, where each $h_{\theta_j} : \mathcal{N}(\mathcal{M}_c) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$ represents a stream function. For each control parameter c , the networks produce Laplacian modes $\{\Phi_j(\cdot; c) = \text{rot}_{\mathcal{M}_c} h_{\theta_j}(\cdot; c)\}_{j=1}^k$ that vary continuously over the shape space.

- 1 **Initialization**: Initialize all networks $\{h_{\theta_j}\}_{j=1}^k$ with SIREN initialization. For each $\mathcal{M}_c$, build a sparse grid around $\mathcal{M}_c$ and cache all sparse grids. At each grid vertex, evaluate the SDF value and compute the required derivatives by finite differences;
- 2 **for** $I = 1$ **to** N **do**
- 3 Randomly sample a control parameter $c \in C$;
- 4 Sample N_M points $X = \{x_i\}$ on $\mathcal{M}_c$ and N_N points $Y = \{y_i\}$ in its narrow band;
- 5 Query normals and SDF derivatives for the sampled points;
- 6 Evaluate the modes $\{\Phi_j(\cdot; c)\}_{j=1}^k$ from the networks $\{h_{\theta_j}\}_{j=1}^k$;
- 7 Apply Gram–Schmidt orthogonalization to the modes to obtain $\{\tilde{\Phi}_j\}_{j=1}^k$;
- 8 Compute the loss $\mathcal{L}(\{\tilde{\Phi}_j\}_{j=1}^k, X, Y, c)$ according to Sec. 5.2, Eq. (13);
- 9 Update the network parameters by back-propagation;
- 10 **return** $\{h_{\theta_j}\}_{j=1}^k$, which generate the shape-conditioned modes $\{\Phi_j(\cdot; c)\}_{j=1}^k$ for any $c \in C$;

On each fixed surface $\mathcal{M}_{c_m}$, the velocity field is reconstructed through

$$u^t(x_i^m) = \sum_{j=1}^k \alpha_j^t \Phi_j(x_i^m; c_m).$$

We then use a semi-Lagrangian integrator to advect both the density and the velocity coefficients. For each vertex x_i^m , we compute the origin position

$$\tilde{x}_i = \text{cp}_{\mathcal{M}_{c_m}}(x_i^m - \Delta t u^t(x_i^m)),$$

where $\text{cp}_{\mathcal{M}_{c_m}}$ projects the backtraced point to the closest point on the current surface. If $\tilde{x}_i$ lies in a triangle (a, b, c) with barycentric weights $(\lambda_a, \lambda_b, \lambda_c)$, the density is updated by

$$\rho_i^{t+1} = \lambda_a \rho_a^t + \lambda_b \rho_b^t + \lambda_c \rho_c^t.$$

The velocity at the origin position is similarly interpolated as

$$\tilde{u}_i = \lambda_a u^t(x_a^m) + \lambda_b u^t(x_b^m) + \lambda_c u^t(x_c^m).$$

Since $\tilde{u}_i$ lies in the tangent plane at $\tilde{x}_i$, we transport it to $T_{x_i^m} \mathcal{M}_{c_m}$ using the minimum rotation between the corresponding surface normals, obtaining $\tilde{u}_i$. The transported velocity field may not be exactly representable by the first k modes, so we compute the next coefficient vector through

$$\alpha^{t+1} = \arg \min_{\alpha} \sum_i A_i^m \left\| \sum_{j=1}^k \alpha_j \Phi_j(x_i^m; c_m) - \tilde{u}_i \right\|^2.$$

ALGORITHM 5: Training the cross field across offset surfaces.

Input : A surface $\mathcal{M}$ represented by a triangular mesh; an offset-parameter domain C , defaulting to $[-0.02d_{\text{bbox}}, 0.02d_{\text{bbox}}]$, where d_{bbox} is the bounding-box diagonal; the number of training epochs N ; and the number of narrow-band samples N_N .

Output : A SIREN neural network $f_{\theta} : \mathcal{N}(\mathcal{M}_c) \times \mathbb{S}^2 \times C \rightarrow \mathbb{R}$ that represents an angle field on the offset surface $\mathcal{M}_c$. For any offset value $c \in C$, the network produces a frame field on $\mathcal{M}_c$.

- 1 **Initialization**: Initialize f_{θ} with SIREN initialization;
- 2 **for** $I = 1$ **to** N **do**
- 3 Randomly sample an offset parameter $c \in C$;
- 4 Construct the offset surface $\mathcal{M}_c$ from $\mathcal{M}$ using the offset distance c ;
- 5 Build a sparse grid around $\mathcal{M}_c$. At each grid vertex, evaluate the SDF value and compute the required derivatives by finite differences;
- 6 Randomly sample one point on each face of $\mathcal{M}_c$ using barycentric coordinates, yielding surface samples $X = \{x_i\}$;
- 7 For each sampled point x_i , construct its neighborhood $\mathcal{NB}(i)$ from the vertex-based face adjacency of the mesh;
- 8 Sample N_N points $Y = \{y_i\}$ in the narrow band of $\mathcal{M}_c$;
- 9 Query normals and SDF derivatives for the sampled points;
- 10 Construct a local reference frame F_i at each surface sample x_i according to Alg. 1;
- 11 Evaluate the angle field $\alpha_i = f_{\theta}(x_i, n_i, c)$, rotate the local reference frame by α_i , and convert the resulting frame $F_i(\alpha_i)$ to its spherical-harmonic coefficients $q(F_i(\alpha_i))$ on $\mathcal{M}_c$;
- 12 Compute the loss $\mathcal{L}(f_{\theta}, X, Y, \{\mathcal{NB}(i)\}, c)$ according to Sec. 5.3, Eq. (14);
- 13 Update the network parameters by back-propagation;
- 14 **return** f_{θ} , which generates offset-conditioned frame fields on $\mathcal{M}_c$ for any $c \in C$;

Finally, the updated velocity is reconstructed as

$$u^{t+1}(x_i^m) = \sum_{j=1}^k \alpha_j^{t+1} \Phi_j(x_i^m; c_m).$$

After running a fixed number of semi-Lagrangian steps on $\mathcal{M}_{c_m}$, we move to the next surface $\mathcal{M}_{c_{m+1}}$. Because the mesh sequence has vertex correspondence, the density is transferred by keeping the same vertex indices, i.e.,

$$\rho_i^{m+1} \leftarrow \rho_i^m.$$

The coefficient vector is also inherited,

$$\alpha^{m+1} \leftarrow \alpha^m,$$

and the geometry change is handled by evaluating the same coefficients on the new shape-conditioned modes $\{\Phi_j(\cdot; c_{m+1})\}_{j=1}^k$.

F Experimental Settings in Detail

Screened Poisson Equations. In our implementation, we set $\lambda_C = 1.0$ and $\lambda_{\text{cp}} = 1.0$. We train for 5K epochs for a single instance and 20K epochs for the generalized setting. When the input source is a

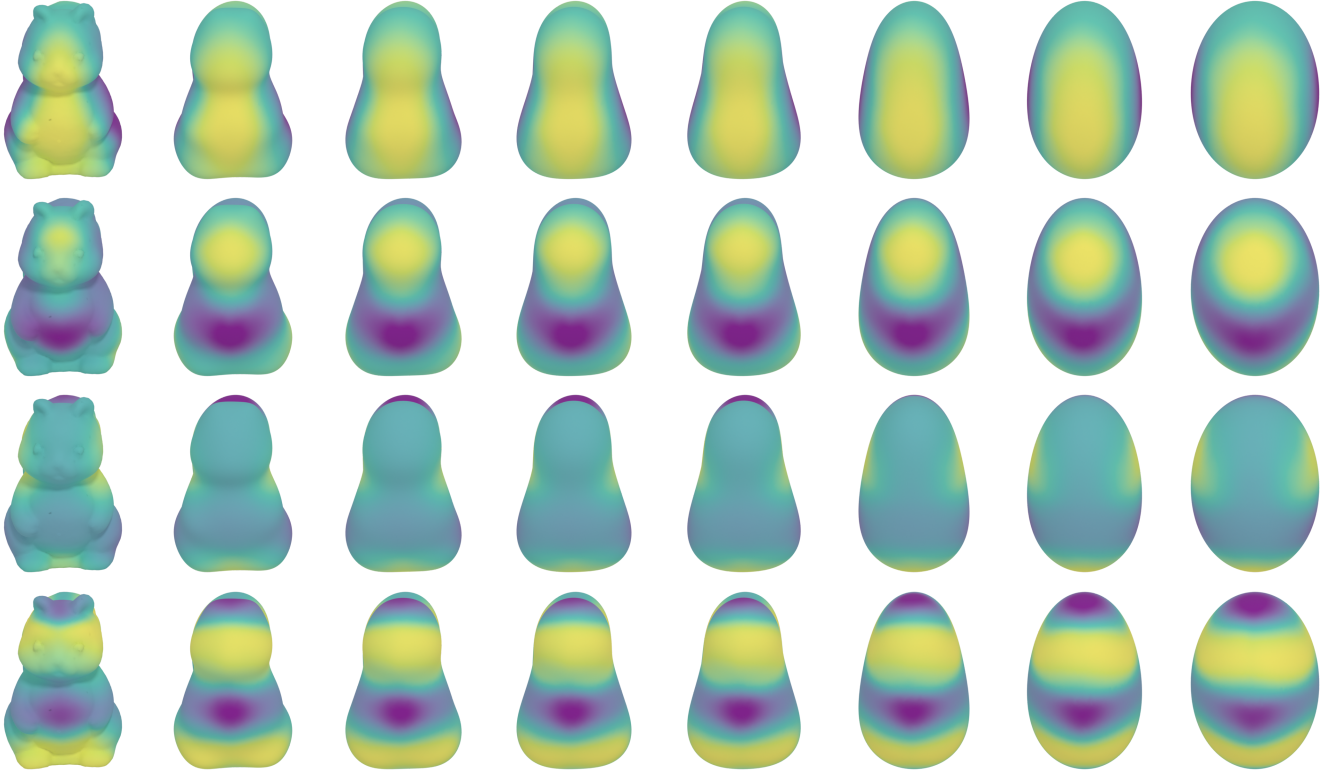


Fig. S3. **Generalizable scalar Laplacian modes.** The figure shows four scalar Laplacian modes using false colors on an MCF-deforming surface. Each mode evolves continuously with the deformation of the surface while preserving its eigenfunction property.

curve, we uniformly sample points along the curve to evaluate the Dirichlet constraints. We randomly sample 30K surface points and 20K narrow-band points at each epoch.

Vector Field Design. In our implementation, we set $\lambda_G = 1$, $\lambda_C = 10$, and $\lambda_{cp} = 1$. We train for 5K epochs for a single instance and 20K epochs for the generalized setting. We randomly sample 30K surface points and 20K narrow-band points at each epoch.

Scalar Laplacian Modes. In our implementation, we set $\lambda_{cp} = 10$, $\lambda_{\perp} = 10^3$, and $\lambda_{\parallel} = 10^3$. We train for 8K epochs for a single instance and 20K epochs for the generalized setting. We randomly sample 30K surface points and 20K narrow-band points at each epoch.

Divergence-Free Laplacian Modes. In our implementation, we set $\lambda_G = 1$, $\lambda_{cp} = 10$, $\lambda_{\perp} = 10^3$, and $\lambda_{\parallel} = 10^3$. For bounded 2D surfaces, we impose the free-slip boundary condition with $\lambda_{bc} = 10^4$. We train for 8K epochs for a single instance and 20K epochs for the generalized setting. We randomly sample 30K surface points and 20K narrow-band points at each epoch.

Cross Field Generation. In our implementation, the alignment weight λ_{align} follows an annealing schedule: it linearly increases from 0 to 1.0 during the first 20% of training, and then decreases from 1.0 to 0.2 with a cosine schedule during the last 20%. λ_{cp} is also annealed: it is initialized to 0 and linearly increases to 1.0 during the

last 40% of training. We train for 10K epochs for a single instance and 20K epochs for the generalized setting.

Note that we anneal the weight of the principal-curvature alignment loss because robust curvature estimation on surfaces is nontrivial [Yang et al. 2006], as the estimated directions may be inconsistent in high-curvature regions. Nevertheless, the estimated principal directions are reliable in most regions. We therefore use an annealing strategy: the alignment weight is weakened at the beginning and near the end of training, allowing the smoothness energy to correct wrong cross-field alignments caused by numerical errors in curvature estimation.

G Additional Experimental Results

Finally, we present several additional experimental results that are not included in the main text.

In Fig. S3, we show the deformation of four scalar Laplacian modes during the continuous mean-curvature flow (MCF) deformation of a surface. Similarly, we show that of four divergence-free vector Laplacian modes on another surface, as illustrated in Fig. S4. With an additional parameter encoding the time of the MCF, these modes change continuously across evolving surfaces while keeping their eigenfunction properties.

In Fig. S5, we visualize streamline comparisons with NeurCross on ten models. NeurCross jointly optimizes a neural SDF and a cross

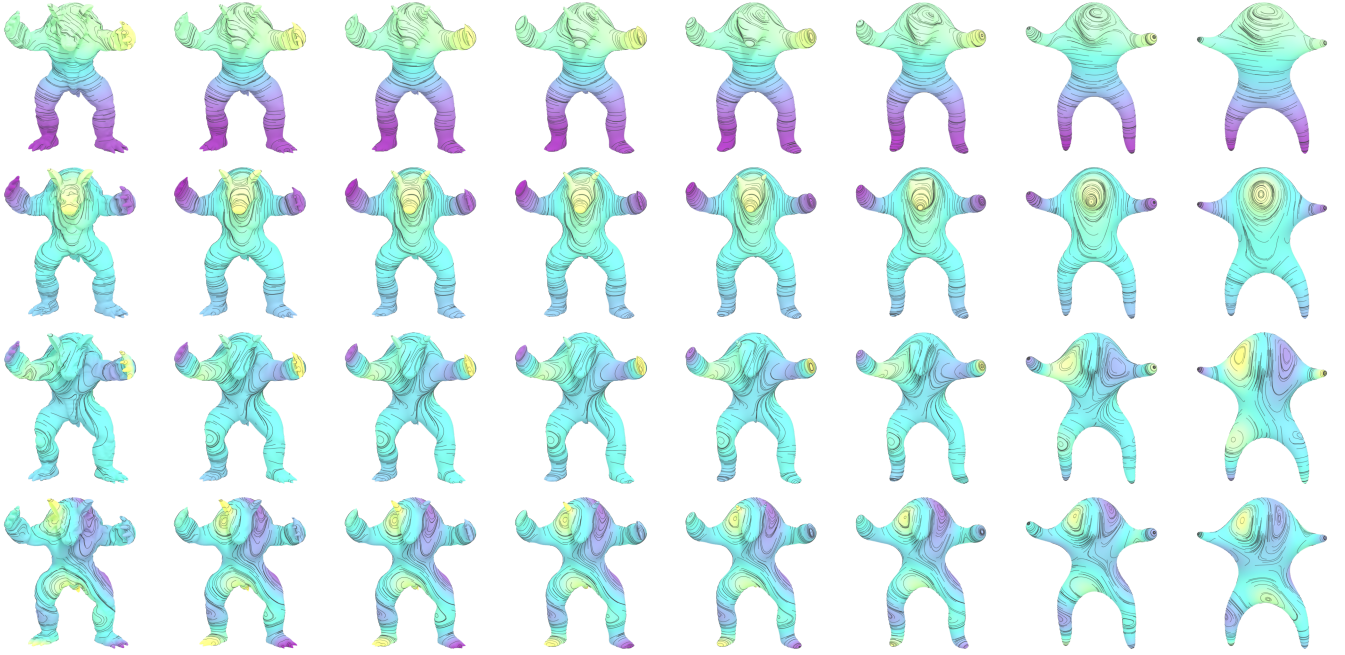


Fig. S4. **Generalizable divergence-free Laplacian modes.** The figure shows four divergence-free vector Laplacian modes using streamlines on an MCF-deforming surface. We use false colors to indicate the stream function here. Each mode evolves continuously with the deformation of the surface.

field, using the neural SDF to compute principal curvatures and guide the cross field toward principal curvature directions. However, computing the surface Hessian from a neural SDF requires second-order automatic differentiation, which can be numerically unstable. Moreover, the neural SDF itself is influenced by the current cross field, making it difficult to robustly align with sharp features. In contrast, our method uses a sparse-grid SDF and explicitly computes surface derivatives via finite differences. The resulting SDF-based principal curvature directions and curvature-anisotropy weights

[Zhuang et al. 2014] guide the cross field to better align with high-curvature features.

References

- Yong-Liang Yang, Yu-Kun Lai, Shi-Min Hu, and Helmut Pottmann. 2006. Robust principal curvatures on multiple scales. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. 223–226.
- Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. 2020. Octahedral Frames for Feature-Aligned Cross Fields. *ACM Trans. Graph.* 39, 3, Article 25 (2020). doi:10.1145/3374209
- Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. 2014. Anisotropic geodesics for live-wire mesh segmentation. *Comput. Graph. Forum* 33, 7 (2014), 111–120. doi:10.1111/cgf.12479

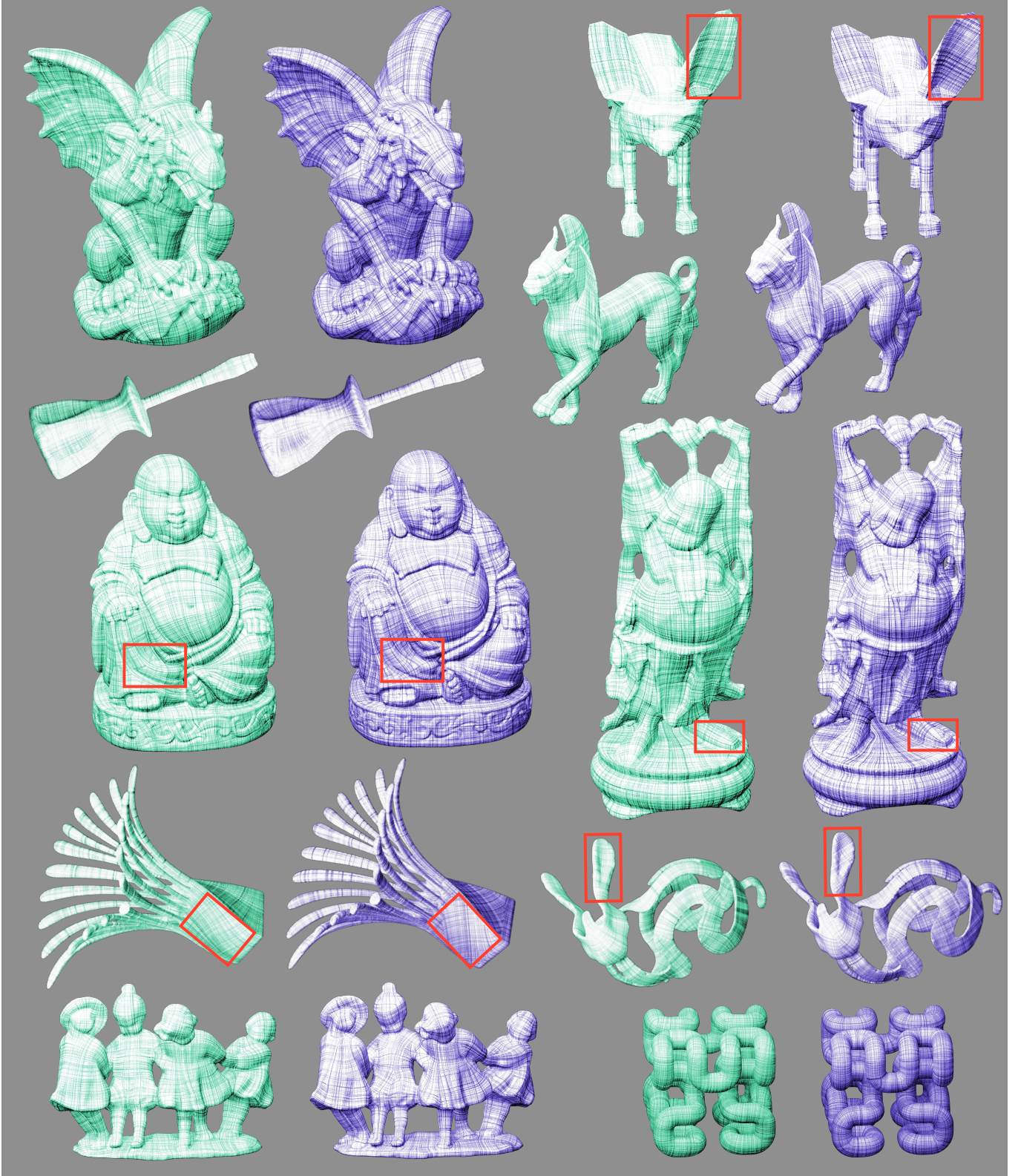


Fig. S5. **Streamlines of cross frames.** We compare the streamlines generated by our method (in green) and NeurCross (in purple) on ten representative models. Guided by SDF-based principal curvature directions and curvature-anisotropy weights (Eq. (16) of the main paper), our streamlines follow principal directions and sharp features more closely; see Sec. G for a discussion.