

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

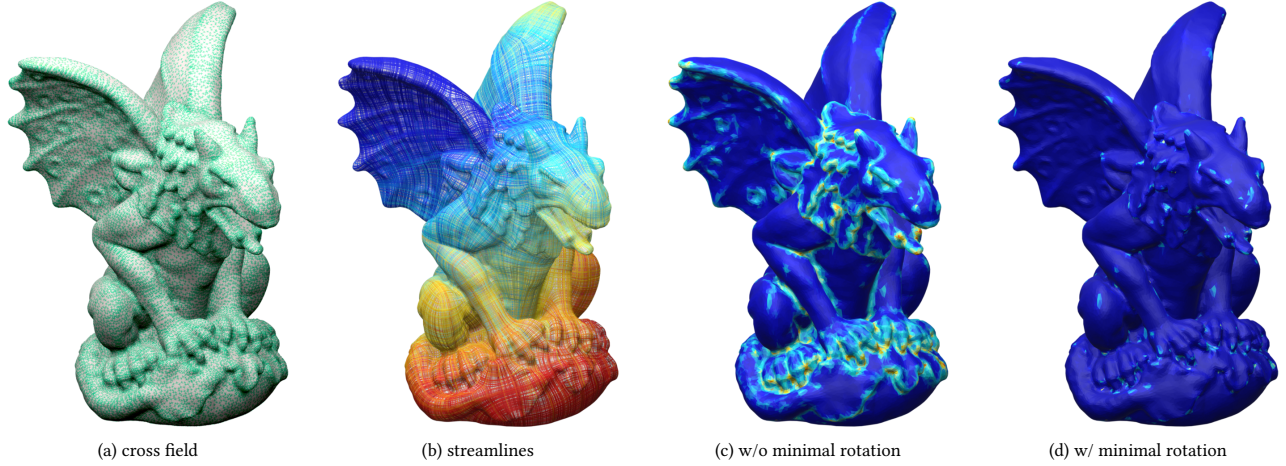


Fig. S1. **Effect of the normal minimal rotation.** Given a high-quality cross field (a) and its streamlines (b), the Dirichlet energy computed without normal minimal rotation remains large in curved regions (c), causing the optimization to keep penalizing high-curvature areas in practice. With normal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

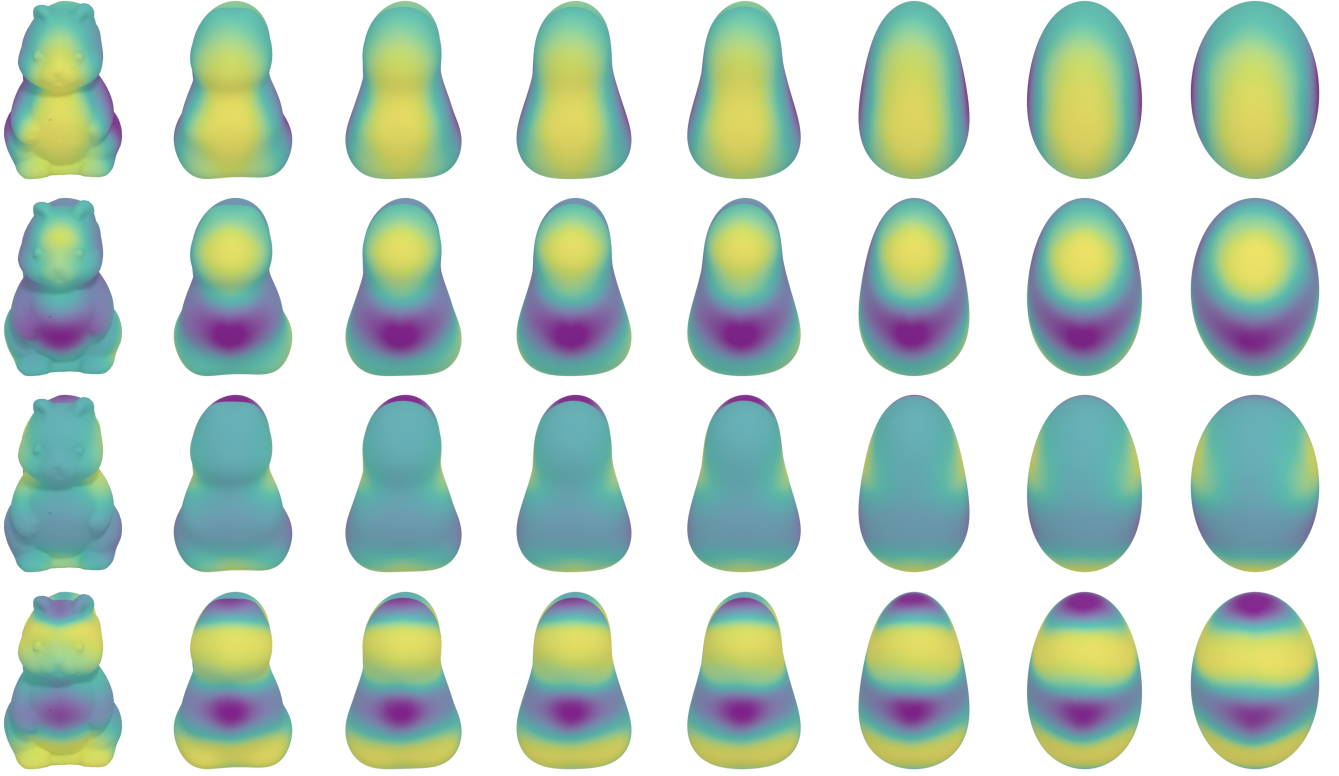


Fig. S3. **Generalizable scalar Laplacian modes.** The figure shows four scalar Laplacian modes using false colors on an MCF-deforming surface. Each mode evolves continuously with the deformation of the surface while preserving its eigenfunction property.

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

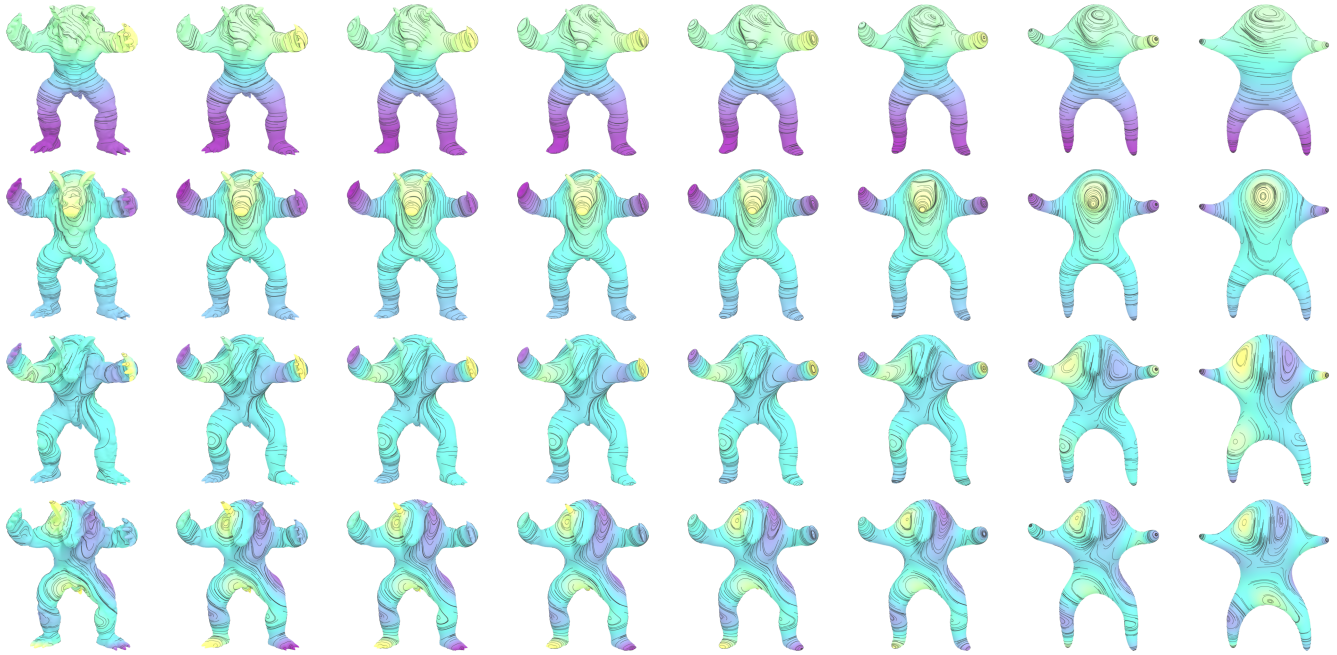


Fig. S4. **Generalizable divergence-free Laplacian modes.** The figure shows four divergence-free vector Laplacian modes using streamlines on an MCF-deforming surface. We use false colors to indicate the stream function here. Each mode evolves continuously with the deformation of the surface.

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

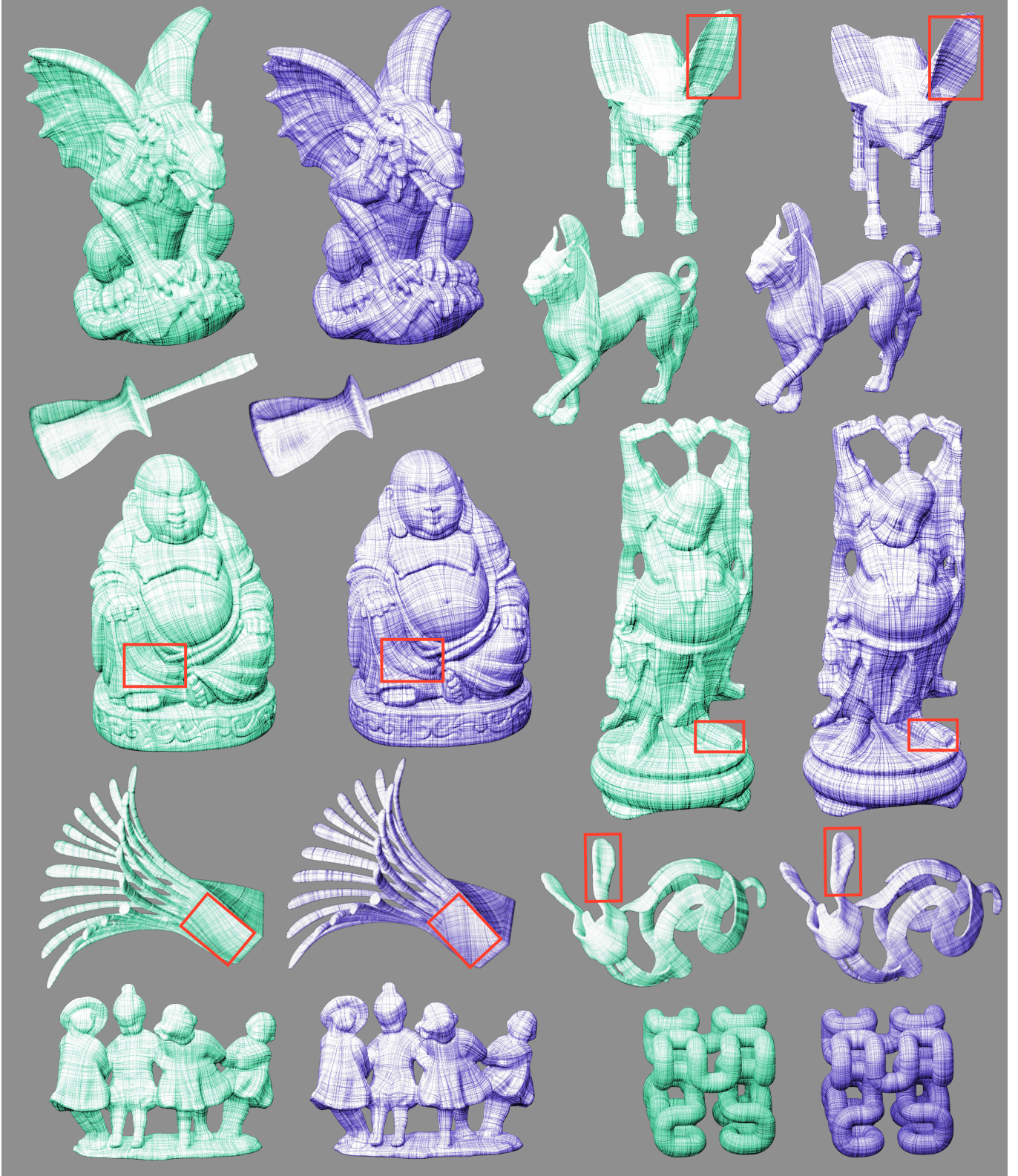


Fig. S5. **Streamlines of cross frames.** We compare the streamlines generated by our method (in green) and NeurCross (in purple) on ten representative models. Guided by SDF-based principal curvature directions and curvature-anisotropy weights (Eq. (16) of the main paper), our streamlines follow principal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