Supplementary Material for Learning Geometry-Aware Embedding Fields for Intrinsic Riemannian Mappings

Bo Pang[1,2], Simone Foti[1], and Tolga Birdal[1] *

[1] Imperial College London
[2] Peking University

Appendix

This supplementary document provides additional details that are not included in the main paper due to space constraints.

S1 Input Representation for Exponential Map

In classical differential geometry, the exponential map is defined on the tangent plane, so its input is naturally a 2D tangent vector. In our implementation, however, ExpNet receives a 3D vector in ambient space. This is not a different mathematical object; it is only a different coordinate representation of the same tangent direction.

The conversion is straightforward. For each source point p, we first build a local tangent frame with two orthonormal basis vectors on $T_p\mathcal{M}$. A 2D tangent vector can then be embedded into 3D by taking its linear combination under this local basis. Conversely, a 3D vector constrained to the tangent plane can be projected back to 2D coordinates under the same frame.

We adopt the 3D representation mainly for robustness in learning. A purely 2D parameterization requires choosing a local chart orientation, and that choice is not unique: different but equally valid basis orientations can produce different coordinate values for the same geometric direction. This introduces unnecessary coordinate inconsistency across training samples. By using tangent-aligned 3D relative vectors, we reduce this basis-selection ambiguity and make the network behavior more stable in practice.

Therefore, our formulation remains geometrically equivalent to the standard 2D exponential-map definition. The difference is only the coordinate system used for network input, not the underlying map itself, and it can be easily eliminated by projection back to 2D after network inference. For the sake of convenience, these two representations might be used interchangeably in this paper.

* Corresponding author.

S2 Network Architecture and Training Details

Hypernetwork Configuration Our method employs a two-stage framework: a backbone network for context encoding and a pair of hyper-modulated decoders for exp map and log map. The backbone extracts conditional features at arbitrary position in the 3D space. Such features then drive two structurally isomorphic modulated networks:

- **ExpNet Branch**: Maps tangent vectors to surface positions.
- **LogNet Branch**: Maps surface positions to tangent vectors.

Both branches share the same architecture capacity but learn distinct semantic tasks. They utilize a tri-plane modulation mechanism where the condition embedding generates three orthogonal 2D feature planes. The query coordinates are projected onto these planes, and features are retrieved via bilinear interpolation and concatenated for the final lightweight MLP decoding.

Specifically, the architecture consists of three orthogonal feature planes (xy, xz, yz), each with a resolution of 16×16 and a feature dimension of $C = 8$. A linear generator maps the input condition embedding to the plane features (totaling $3 \times 16 \times 16 \times 8$ parameters). During inference, we query the triplane features via bilinear interpolation and concatenate them with the raw input coordinates. This aggregated feature vector is then processed by a lightweight MLP decoder with 3 layers and 256 hidden units to predict the target output.

Backbone Architecture (O-CNN) We use an Octree-based U-Net as the backbone to extract multi-scale geometric features. The architecture follows a standard encoder-decoder design with skip connections:

- **Encoder**: Consists of 4 stages with channel dimensions $[64, 64, 64, 128]$. Each stage includes multiple residual blocks (specifically $[2, 3, 3]$ blocks respectively) to progressively extract semantic features.
- **Decoder**: Mirrors the encoder with channel dimensions $[128, 128, 128, 128]$ and residual blocks $[2, 2, 2]$. Skip connections fuse features from the encoder to preserve local geometric details.
- **Header**: A final projection layer maps the decoded features to a 256-dimensional embedding space, which serves as the condition for the modulation networks.

The input to the backbone is the octree representation of the shape, and the output is a continuous embedding field that can be queried at any vertex location. We train our model using the **AdamW** optimizer with a base learning rate of 1×10^{-3}. We employ a linear learning rate decay that decays the learning rate to 0 by the end of training. In every iteration during training, we sample $10,000$ query points per mesh.

S3 Extension to Other 3D Representations

Although our main experiments focus on point clouds, our method is agnostic to the underlying 3D representation, provided that a surface can be sampled.

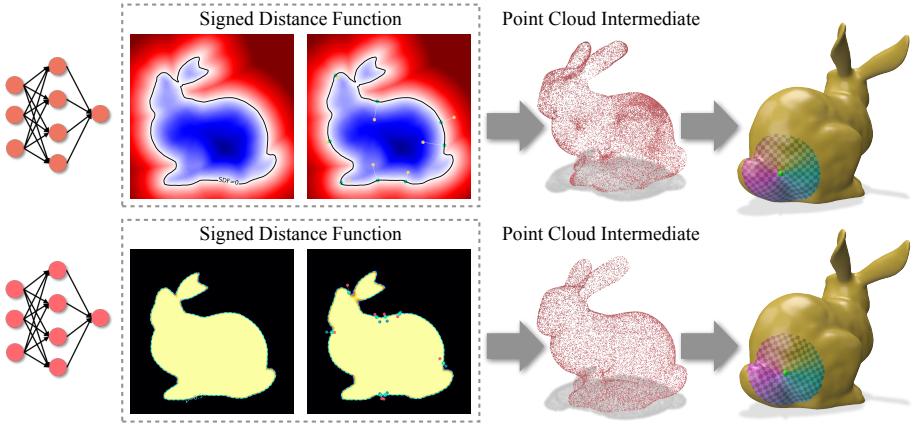

Fig. S1: Unified Pipeline for Various 3D Representations. Whether starting from a signed distance function or an occupancy grid, we first extract an intermediate point cloud (which may be inaccurate, or not uniform). This intermediate representation is directly compatible with our learning pipeline, and can be directly fed into our network.

The core idea is to leverage an intermediate point cloud representation, which can be generated easily from any 3D implicit representation.

As shown in S1, for a given 3D implicit representation, we can use sphere tracking, or gradient-free bisection, to extract a high-fidelity point cloud. For SDF, to sample surface points, we initialize random queries x_0 in the bounding box and iteratively project them onto the zero-level set using the signed distance gradients:

$$x_{k+1} = x_k - f_\theta(x_k) \cdot \frac{\nabla_x f_\theta(x_k)}{\|\nabla_x f_\theta(x_k)\|_2} \tag{S1}$$

This process, typically converging within 5 iterations, yields a dense point cloud P_{sdf} that lies precisely on the implicit surface. For binary Occupancy Grids, we employ a bisection-based sampling strategy. We sample pairs of points (p_{in}, p_{out}) and identify edges that cross the surface boundary (where $g_\phi(p_{in}) \cdot g_\phi(p_{out}) < 0$). We then perform a bisection search along the ray connecting these pairs to find the exact zero-crossing point p_{surf} where $g_\phi(p_{surf}) \approx 0$.

S4 Detailed Baseline Implementations

We primarily compare our method against the state-of-the-art heat diffusion methods and purely heuristic geometric methods. In this section, we provide detailed implementations of these baselines. In all our experiments, we assume the input point cloud having no normal information. When normal is needed, we estimate the local normal using PCA over neighboring points.

Please note that algorithms designed solely for computing geodesic distances cannot be directly repurposed as baselines for the exponential or logarithmic maps. While these methods efficiently compute the scalar distance field d, they do not inherently provide the direction information required for the two maps. Recovering the gradient ∇d from a discrete distance field to approximate the log map is often numerically unstable and sensitive to triangulation quality. Therefore, we do not compare our method with those methods.

Heat Method. The Heat Method approximates geodesic distances and recovers the logarithmic map by solving the heat diffusion equation on the manifold. We utilize the `potpourri3d` [1] library for efficient inference on both manifold meshes and point clouds. For meshes, we use the `MeshVectorHeatSolver`, and for point clouds, we use the `PointCloudHeatSolver` which relies on a k-NN implicit neighborhood graph. The results on manifold meshes are used as ground truth for error evaluation, while the results on point clouds are used as a baseline of performance comparison.

Linear Projection (Log Map Baseline). This baseline tests the performance of simple linear tangent space dimensionality reduction when ignoring complex surface curvature. First, for point cloud inputs, we estimate the local normal N_i for each point V_i using PCA on its nearest neighbors. Given a source point V_s and its normal N_s, we then construct an orthogonal basis $u, v \in \mathbb{R}^3$ for the tangent plane. For any point $V_i \in \mathbb{R}^3$, we compute the relative vector $\Delta V = V_i - V_s$ and project this vector onto the tangent basis to obtain 2D coordinates: $x_i = \Delta V \cdot u, \quad y_i = \Delta V \cdot v$.

Heuristic Walking. This iterative method simulates geodesic walking to approximate the exponential map. We first estimate a dynamic step size based on the local point density (e.g., 0.5 times the average edge length). In each step, we move along the current direction vector, finding the nearest neighbor on the manifold. We then re-estimate the local normal and tangent plane using PCA, and project the current position onto the new tangent plane to correct for drift. Finally, we update the direction vector by removing the component parallel to the new normal to ensure the walk remains tangent to the surface.

Linear Projection (Exp Map Baseline). This is a single-step projection method serving as a simplified exponential map. Given a tangent vector v_{tan} at source P_{src}, we compute a candidate point in 3D Euclidean space: $P_{est} = P_{src} + v_{tan}$. We then use a K-D Tree to find the nearest point on the manifold to P_{est} and snap the result to the surface.

S5 Specifics of the Data Filtering

To ensure high-quality supervision, we implement a data filtering mechanism that removes ambiguous samples, particularly those near the cut locus where the exponential map is discontinuous or multi-valued in the inverse direction. Our filtering strategy is based on checking the local consistency of the mapping between the surface manifold and the tangent space.

Intuitively, if two samples are close in the tangent space, they should also remain close on the 3D surface; when this assumption is violated, the pair usually lies near cut-locus regions, and should not be used for training. Specifically, given a set of surface points $P = \{p_i\} \subset \mathbb{R}^3$ (denoted as `pos` in our code) and their corresponding tangent vectors $V = \{v_i\} \subset \mathbb{R}^3$ (denoted as `vec`), the filtering process proceeds as follows:

1. **Neighborhood Search in Surface Space:** For each point p_i, we identify its k-nearest neighbors $\mathcal{N}(p_i)$ based on the Euclidean distance in the surface space P. We typically set $k = 5$.
2. **Consistency Check in Tangent Space:** We then examine the corresponding tangent vectors $\{v_j \mid p_j \in \mathcal{N}(p_i)\}$. We compute the pairwise distances in the tangent space between the central point's vector v_i and its neighbors' vectors v_j:
$$d_{ij} = \|v_i - v_j\|_2, \quad \forall p_j \in \mathcal{N}(p_i) \tag{S2}$$
3. **Divergence Thresholding:** We define the local divergence score δ_i as the maximum distance in the tangent space among the neighbors. Points with a divergence score exceeding a threshold τ (default $\tau = 0.5$) are considered ambiguous and are masked out during training.

This criterion effectively filters out points where small changes in surface position correspond to large jumps in the tangent vector space, which is characteristic of discontinuities across the cut locus.

S6 Log Map Stitching Algorithm

To extend the logarithmic map beyond the a small local radius, we use a chart stitching strategy based on weighted rigid alignment and confidence-aware blending. The algorithm starts from one source vertex, computes a local log map in a bounded radius, and then propagates to new centers in a BFS manner. At each center, we only keep points inside the local radius, so each chart is constrained to a region where the local parameterization is more reliable.

Since local log maps become less stable near patch boundaries, we assign higher confidence to points near the chart center and lower confidence to peripheral points. For a point with normalized radial distance d_x in the current chart, we use:
$$w_{local}(x) = \left(\max(0, 1 - d_x^2)\right)^2 \tag{S3}$$
where d_x is normalized by the local radius. This smooth falloff is used consistently in both alignment and blending, which makes the procedure robust to noisy local predictions.

When a chart has at least 3 vertices overlapping with the current global atlas, we estimate a 2D rigid transform for it using Kabsch-Umeyama algorithm, to align it to the current global atlas. After alignment, transformed local coordinates are merged into the atlas.

Chart expansion also follows confidence-aware heuristics. New BFS centers are selected from a boundary shell of the current chart (normalized radius roughly between 0.75 and 0.95), which encourages outward coverage while maintaining overlap. To avoid redundant expansion, candidates are distributed by angular bins in local 2D coordinates and then prioritized by distance from the original source; in each expansion round we add a small set of centers and continue until the preset depth is reached. In practice, this gives a good balance between coverage and numerical stability.

S6.1 Retraction Properties

To verify the local behavior of our learned exponential and logarithmic maps, we evaluate four simple diagnostics at each source point $x \in \mathcal{M}$. They test two expected properties: first, the maps should be correctly anchored at the base point, namely $\mathrm{Exp}_x(0) = x$ and $\mathrm{Log}_x(x) = 0$; second, their first-order behavior should be close to the identity:

$$d(\mathrm{Exp}_x)_0 \approx I, \qquad d(\mathrm{Log}_x)_x \approx I. \tag{S4}$$

Here, $\mathrm{Exp}_x : T_x\mathcal{M} \to \mathcal{M}$ denotes the learned exponential map centered at x, and $\mathrm{Log}_x : \mathcal{M} \to T_x\mathcal{M}$ denotes the learned logarithmic map. For evaluation only, we introduce a local 2D coordinate system around the source point and represent the differentials of both maps in these coordinates. In this setting, $d(\mathrm{Exp}_x)_0$ and $d(\mathrm{Log}_x)_x$ are 2×2 matrices, and I denotes the 2×2 identity matrix. We report:

Zero-Point Errors.. We first measure the errors of the two maps at the base point:

$$E_{\mathrm{Exp},0} = \|\mathrm{Exp}_x(0) - x\|, \qquad E_{\mathrm{Log},0} = \|\mathrm{Log}_x(x)\|. \tag{S5}$$

These two quantities test whether the learned maps send the tangent origin and the source point to each other as expected.

Differential Errors.. We then evaluate how far the first-order behavior of the two maps is from the identity:

$$E_{\mathrm{Exp},d} = \|d(\mathrm{Exp}_x)_0 - I\|_F, \qquad E_{\mathrm{Log},d} = \|d(\mathrm{Log}_x)_x - I\|_F, \tag{S6}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Smaller values indicate that the learned maps behave more like the ideal first-order identity in a local neighborhood of the source point.

Table S1 reports the corresponding results for our method. Both zero-point errors are very small, indicating accurate anchoring at the source point, while the differential errors are also small, suggesting that the learned maps remain close to the ideal first-order identity behavior.

S7 Additional Ablation Study

To further analyze the impact of architectural design, we compare different backbone-decoder combinations under the same training and evaluation protocol, as in Tab. S2. We keep the loss terms unchanged, and only replace the

Table S1: Local zero-point and differential consistency checks for our method. Lower is better for all metrics.

Method	Exp Zero ↓	Exp Differential Error ↓	Log Zero ↓	Log Differential Error ↓
Ours	0.002	0.133	0.004	0.149

network architecture. As shown in the table, our full model with an octree backbone and triplane decoding achieves the best performance in our setting.

Table S2: Ablation Study on Network Architecture. We compare different backbone-decoder combinations.

Architecture	$\mathcal{E}_{\text{MDD}}$ (Log)	$\mathcal{E}_{\theta}$ (Log)	$\mathcal{E}_{\text{Exp}}$ (Exp)
OCNN + TriPlane	0.11	11.1	0.026
PointNet + TriPlane	0.31	29.2	0.056
OCNN + SIREN	0.13	14.3	0.031

S8 Failure Cases

Similar to previous Heat-based methods, our approach has poor quality on thin-structures, as in Fig. S2 A representative failure mode of our method occurs on thin articulated structures, such as wrists, fingers, or other slender parts with rapidly changing local geometry. In these regions, nearby surface patches can be very close in Euclidean space while remaining intrinsically different. In addition, our method only take point cloud as input, which impose more difficulties to the mapping.

References

1. Sharp, N.: Potpourri3D: An invigorating blend of 3d geometry tools in python, `https://github.com/nmwsharp/potpourri3d`

Fig. S2: Failure case on a thin articulated region. With the source point placed near the wrist, the predicted local map becomes less stable across the wrist-hand transition, where nearby surface regions are close in Euclidean space but have different intrinsic geodesic behavior.