

Local Surface Parameterizations via Smoothed Geodesic Splines

ABHISHEK MADAN, University of Toronto, Canada

DAVID I.W. LEVIN, University of Toronto and NVIDIA, Canada

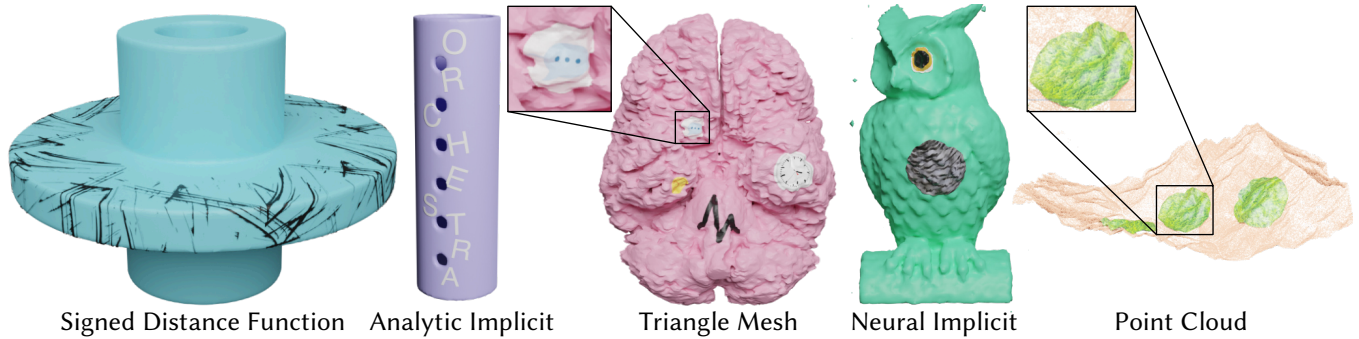


Fig. 1. The local parameterizations produced by our method provide low-distortion maps on a variety of different types of geometry. We can place scratches on a well-used children’s toy represented by a signed distance function (SDF); stick letters onto a flute composed with CSG operations, which as a result does not behave like an SDF; place pictorial labels on a triangle mesh of a brain reconstructed from an MRI scan (zoom in to see more details); fill in one eye and some plumage of an owl represented by a neural implicit function; and even add some forest decals onto a terrain point cloud with over 29 million points acquired from LiDAR data.

We present a general method for computing local parameterizations rooted at a point on a surface, where the surface is described only through a signed implicit function and a corresponding projection function. Using a two-stage process, we compute several points radially emanating from the map origin, and interpolate between them with a spline surface. The narrow interface of our method allows it to support several kinds of geometry such as signed distance functions, general analytic implicit functions, triangle meshes, neural implicits, and point clouds. We demonstrate the high quality of our generated parameterizations on a variety of examples, and show applications in local texturing and surface curve drawing.

CCS Concepts: • **Computing methodologies** → **Parametric curve and surface models**; **Shape analysis**.

Additional Key Words and Phrases: local parameterizations, exponential maps, implicit surfaces, point clouds

ACM Reference Format:

Abhishek Madan and David I.W. Levin. 2025. Local Surface Parameterizations via Smoothed Geodesic Splines. *ACM Trans. Graph.* 1, 1, Article 1 (January 2025), 14 pages. <https://doi.org/10.1145/3767323>

1 INTRODUCTION

Local parameterizations are particularly useful geometric tools mapping a planar coordinate system to a local, manifold region around a specified origin, and enable graphics tasks like placing decals and small texture patches on surfaces (Fig. 1). Local parameterizations are particularly well suited for complex geometry, trading a global

parameterization problem that must deal with topologically mandated difficulties like singularity placements and seams [Lévy et al. 2002; Mullen et al. 2008; Soliman et al. 2018] for a local problem with looser preconditions.

However, most of the approaches developed for computing local parameterizations only work on meshes, or more generally, graphs. This allows them to leverage existing tools such as discrete differential operators [Herholz and Alexa 2019; Sharp et al. 2019] or Dijkstra’s algorithm [Melvør and Reimers 2012; Schmidt 2013; Schmidt et al. 2006], but ties output quality to sampling or mesh quality. The sources of geometric representations in modern applications are quite varied (from large artist-generated meshes, to LiDAR-scanned point clouds, to hand-crafted implicits, and even neural network-encoded implicit functions [Müller et al. 2022; Takikawa et al. 2021]) and for many of these, it can be difficult to extract the high-quality samples and topological information required to apply previous methods. Meshes, for instance, may have low-quality triangles that are unsuitable for finite element-based computation; point clouds lack explicit topology altogether and may contain noise from the acquisition procedure; and implicit functions lack both explicit geometry and topology, with possibly noisy isosurfaces in the case of neural implicits. Many methods exist to address these problems, from remeshing [Botsch and Kobbelt 2004], to surface reconstruction [Chen et al. 2022; Chen and Zhang 2021; Ju et al. 2002; Lorensen and Cline 1987; Sellán et al. 2023], and even sampling the surface [De Goes et al. 2012; Witkin and Heckbert 1994], but these methods are quite disparate, and as such there is no unified way to generate local parameterizations for all these types of representations.

We propose a technique to avoid such preprocessing by only requiring a very narrow interface from the input geometry: a signed implicit function, along with a projection function derived from

Authors’ addresses: Abhishek Madan, University of Toronto, Toronto, Canada, amadan@cs.toronto.edu; David I.W. Levin, University of Toronto and NVIDIA, Toronto, Canada, diwlevin@cs.toronto.edu.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, <https://doi.org/10.1145/3767323>.

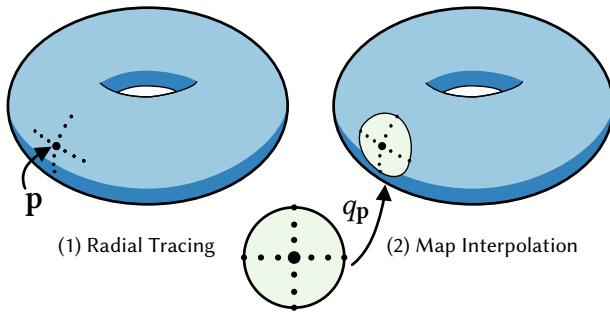


Fig. 2. Our method consists of two main components: tracing out points along radial curves from an origin p (left), and interpolating those points to form a continuous map q_p (right).

this implicit function. It is typically straightforward to compute this implicit representation: for example, obtaining an implicit function from meshes is simply a matter of computing the unsigned minimum distance and signing it with the winding number [Barill et al. 2018; Jacobson et al. 2013]. Our algorithm proceeds in two stages: first tracing geodesic-like paths from a prescribed origin, and then fusing them into a continuous local map using a spline surface. Tracing geodesics along surfaces via projection is a simple yet surprisingly delicate procedure (Fig. 3), so we also present a curvature-sensitive substepping procedure that avoids path-splitting steps in high-curvature regions, as well as an inter-curve smoothing method that greatly reduces map artifacts created by large variations in curvature across the surface. To our knowledge, our method is the first to develop a radial tracing method for local parameterizations that explicitly addresses both of these issues. By exclusively relying on pointwise implicit function queries rather than global surface operators, we obtain an output-sensitive method (i.e., where computational work scales in proportion to the size of the map) that produces local parameterizations meeting or exceeding the quality of previous methods, particularly on surfaces other than high-quality triangle meshes. We demonstrate the efficacy of our approach on a wide variety of geometric representations and regions on these geometries.

2 RELATED WORK

Computing geodesics. Several methods have been developed to compute geodesics on meshes, both exact and approximate [Crane et al. 2017; Kimmel 1998; Polthier and Schmies 2006; Surazhsky et al. 2005]. This problem comes in many flavours, from tracing out geodesic paths, to finding globally shortest geodesics between two given points, and even to finding all geodesics from one point to the rest of a set of points — see Crane et al. [2020] for a more comprehensive survey. Many of these methods can operate in a purely intrinsic setting due to them having access to an underlying mesh, which allows them to avoid an extrinsic projection-like procedure, but as such they are limited in the kinds of representations they can operate on. On the other hand, our work removes the underlying mesh assumption and works on a broader set of representations, though we note that we focus only on the geodesic tracing problem to build

our parameterizations, rather than the full set of geodesic problems that have been studied. Some recent work also explores regularized geodesic distances [Edelstein et al. 2023], but this is primarily restricted to meshes as well. While there has been some prior work on computing geodesics on implicit surfaces which also leverages a projection operator derived from the implicit function [Pedersen 1995], it requires an underlying blue noise global sampling of the surface to warm start the procedure, which our method avoids. Projection can also behave poorly in high-curvature regions (Fig. 3), which is why many previous methods prefer intrinsic mesh-based operations; however, we utilize a substepping procedure that preemptively avoids difficult projection scenarios, making it a feasible building block for our parameterization.

Parallel transport. A concept closely related to geodesics is parallel transport, an operation which provides a way to connect tangent spaces on a manifold. For example, tracing out geodesics is equivalent to parallel transporting the geodesic’s tangent vector across the surface, while simultaneously following along that tangent. One early method for discretizing parallel transport is Schild’s ladder, which approximates parallel transport by drawing geodesic parallelograms on the surface. Although it has found some use in image analysis [Lorenzi and Pennec 2014], it requires a method to find geodesics and as such does not help to compute geodesics on its own. There exist multiple frameworks for parallel transport on meshes which operate on different subsimplices, such as faces and the dual edges connecting them [Crane et al. 2010], and vertices [Knöppel et al. 2013]. A practical mesh-free way to compute parallel transport involves computing the smallest rotation to align nearby tangent planes [Schmidt et al. 2006], which is the approach we use as well.

Local parameterizations. Local parameterizations are also a well-explored topic in graphics. Pedersen [1995] presents an interactive system for drawing parameterization boundaries for rectangular patches, but this approach can be difficult to implement and use robustly, and also requires global surface samples. As such, many interactive local parameterization methods compute maps about a fixed origin instead, and let the algorithm expand the map on the surface rather than having the user guess appropriate boundaries. Local parameterization methods have also been developed using neural networks [Groueix et al. 2018; Srinivasan et al. 2023; Williams et al. 2019], but like Pedersen [1995], these methods do not center about a specific point and also must be trained on surface samples (though it is worth noting that the aim of these neural methods is to produce an ensemble of local charts for a global chart, so they are less concerned with user-defined initial conditions).

Exponential maps. Another local parameterization approach is based on the exponential map, which maps tangent space points onto the surface around an origin p . Discrete exponential maps are one example of such a method [Schmidt 2013; Schmidt et al. 2006]; despite the method name, it computes the logarithmic map (i.e., the inverse of the exponential map), given a graph connecting the point to nearby surface samples. As a result, the method is very sample-dependent, producing bad results in regions that are insufficiently sampled relative to their curvature. Another Dijkstra-like method is by Melvæ and Reimers [2012], which infers appropriate distance information by leveraging triangle mesh connectivity rather than a general graph. More recently, the vector heat method [Sharp et al.

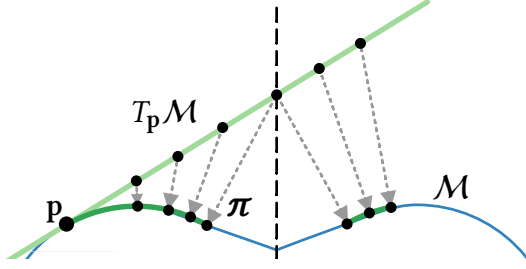


Fig. 3. Farther away from the origin \mathbf{p} on $T_{\mathbf{p}}\mathcal{M}$, the (unsmoothed) projection image of a tangent line onto \mathcal{M} can shrink significantly, and even split into disconnected pieces on \mathcal{M} past the medial axis (vertical dashed line).

2019] computes logarithmic maps using a connection Laplacian operator on the surface to approximate parallel transport, but the result quality is also strongly tied to the underlying triangulation (even with the aid of intrinsic Delaunay remeshing). Although the method only requires a way to compute the required operator matrix and thus is not strictly tied to triangle meshes, it can struggle to produce good results on poorly distributed point clouds, even when leveraging a high-quality point cloud Laplacian algorithm [Sharp and Crane 2020]. A similar method was concurrently proposed by Herholz and Alexa [2019], which also uses a heat method to compute log maps, but uses angles between distance gradients to approximate geodesic polar angles. Another drawback of PDE-based approaches is that they solve a global linear system to compute a local parameterization. Like Schmidt et al. [2006] and Schmidt [2013], our method is output-sensitive, doing work proportional to the size and complexity of the region to be parameterized, independent of the rest of the surface. This is because our method generates the samples it needs on the fly, and can be evaluated to produce high-quality maps on even poor sample sets. Local parameterizations based on exponential maps have also been used to develop a generalized convolution neural architecture for triangle meshes [Masci et al. 2015], where geodesics are estimated by triangle unfolding, which (unlike our approach) limits it to triangle meshes by construction.

3 METHOD

The theoretical framework for our local parameterization technique is based on the exponential map. Mathematically speaking, an exponential map $\exp_{\mathbf{p}}$ is a function rooted at a point \mathbf{p} on a manifold \mathcal{M} , which maps tangent vectors \mathbf{t} from \mathbf{p} 's tangent space $T_{\mathbf{p}}\mathcal{M}$ (represented by vectors in \mathbb{R}^2) and has the following properties:

- $\exp_{\mathbf{p}}(\mathbf{0}) = \mathbf{p}$ (i.e., the zero tangent vector maps to the origin \mathbf{p}); and
- $\exp_{\mathbf{p}}(\mathbf{t}) = \gamma(\|\mathbf{t}\|)$, where γ is an (arc-length parameterized) geodesic described by the initial conditions $\gamma(0) = \mathbf{p}$ and $\gamma'(0) = \mathbf{t}/\|\mathbf{t}\|$.

Essentially, exponential maps describe how to trace out geodesics emanating from \mathbf{p} onto \mathcal{M} . As such, they exhibit low distortion near the origin and are thus a very useful construction to build upon for local parameterizations. This radial tracing interpretation also suggests a two-step process for generating a local parameterization

around \mathbf{p} : (1) trace out several geodesic-like curves from \mathbf{p} (Sections 3.1–3.3), and (2) interpolate these curves radially to produce a continuous map (Section 3.4). Fig. 2 contains a visual overview of the algorithm.

Key to the generality of our approach is the limited interface it requires from input geometry. We assume that we are only given an implicit function $f(\mathbf{x})$ which contains the surface of interest in its zero isosurface \mathcal{M} . From f , we can obtain a gradient $\nabla f(\mathbf{x})$, a normal $\mathbf{n}(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$ (defined only on \mathcal{M}), and a projection operator $\pi(\mathbf{x})$ onto \mathcal{M} . We define π based on the general projection procedure described in Atzmon et al. [2019], which generalizes signed distance projection through an iterative root-finding procedure $\bar{\mathbf{x}}_{i+1} = \bar{\mathbf{x}}_i - f(\bar{\mathbf{x}}_i) \frac{\nabla f(\bar{\mathbf{x}}_i)}{\|\nabla f(\bar{\mathbf{x}}_i)\|^2}$, $\bar{\mathbf{x}}_0 = \mathbf{x}$. It is worth noting that, in order to have a well-defined \mathbf{n} (and hence well-defined tangent planes), 0 must not be a local extremum of f ; in particular, unsigned distance functions do not work unless they are offset to have a different zero isosurface. Also, unless otherwise stated, we use a smoothed gradient $\bar{\nabla} f(\mathbf{x}) = (\int_{B_{\mathbf{x},\epsilon}} \nabla f(\mathbf{y}) d\mathbf{y}) / |B_{\mathbf{x},\epsilon}|$ instead of $\nabla f(\mathbf{x})$ to define normals, tangent planes, and projections, where $B_{\mathbf{x},\epsilon}$ is the ball of radius ϵ centered at \mathbf{x} and $|B_{\mathbf{x},\epsilon}|$ is its volume, to remove C^1 discontinuities in f . This essentially turns the medial axis into regions of rapidly changing gradients, which becomes useful for detecting large accumulated curvature (Section 3.2).

3.1 Radial Tracing

We wish to find a local parameterization about a point $\mathbf{p} \in \mathcal{M}$, which we denote by $q_{\mathbf{p}}: T_{\mathbf{p}}\mathcal{M} \rightarrow \mathcal{M}$ (we do not write it as $\exp_{\mathbf{p}}$ since, as we will describe later, we do not always want an exact exponential map). To start, we set $q_{\mathbf{p}}(\mathbf{0}) = \mathbf{p}$, and then we numerically integrate m equally spaced geodesic-like radial curves radiating from \mathbf{p} , with initial tangent directions defined extrinsically in \mathbb{R}^3 . Using $T_{\mathbf{p}}\mathcal{M}$ to represent the tangent plane at \mathbf{p} to \mathcal{M} (i.e., the embedding of $T_{\mathbf{p}}\mathcal{M}$ in \mathbb{R}^3), these tangent directions are denoted by $\mathbf{t}_{i,0} \in T_{\mathbf{p}}\mathcal{M}$, $0 \leq i \leq m-1$. We perform a single integration step in direction i by moving along $\mathbf{t}_{i,0}$ for a prescribed step size h , and projecting to \mathcal{M} :

$$\mathbf{q}_{i,1} = \pi(\mathbf{p} + h\mathbf{t}_{i,0}). \quad (1)$$

For small h , this is a good approximation to integrating a short geodesic from \mathbf{p} along the surface of \mathcal{M} , where the exact geodesic path's preimage through π on $T_{\mathbf{p}}\mathcal{M}$ would be a curved path.

To continue integrating the radial curves for more steps, we now need to parallel transport the $\mathbf{t}_{i,0}$'s to their corresponding $\mathbf{q}_{i,1}$ points, or in other words, transfer them from $T_{\mathbf{p}}\mathcal{M}$ to the tangent plane at $\mathbf{q}_{i,1}$, $T_{\mathbf{q}_{i,1}}\mathcal{M}$. We can do this by computing the smallest rotation that transforms $T_{\mathbf{p}}\mathcal{M}$ into $T_{\mathbf{q}_{i,1}}\mathcal{M}$, denoted $R_{i,0}$, and applying it to $\mathbf{t}_{i,0}$: $\mathbf{t}_{i,1} = R_{i,0}\mathbf{t}_{i,0}$. More concretely, $R_{i,0} = \exp[\alpha_{i,0}\hat{\mathbf{a}}_{i,0}]$, where $\alpha_{i,0} = \arccos(\mathbf{n}(\mathbf{p}) \cdot \mathbf{n}(\mathbf{q}_{i,1}))$, $\hat{\mathbf{a}}_{i,0} = \frac{\mathbf{n}(\mathbf{p}) \times \mathbf{n}(\mathbf{q}_{i,1})}{\|\mathbf{n}(\mathbf{p}) \times \mathbf{n}(\mathbf{q}_{i,1})\|}$, and $\exp[\mathbf{v}]$ is the matrix exponential of the skew-symmetric cross product matrix of \mathbf{v} . According to Minding's theorem, $\mathbf{t}_{i,1}$ is precisely the result of parallel transporting $\mathbf{t}_{i,0}$ to $\mathbf{q}_{i,1}$ for constant curvature surfaces, and therefore it is a good approximation for nearby points whose local neighbourhoods are similar [Schmidt et al. 2006]. Using the $\mathbf{t}_{i,1}$'s, we can repeat the integration step described earlier, but where $\mathbf{q}_{i,1}$ replaces \mathbf{p} along each path i . For a general step $j+1$, we therefore

have:

$$\mathbf{q}_{i,j+1} = \pi(\mathbf{q}_{i,j} + h\mathbf{t}_{i,j}). \quad (2)$$

To more explicitly show the correspondence between Eq. 1 and Eq. 2, we can denote \mathbf{p} by $\mathbf{q}_{i,0}$. We repeat for n iterations until we have the set of traced points $Q = \{\mathbf{q}_{i,j} \mid 0 \leq i \leq m-1, 0 \leq j \leq n\}$, which can be arranged into the set of radial curves $\Gamma = \{\gamma_0, \gamma_1, \dots, \gamma_{m-1}\}$ where $\gamma_i = [\mathbf{q}_{i,0}, \dots, \mathbf{q}_{i,n}]$, as well as the set of isolines $\Psi = \{\psi_0, \psi_1, \dots, \psi_n\}$ where $\psi_j = [\mathbf{q}_{0,j}, \mathbf{q}_{1,j}, \dots, \mathbf{q}_{m-1,j}]$ (note that ψ_0 is simply \mathbf{p}). We treat γ_i and ψ_j as both point sequences and curves connecting the points, depending on context.

Pseudocode describing this step, along with the improvements described in Sections 3.2 and 3.3, is given in Alg. 1.

ALGORITHM 1: Radial tracing RADIALTRACE

Inputs : Implicit function f , # of radial curves m , # of steps n , step size h
Outputs : Point samples Q

```

1  $Q \leftarrow \{\mathbf{p}\};$ 
2 for  $j \leftarrow 0, \dots, n-1$  do
3   for  $i \leftarrow 0, \dots, m-1$  do
4     // Section 3.2
5      $\tilde{h} \leftarrow h;$ 
6      $\tilde{\mathbf{q}}_{i,j} \leftarrow \mathbf{q}_{i,j};$ 
7      $\tilde{\mathbf{t}}_{i,j} \leftarrow \mathbf{t}_{i,j};$ 
8     while  $\tilde{h} \geq 10^{-6}$  do
9        $\ell \leftarrow \text{solve Eq. 3 in } [0, \tilde{h}];$ 
10       $\tilde{\mathbf{t}}_{i,j} \leftarrow \tilde{\mathbf{q}}_{i,j} + \ell \tilde{\mathbf{t}}_{i,j};$ 
11       $\tilde{h} \leftarrow \tilde{h} - \|\pi(\tilde{\mathbf{t}}_{i,j}) - \tilde{\mathbf{q}}_{i,j}\|;$ 
12      Transport  $\tilde{\mathbf{t}}_{i,j}$  from  $\tilde{\mathbf{q}}_{i,j}$  to  $\pi(\tilde{\mathbf{t}}_{i,j});$ 
13       $\tilde{\mathbf{q}}_{i,j} \leftarrow \pi(\tilde{\mathbf{t}}_{i,j});$ 
14    end
15  end
16  // Section 3.3
17   $\Theta_{j+1} \leftarrow \text{solve Eq. 5};$ 
18  for  $i \leftarrow 0, \dots, m-1$  do
19     $\mathbf{q}_{i,j+1} \leftarrow \tilde{\mathbf{q}}_{i,j};$ 
20     $\mathbf{t}_{i,j+1} \leftarrow \text{rotate } \tilde{\mathbf{t}}_{i,j} \text{ about } \mathbf{n}(\mathbf{q}_{i,j+1}) \text{ by } \theta_{i,j+1};$ 
21     $Q \leftarrow Q \cup \{\mathbf{q}_{i,j+1}\};$ 
22  end
23 end
24 Return  $Q;$ 

```

3.2 Substepping

Although h is small, projecting a straight line from a general $T_{\mathbf{q}_{i,j}}\mathcal{M}$ onto \mathcal{M} can still behave poorly far away from the plane origin: its projection image can shrink significantly to almost zero length in regions of very high curvature, and can even split near the medial axis (Fig. 3). Since \mathbf{n} is a smoothed normal, it already prevents image splitting, but can instead lead to deformed projection images, which significantly degrade the quality of the final parameterization. Therefore, we sometimes need to take substeps smaller than h to ensure that each full step has a geodesic length of roughly h , which is a technique commonly used when numerically integrating

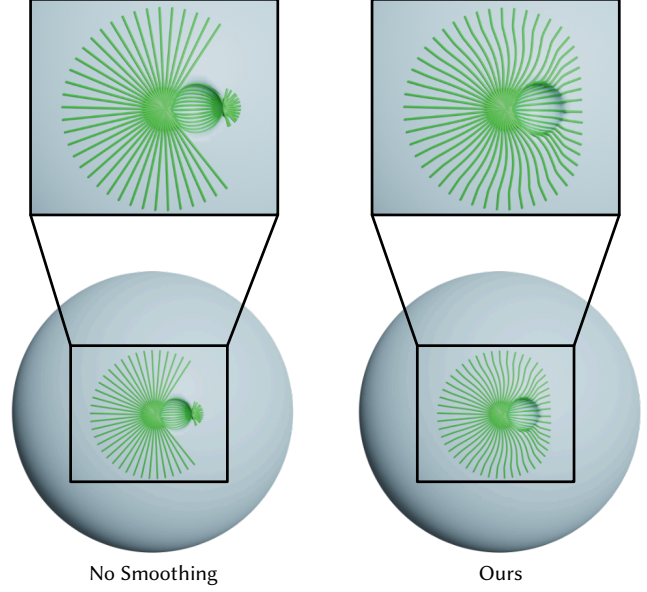


Fig. 4. Without holonomy smoothing (left), the traced paths can drastically separate and intersect with each other away from the origin after they encounter a small pothole. Smoothing out the wedge holonomies (right) causes the paths to more closely emulate their trajectories on a similar surface with constant curvature (in this case, a sphere without a pothole).

differential equations in order to obtain higher accuracy [Dahlquist and Björck 2008].

To achieve a uniform step size across all directions, we find a point along each straight line $\tau_{i,j}(\ell) = \mathbf{q}_{i,j} + \ell \mathbf{t}_{i,j}$ which satisfies the following *projected alignment equation* for each i :

$$\mathbf{n}(\mathbf{q}_{i,j}) \cdot \mathbf{n}(\pi(\tau_{i,j}(\ell))) = s \quad \text{s.t. } \ell \in [0, h], \quad (3)$$

where s is a prescribed alignment cosine (we used $s = \cos \frac{\pi}{4} = \frac{1}{\sqrt{2}}$ in all our experiments). This equation can be solved using a general non-linear root finding procedure such as the bisection method, and if no solution is found, we return the upper bound of the interval, h . Essentially, we want to find the point along $\tau_{i,j}$ where its projected normal significantly deviates from the plane normal, which indicates a large accumulation of curvature and/or having approached a normal discontinuity on \mathcal{M} . To further ensure that we are taking full steps of approximate geodesic length h , we estimate our projected step size for the substep as $\|\pi(\tau_{i,j}(\ell)) - \mathbf{q}_{i,j}\|$, subtract it from h to get a new max step size, and repeat the procedure until the max step size is sufficiently small (see Alg. 1).

3.3 Holonomy Smoothing

Aside from high curvature along a path on the surface, another issue is the variation in curvature between nearby geodesics. The relationship between nearby geodesics is captured by the *Jacobi equation* [Do Carmo 1992; Pottmann et al. 2010], where the acceleration of the separation between geodesics is proportional to the negative Gaussian curvature of the surface. As a result, geodesics accelerate

towards each other on surfaces with positive Gaussian curvature, accelerate away from each other on surfaces with negative Gaussian curvature, and retain their separation velocity on surfaces with zero Gaussian curvature (Fig. 4, left). When acceleration rates vary significantly between traced geodesics, downstream interpolation becomes more challenging, because some regions of the map exhibit significant stretching and poorly represent the underlying surface, while others experience foldovers and lose injectivity (Fig. 8, right). These issues are prominent on surfaces with high frequency details, or even “noisy” surfaces like those from neural implicit.

Since the underlying issue is variance in acceleration, we must simultaneously correct all traced radial curves to give them more uniform acceleration. One possible approach is to adjust the traced points directly, but this would require a difficult constrained optimization over surface point positions. Instead, we use the following simplifications: (1) we perform an optimization after each (full) integration step j , and (2) we optimize over rotations in the tangent plane that, at a high level, “smooth out” the traced paths over the integration front ψ_j . Optimizing rotations in this way allows the projected steps do most of the work while we nudge the paths in better directions as we progress.

We model our optimization on the case of constant acceleration, to encourage the curves to spread out as uniformly as possible. The surfaces satisfying this ideal property are those with constant Gaussian curvature, so the optimization procedure should encourage the curves to behave as if they were on a constant curvature surface. On such surfaces, we expect the total Gaussian curvature in the wedges between curves to be equal, where wedges are defined as $\Delta_{i,j} = [p, \dots, q_{i,j}, q_{i+1,j}, \dots, p]$ (see inset) where each point is connected by a geodesic to the next point in the sequence and i indices are modulo m . Rather than directly integrating the curvature within this loop, which is difficult to do on an implicitly defined surface, we will instead use *holonomy*, which is defined as the amount of rotation induced by parallel transport along a closed loop on the surface, and equivalently the total Gaussian curvature contained in that loop on the surface. This way, we can convert an integration problem into a parallel transport problem, which we are well-equipped to solve. Since we want to find in-plane rotations, we represent the rotations by angles $\theta_{i,j}$ in each $\mathcal{T}_{q_{i,j}}\mathcal{M}$ and solve for these angles. (More precisely, we are now measuring the holonomy of a slightly modified $\Delta_{i,j}$ where infinitesimally small geodesics are extended from each $q_{i,j}$ after rotating each geodesic tangent vector by $\theta_{i,j}$, though with a slight abuse of notation we will refer to both the infinitesimally extended wedge and the original wedge by $\Delta_{i,j}$, as we are exclusively concerned with the infinitesimally extended version.) Denoting the holonomy in a wedge by $R(\Delta_{i,j})$, we smooth out the wedge holonomies by minimizing a Dirichlet-like energy to obtain rotations $\Theta_j = [\theta_{0,j}, \theta_{1,j}, \dots, \theta_{m-1,j}]$:

$$\Theta_j^* = \min_{\Theta_j} \sum_i R(\Delta_{i,j})^2 + \frac{1}{\kappa} \|\Theta_j\|^2. \quad (4)$$

We measure total wedge holonomy by decomposing the wedge loop into strips $S_{i,k} = [q_{i,k-1}, q_{i,k}, q_{i+1,k}, q_{i+1,k-1}, q_{i,k-1}]$ (see inset), where each point is connected by a geodesic just like in the definition of $\Delta_{i,j}$, and then adding the holonomies of these strips. Starting at $q_{i,k-1}$, our goal is to transport $t_{i,k-1}$ around $S_{i,k}$, and to simplify the result, we will set each $t_{p,q}$ to have an angle of 0 in its corresponding tangent space. By construction, transporting along the geodesic from $q_{i,k-1}$ to $q_{i,k}$ does not change the angle, and the same is true from $q_{i+1,k}$ to $q_{i+1,k-1}$, so the main challenge is transporting from $q_{i,k}$ to $q_{i+1,k}$ ($q_{i+1,k-1}$ to $q_{i,k-1}$ is similar). When the distance between $q_{i,k}$ and $q_{i+1,k}$ is small, we can approximately transport vectors from $\mathcal{T}_{q_{i,k}}\mathcal{M}$ to $\mathcal{T}_{q_{i+1,k}}\mathcal{M}$ using the smallest rotation that aligns $T_{q_{i,k}}\mathcal{M}$ with $T_{q_{i+1,k}}\mathcal{M}$, in the same manner by which we transported tangent vectors along short geodesic segments during tracing. Then, we can find the change in angle (induced by a change of basis between tangent spaces) by measuring the angle between the transported $t_{i,k}$ (denoted $\hat{t}_{i,k}$) and $t_{i+1,k}$ in $\mathcal{T}_{q_{i,k}}\mathcal{M}$, which is just the signed angle from $t_{i+1,k}$ to $\hat{t}_{i,k}$ in \mathbb{R}^3 . When $\theta_{i,k} = \theta_{i+1,k} = 0$ (i.e., before smoothing), we denote this angle by $\phi_{i,k}$ (and all such angles at step k by Φ_k), so in the general non-zero case, we have an angle of $\theta_{i,k} + \phi_{i,k} - \theta_{i+1,k}$ (Fig. 5). The change in angle from $q_{i+1,k-1}$ to $q_{i,k-1}$ has a similar expression but negated since we travel in the opposite direction with respect to the indices; thus we have

$$R(S_{i,k}) = (\theta_{i,k} + \phi_{i,k} - \theta_{i+1,k}) - (\theta_{i,k-1} + \phi_{i,k-1} - \theta_{i+1,k-1}). \quad (5)$$

Since we do not wish to rotate the initial tangent vectors $t_{i,0}$ and they are all in $T_p\mathcal{M}$, we have $\theta_{i,0} = 0$ and $\phi_{i,0} = -\frac{2\pi}{m}$.

When we sum up the $R(S_{i,k})$, the $\theta_{i,k} + \phi_{i,k} - \theta_{i+1,k}$ terms cancel out when $1 \leq k < j$, which gives

$$R(\Delta_{i,j}) = \sum_{k=1}^j R(S_{i,k}) = \theta_{i,j} + \phi_{i,j} - \theta_{i+1,j} + \frac{2\pi}{m}. \quad (6)$$

Surprisingly, this expression does not directly depend on the angles obtained from previous steps, but in fact the change in angle terms Φ_j implicitly encode this information, because they are derived from the tangent directions that were obtained from the previous step’s smoothing angles Θ_{j-1} ; in turn, Θ_{j-1} depends on the previous step’s Φ_{j-1} , which depend on Θ_{j-2} , and so on. Unrolling the dependency chain reveals that Φ_j contains information from all previous holonomy smoothing steps.

With the definition of $R(\Delta_{i,j})$, we can now see that Eq. 4 resembles a 1D Dirichlet problem with periodic boundary conditions, which can be solved with a circulant tridiagonal linear system. The linear system contains a null space and can be poorly conditioned, so we also include a regularization term $\frac{1}{\kappa} \|\Theta_j\|^2$ to mitigate these issues, where κ is a tuneable parameter which we typically set to 10^3 in our experiments. Using the Θ_j from solving Eq. 4 to rotate the tangent vectors after each tracing step, we largely eliminate artifacts caused by variations in curvature on the surface (Fig. 4, right).

One potential alternative is to instead smooth the outermost strip holonomies $R(S_{i,j})$ instead of wedge holonomies $R(\Delta_{i,j})$ to more directly smooth out the accelerations at the points on ψ_j ; unfortunately, as we show in Appendix A, this scheme produces undesirably large rotations.

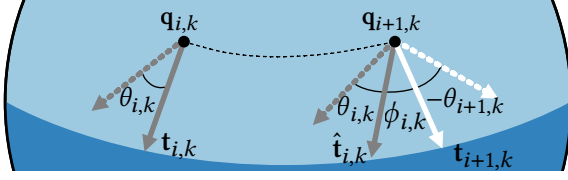


Fig. 5. An illustration of measuring the change in angle as $\mathbf{t}_{i,k}$ is transported from $\mathcal{T}_{q_{i,k}}\mathcal{M}$ to $\mathcal{T}_{q_{i+1,k}}\mathcal{M}$ (represented as $\hat{\mathbf{t}}_{i,k}$ in the latter space). Solid arrows represent tangent vectors before holonomy smoothing, and dashed arrows represent tangent vectors after holonomy smoothing. All angles are measured clockwise.

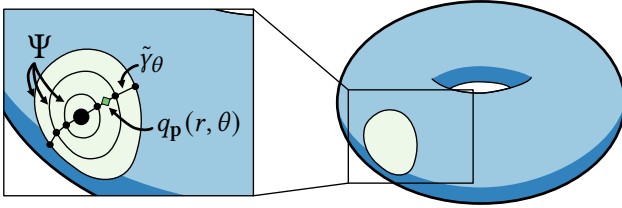


Fig. 6. Using the same torus example from Fig. 2, the map $q_p(r, \theta)$, $(r, \theta) \in \mathcal{T}_p\mathcal{M}$, is evaluated by first evaluating the isolines Ψ at θ and $\theta + \pi$, interpolating those results to form a radial curve $\tilde{\gamma}_\theta$ passing through \mathbf{p} , and evaluating this curve at r .

3.4 Map Interpolation

After tracing out the points, the next step is to interpolate these points to form a continuous map from the planar disc $D_R \subset \mathcal{T}_p\mathcal{M}$ ($R = nh$) to the surface: $q_p: D_R \rightarrow \mathcal{M}$. Each input point $\mathbf{x} = (x, y) \in D_R$ can be expressed in polar coordinates (r, θ) , where $r \in [0, R]$, and from there, we evaluate the surface as a spline in polar coordinates. First, we interpolate the isolines Ψ using periodic cubic splines as functions of angle. We then evaluate all these splines at θ and $\theta + \pi$ (i.e., the opposite angle) to get $2n + 1$ unique points that sample two opposite radial curves connected at the map origin, which we can connect using a natural cubic spline $\tilde{\gamma}_\theta$ and evaluate at r .

3.5 Logarithmic Map

Along with the forward map q_p , it is also useful to have the inverse q_p^{-1} (which we will call the logarithmic map even though q_p is not exactly an exponential map). Finding the exact inverse of the spline surface defined in Section 3.4 would require solving a non-linear optimization problem, so we instead mesh the spline by uniformly sampling several points in D_R , Delaunay triangulating those points, and pushing those points (and their connectivity) through q_p ; with this mesh, we can evaluate q_p^{-1} via closest point queries on the mesh with a small maximum radius (we use 10^{-2}). The point sampling can be improved by, e.g., drawing more samples from high-curvature portions of the map, but at a high enough resolution this basic scheme is sufficient for our applications. For all examples in the paper, we sample 10000 points on the interior of D_R , and use $8m$ uniformly sampled points on ∂D_R for the Delaunay triangulation boundary. Optionally, to obtain a closer fit to the true surface, the spline-evaluated samples can be projected onto the zero isosurface,

though this incurs an additional performance cost proportional to the number of samples, and we did not find it necessary for the results in the paper.

4 EVALUATION AND RESULTS

4.1 Implementation

We implemented our code primarily in Python using the PyTorch library [Paszke et al. 2019] to have a framework for automatic differentiation, which we used to compute gradients of implicit functions and interface with trained neural implicits. We used a simple feature grid architecture for the neural implicits used in the paper, based on the NGLOD architecture [Takikawa et al. 2021] with a single 128^3 feature grid, and an MLP with two 128-unit hidden layers and softplus activations. Point cloud implicits were obtained using a Log-SumExp smooth distance function [Madan and Levin 2022]. Also, we used JIT compilation for analytic implicits and signed distance functions to speed up their evaluation. To have comparable parameters across all experiments, we isotropically rescaled every object to fit within the cube $[-1, 1]^3$ (Table 1).

4.2 Performance

Asymptotically, each component of a single tracing step (projection, substepping, tridiagonal linear system solving) takes $O(m)$ time to advance the integration front from ψ_j to ψ_{j+1} , so tracing n steps is overall $O(mn)$. Fitting the surface also takes $O(mn)$ time to build all of the n isoline splines, and $O(n)$ time to evaluate the surface spline at a point (mainly to build $\tilde{\gamma}_\theta$). One key advantage of our approach is that the performance is *output-sensitive*; in other words, it does not strongly depend on the complexity or size of the underlying surface (though in some cases like triangle mesh distance, each implicit distance query is $O(\log N)$ where N is the number of mesh triangles). For example, mesh-based heat methods [Herholz and Alexa 2019; Sharp et al. 2019] are slightly superlinear in N from solving sparse matrices, but if $mn \ll N$, this is much worse than $O(mn)$. Although the method of Herholz and Alexa [2019] provides fast local solves using only a small output-sensitive subset of vertex values, it still requires a global prefactorization that is superlinear and dependent on N [Herholz et al. 2017]. In Table 2, we show a timing breakdown of the examples in Fig. 1, which are representative of the types of implicit functions we used throughout the paper. All timings were recorded on a 2020 MacBook Pro with an M1 processor. From these results, we see that none of the examples took more than 1.5s to complete, and the primary bottleneck is tracing, particularly root finding in each substep (line 8 of Alg. 1). Furthermore, by also looking at the parameters in Table 1 we observe that the terrain point cloud map took roughly twice as long to trace as the brain mesh map, despite the former having 400 times as many vertices as the latter, which confirms the output-sensitive nature of our algorithm. Meshing the log map takes hundreds of milliseconds for every example, though this is because we evaluate the surface at over 10000 points to build the mesh, so this can easily be reduced if a coarser mesh is acceptable. Furthermore, it is apparent that overall performance is proportional to the amount of time required to evaluate the implicit f and its gradient. Overall, these results are sufficient for interactive applications, even on the LiDAR terrain

Table 1. Parameter listings of all the results in the paper, where m is the number of radial curves, n is the number of steps, and h is the step size. The “Size” column lists the number of points in a point cloud, or the number of vertices in a mesh.

Surface	Fig.	Type	Size	m	n	h	Smoothing Enabled?
Saddle	10	Analytic Implicit	—	50	10	1.0×10^{-2}	Yes
Armadillo	8	Neural Implicit	—	50	20	5.0×10^{-3}	Yes+No
Cone	14	SDF	—	50	12	1.0×10^{-2}	No
Bunny (coarse)	12 (top right)	Mesh	3,485	20	20	2.5×10^{-2}	Yes
Bunny (fine)	12 (bottom right)	Mesh	13,934	20	20	2.5×10^{-2}	Yes
Toy	1	SDF	—	50	30	1.0×10^{-2}	Yes
Flute	1	Analytic Implicit	—	50	12	5.0×10^{-3}	Yes
Brain	1	Mesh	73,820	50	10, 15, 20	5.0×10^{-3}	Yes
Owl	1	Neural Implicit	—	50	5, 10	1.0×10^{-2}	Yes
Terrain	1	Point Cloud	29,474,037	50	10	1.0×10^{-2}	Yes
Star Torus	15	SDF	—	50	40	1.0×10^{-2}	Yes
Shellfish	13 (left)	Neural Implicit	—	50	15	1.0×10^{-2}	Yes
Einstein	13 (middle)	Neural Implicit	—	50	20	5.0×10^{-3}	Yes
Metatron	13 (right)	Neural Implicit	—	50	15	1.0×10^{-2}	Yes
Pothole	4	SDF	—	50	20	1.0×10^{-2}	Yes+No
Spot	9	Mesh	2,930	50	20	1.0×10^{-2}	Yes
Shark	16	Mesh	10,054	50	20	1.0×10^{-2}	Yes

Table 2. Timing breakdown for maps generated on the surfaces in Fig. 1, with an in-depth breakdown for radial tracing. Times are given in *milliseconds*. The root finding (“Root”), projected step (“Proj.”), and frame rotation (“Frame Rot.”) operations are done during each tracing substep, and smoothing is done after a full tracing step. Times are averaged over all maps on each surface, except for the brain and owl surfaces, where timings for the largest map are reported instead.

Surface	Type	n	Avg. # Substeps	Avg. Root	Avg. Proj	Avg. Frame Rot.	Avg. Smoothing	Tracing	Fitting	Meshing
Toy	SDF	30	2.000	5.287	0.5509	0.5982	0.6699	414.3	2.336	196.0
Flute	Analytic Implicit	12	2.417	10.85	2.219	1.304	0.7731	427.3	2.006	165.1
Brain	Mesh	20	2.900	14.66	0.8845	1.419	0.6537	502.5	2.005	158.3
Owl	Neural Implicit	10	2.800	14.75	1.089	1.294	0.7760	492.4	2.058	172.0
Terrain	Point Cloud	10	2.733	34.16	3.403	3.069	0.7709	1137	1.977	165.1

point cloud with over 29 million points, and suggest that speeding up the computation of f and a faster root finding method can further improve performance.

4.3 Parameter Analysis

The parameter m (number of traced radial curves), n (number of steps taken), and h (step size) are the main quantities used to affect the accuracy of the parameterizations. To demonstrate the effect of these parameters on a simple case with a known analytic solution, we ran our method on a unit sphere (with no substepping and no holonomy smoothing) using a range of values for each of these parameters (where nh , the total radius of the map in tangent space, is always 1) and computed the average L^2 error of our output from the analytic exponential map, at 2000 samples inside the parameterized disc in tangent space and 500 points along the boundary of the disc (Fig. 7). Overall, we observe that both adding more steps (and correspondingly smaller step sizes) and more radial curves can improve the results, but after enough steps/radial curves, adding more steps/radial curves does not seem to improve the results. In particular, increasing m from 5 to 50 significantly reduces the error,

though there is little difference between using 50 or 500 radial curves; as such, we use $m = 50$ for most of the results in the paper. Furthermore, we see that the error levels out under refinement of h , but at different points when using 5 radial curves (10 steps) and 50/500 radial curves (100 steps). To further investigate the source of the error, we also plotted the error from our traced points along a single radial curve for each step size (since this is on a sphere, there is little difference between each individual radial curve). This error curve follows a very similar trend to the larger m error curves which decrease and eventually level out around 10^{-5} , which suggests that once there are sufficiently many traced radial curves, the primary source of error is the tracing rather than the spline interpolation. Since the error in the traced points accumulates over more steps, there is a tradeoff between accumulating larger per-step error over fewer steps, and accumulating smaller per-step error over more steps, which causes the error to level out under refinement. Despite the lack of error reduction beyond 10^{-5} , this is quite small, and as demonstrated in the examples throughout the paper, does not cause practical issues.

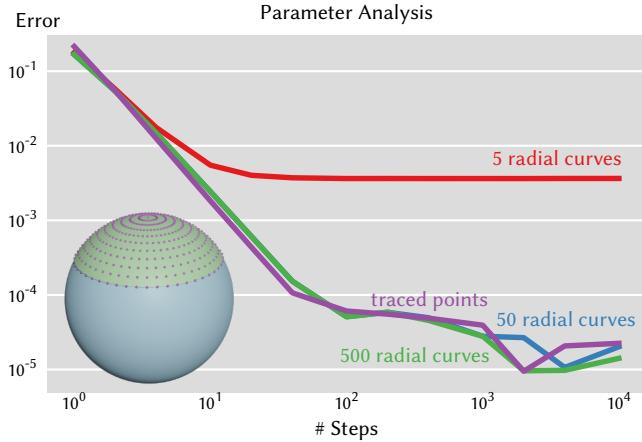


Fig. 7. On an exponential map of the sphere, the mean error of our method compared to the analytical solution generally decreases as we increase the number of steps/decrease step size, but this eventually levels out for maps with more steps. More radial curves can reduce the error, but does not provide any additional benefit between 50 and 500 radial curves. The error along the traced radial points closely matches the overall map error.

Holonomy smoothing strongly influences the output, and is controlled by a regularization parameter κ . In Fig. 8, we examine the effect of κ ; as it decreases, the effect of the smoothing is diminished, and in the limit as $\kappa \rightarrow 0$, the smoothing is completely skipped. We built a local parameterization for a neural implicit representing the Stanford armadillo, around a point near a small indent in the forearm created by isosurface noise. With large κ (i.e., strong smoothing), we see that the map has low distortion away from the origin, and although there is some distortion near the pothole itself, this distortion largely disappears as the map expands. Meanwhile, as κ decreases, the map’s quality degrades; some regions fold over each other (seen through colormap discontinuities), while other regions significantly stretch away from the origin. As a result, we exclusively use $\kappa = 10^3$ everywhere else in the paper. However, in large map regions, holonomy smoothing can produce an undesirable shearing effect on the radial curves, where some curves end up becoming nearly parallel, an effect which primarily happens around large surface protrusions (Fig. 9).

Another parameter that can be controlled is the smoothing radius ϵ , used for computing smoothed gradients $\nabla f(\mathbf{x})$ in the ball $B_{\mathbf{x},\epsilon}$. We used a Monte Carlo estimate of this smoothed gradient, by setting $\epsilon = 10^{-4}$ and averaging 10 samples from $B_{\mathbf{x},\epsilon}$. The variance of this estimate can affect the substepping procedure and produce different results across executions with the same input parameters, but we did not experience this in any of our experiments. Nevertheless, we note that the accuracy of this estimate can be increased by adding more samples and/or reducing the smoothing radius.

4.4 Comparisons

Compared to prior local parameterization methods, our method can extract significantly more information out of implicit surfaces.

Rather than simply sampling the surface for points, and using nearest neighbor information to, e.g., build graphs or Laplacians, we generate the samples we need through tracing, while also being able to evaluate the generated map on other surface points. However, our method does not provide precise control over the size of the resulting parameterization beyond integration parameters, which makes it difficult to directly compare to “log map-based” methods in prior work which require such information as input. That said, our generated maps can still be evaluated on isolated surface samples after the maps have been expanded from a given surface origin, which forms the basis of our comparison methodology against prior work. More specifically, to compare our method to prior work, we generated a local parameterization around a saddle point with 500 traced samples, and rejection sampled the same number of surface points from a 3-dimensional ball around the map origin in \mathbb{R}^3 such that all the points are contained in our map. Then, we evaluated the log map of our method on those sample points, and also provided those points as input to both versions of discrete exponential maps (DEM) [Schmidt 2013; Schmidt et al. 2006] and the vector heat method [Sharp et al. 2019]. We used our own implementation for the DEM methods and the authors’ released code for the vector heat method on point clouds¹, which uses intrinsic Delaunay triangulations to build more numerically robust Laplacians. For each map, we built a mesh connecting the samples by using the connectivity from the Delaunay triangulation computed in tangent space. As seen in Fig. 10, all other methods produce high distortion away from the origin, and the vector heat method struggles throughout the entire map. It is worth noting that these struggles are exacerbated by the naïve point sampling method, which tends to produce clusters of points that can be prevented by more sophisticated blue noise sampling techniques [De Goes et al. 2012; Witkin and Heckbert 1994]; nevertheless, our method produces low distortion throughout the map despite the sample quality, since we do not depend on them for generating the parameterization.

We also used the same point surface point \rightarrow log map setup to quantitatively investigate the distortion of our method’s output compared to prior work. More specifically, we used a dataset of over 100 surfaces collected by Myles et al. [2014], and for each surface sampled 10 surface points as map origins, to obtain a set of over 1000 surface regions over which to build local parameterizations. For half of these local surface regions, we sampled 500 surface points within a geodesic radius of 0.1 of the origin, and for the other half we sampled 1000 points within a geodesic radius of 0.2. These points were then evaluated through the log map of each method used in Fig. 10 and triangulated in tangent space to obtain connectivity for the sample points in \mathbb{R}^3 . We used the area-weighted average symmetric Dirichlet energy [Smith and Schaefer 2015] and least-squares conformal map (LSCM) energy [Lévy et al. 2002] to measure the degree to which the resulting parameterized meshes represent isometries and conformal maps, respectively.

The results of this experiment are provided as histograms in Fig. 11. Across all methods, the symmetric Dirichlet distributions look quite similar, with the majority of maps exhibiting low distortion near the global minimum of 4, and a tail of maps with higher

¹<https://github.com/nmwsharp/potpourri3d>

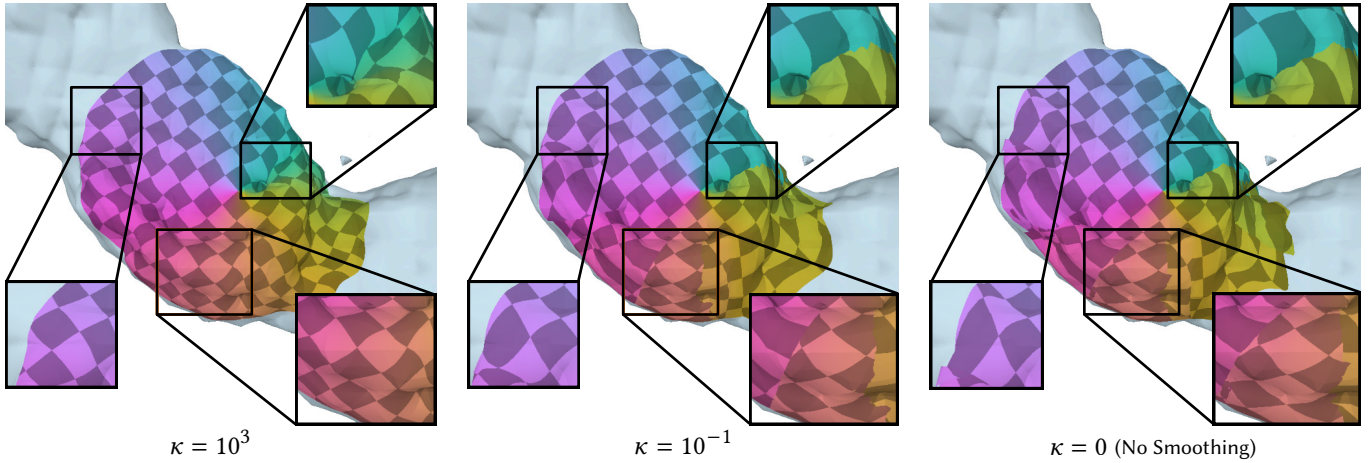


Fig. 8. As we decrease κ , the effect of the smoothing is diminished. Near an indent in this neural implicit armadillo’s arm (top right inset), decreasing κ exchanges distortion near the indent (which diminishes away from the indent) for foldovers in the map. Furthermore, decreasing κ can also affect regions farther away from the problematic indent, by inducing stretching (bottom left inset) and foldovers (bottom right inset) near the boundary of the map.

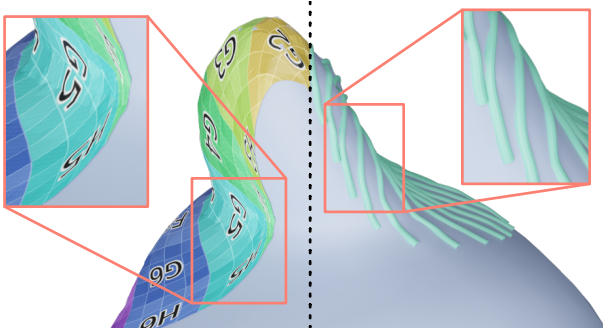


Fig. 9. In the presence of particularly large surface features, holonomy smoothing produces large rotations and shears the integration front, causing curves to be near-parallel and overlap; the resulting map is also distorted. On this mesh of the log map, the symmetric Dirichlet energy [Smith and Schaefer 2015] is 6.52, and the least squares conformal map (LSCM) energy [Lévy et al. 2002] is 0.987, both higher than the averages shown in Fig. 11.

distortion. However, our method (right) produces the largest fraction of low-distortion maps compared to the other tested methods, with over 900 of the maps contained in the lowest histogram bin, and as a result has a much “thinner” tail than the other methods. The DEM methods in particular (left, middle left) have nearly half of their outputs outside of the lowest bin. On the other hand, the LSCM distributions vary significantly across the different methods: DEM methods exhibit a mode near the right side of the distribution near 1, the vector heat method (middle right) has a much narrower distribution but with a lower mode near 10^{-1} , and our method has a much wider variance but a mode of 10^{-3} , orders of magnitude lower than the other methods. We can see one such example of a high LSCM energy in Fig. 9, where the map has an LSCM energy of nearly 1. Overall, we see that the combination of geodesic tracing,

substepping, and holonomy smoothing produces near-isometric and near-conformal maps in most cases, which quantitatively affirms Fig. 10, but can produce non-conformal results when holonomy smoothing shears regions with large variations in curvature, as seen in Fig. 9.

Since we trace points along the surface, another interesting property of our method on triangle meshes in particular is that our result quality is independent of the underlying triangulation. In Fig. 12 we compare our method to both DEM methods, the vector heat method, and another heat-based method [Herholz and Alexa 2019], for which we used our own implementation, taking care to use intrinsic Delaunay operators from LIBIGL [Jacobson et al. 2018] (though our implementation does not use the accelerated solve, since a public implementation of the specialized prefactorization data structure is unavailable). For this experiment, we computed the 10-ring of an origin vertex (i.e., all vertices at most 10 edges away from the origin) on a coarse Stanford bunny mesh with 3500 vertices, and used these as the evaluation points for all methods. The DEM methods only used these points and their derived connectivity from the mesh, while we used the entire mesh to define the required operators for the heat methods (since we would change the boundary conditions if we only used the local patch). Our method does not build a map from pre-selected discrete points (aside from the map origin), so to make a best-effort comparison we ran our method so that the resulting map both contained the same 10-ring and used a comparable point budget (400 traced samples vs. 453 vertices in the 10-ring — see Table 1 for the parameters we used). The original mesh connectivity was used to connect the 10-ring evaluation points into a continuous patch for all methods. We see that our method produces by far the least distortion in the map, and just like in Fig. 10, the DEM methods exhibit distortion far away from the origin, while the heat-based methods struggle throughout the map, since the triangulation is not of a high enough quality to produce well-conditioned operators. After one level of Loop subdivision (which produces a mesh

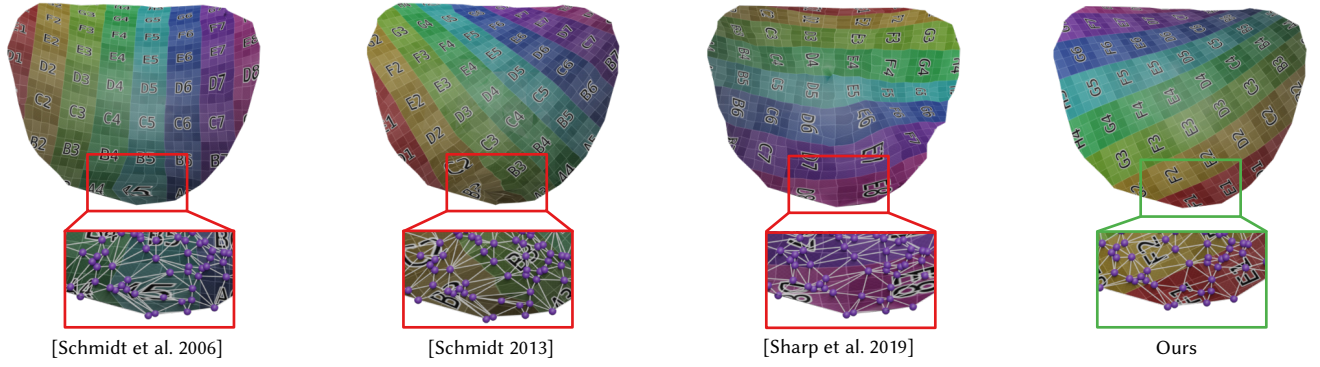


Fig. 10. On a point set rejection sampled from a saddle surface, the original discrete exp map [Schmidt et al. 2006] (left) exhibits shearing artifacts away from the origin in the center. The more recent version [Schmidt 2013] (middle left) with more accurate geodesics exhibits even more severe shearing. The vector heat method [Sharp et al. 2019] (middle right) is strongly influenced by the non-uniform point distribution and has distortion throughout the map. Our method (right) is the only one to exhibit low distortion everywhere, due to our lack of dependence on the sample points. The maps represent point sets connected via tangent space Delaunay triangulation, and the insets show the input samples and their Delaunay edges.

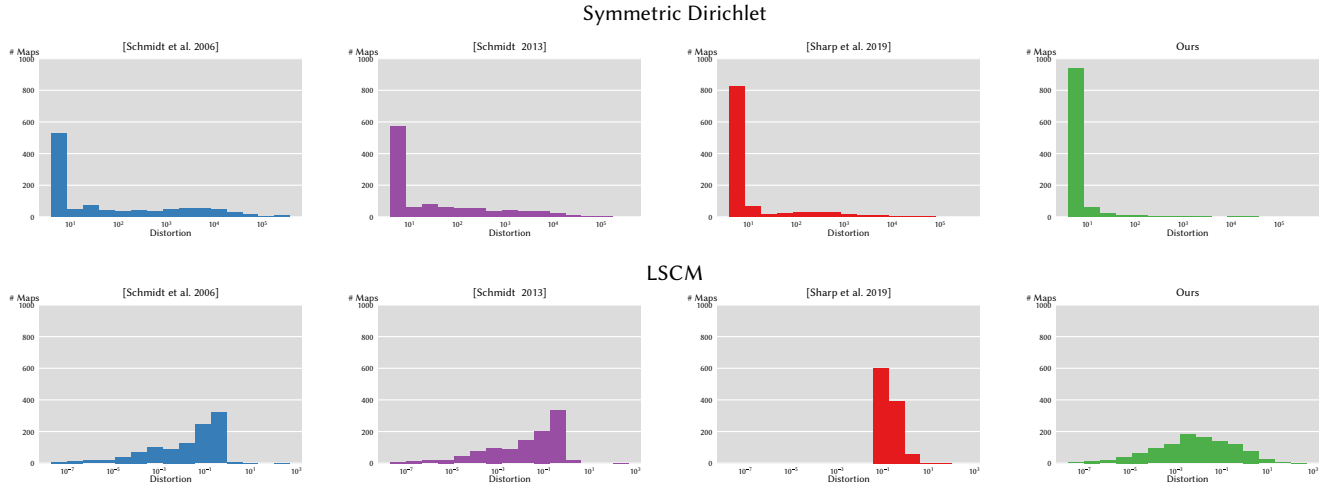


Fig. 11. Across a dataset of surfaces, map positions, and map sizes, almost all of the maps produced by our method have near-minimum symmetric Dirichlet energy, whereas a significant fraction of the maps produced by other methods have higher energy. (top). Our method also produces maps with orders-of-magnitude lower median LSCM energy compared to other methods, though with more variance (bottom). All histograms are plotted on a log scale on the x-axis.

with roughly 14000 vertices), we ran the same experiment on the 20-ring to cover the same region on the surface (now containing 1720 vertices), and did not change the parameters in our method (though we should clarify that the underlying surface is now the *subdivided* mesh and not the original mesh). Now we see that our method produces virtually the same result as before, the heat-based methods perform much better with very little distortion, and the DEM methods improve but still exhibit artifacts at the boundary of the map. However, these other methods only achieve this quality with 4 times the number of points that we need to achieve similar quality, and cannot achieve this at all on a coarser mesh.

4.5 Applications

4.5.1 Local Texturing. A variety of examples of texturing different representation types is provided in Fig. 1. We show several decals on each surface, where each decal is represented by a meshed version of the spline surface, and can see that none of the resulting images are distorted, while still conforming to the original surface. The terrain point cloud, owl neural implicit, and brain mesh all have challenging geometry with surface bumps, ridges, and divets that make parameterization more difficult. By placing maps in small regions, however, even regions like the owl's plumage and the heavily folded surface of the brain can be textured.

To further illustrate map quality on highly detailed surfaces and high-genus surfaces, we show some more maps on neural implicit

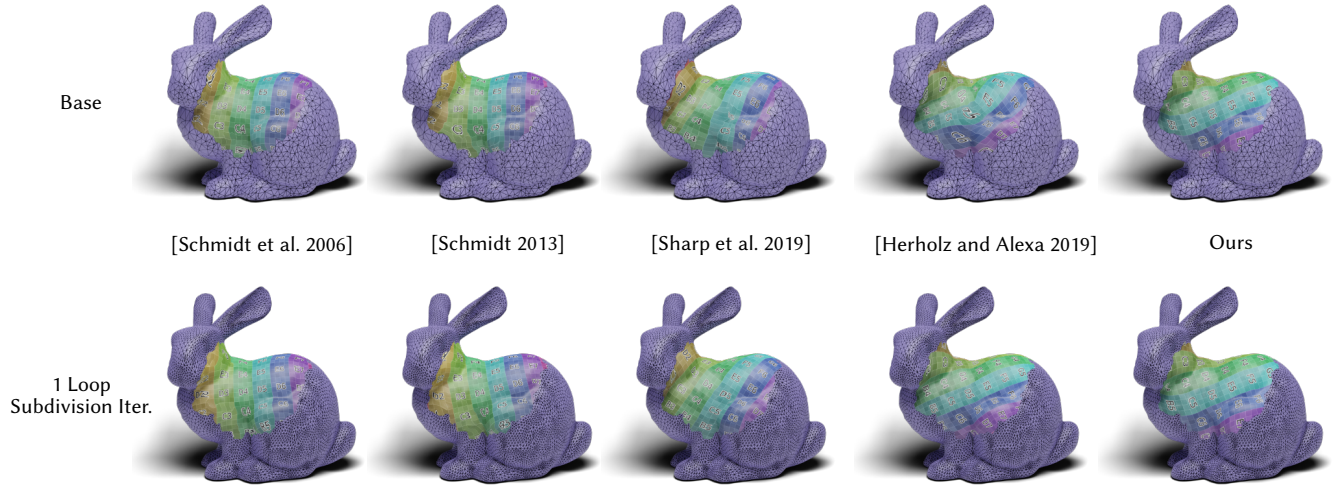


Fig. 12. On a coarse bunny mesh (top), a local parameterization generated by our method (right) avoids the severe artifacts near the boundary of the map (left, centre left) and angular distortion near the origin (centre, centre right) seen by other methods. All methods improve after one level of Loop subdivision (bottom), but the DEM-based methods still possess boundary artifacts (left, centre left). Please zoom in to see map details.

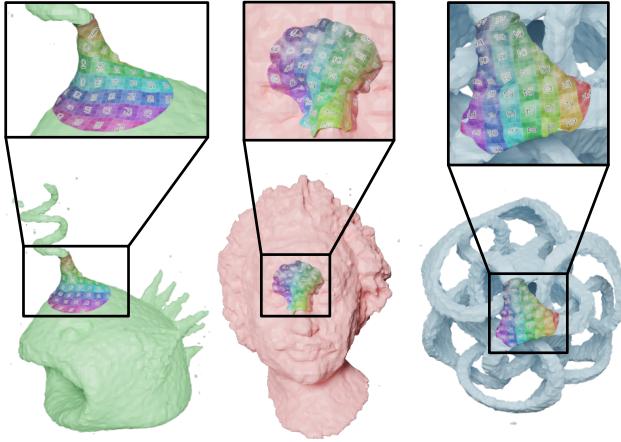


Fig. 13. We obtain high-quality maps on a variety of neural implicit surfaces (rendered as meshes reconstructed with marching cubes), even in the presence of noisy reconstruction artifacts and fine ground truth details. The local maps reveal surface bumps and folds that the marching cubes-reconstructed surface mesh does not capture at its configured resolution.

in Fig. 13. All three surfaces are bumpy from reconstruction error, but due to holonomy smoothing, our method is largely able to ignore the effects of both reconstruction noise and ground truth high-frequency features (e.g., the bridge of the Einstein bust’s nose) on the radial path trajectories, and produce low-distortion maps that still conform to the underlying surfaces.

4.5.2 Multi-Valued Logarithmic Maps. Unlike other methods, which compute maps by assigning texture coordinates to surface points, our method operates in the opposite direction and instead traces

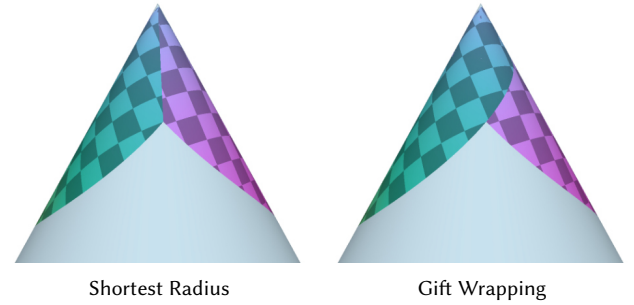


Fig. 14. Although local geodesics can fold over, by computing the map from tangent space rather than from the embedding space, we can obtain a multi-valued log map, which we can use to find the globally shortest paths (left) and “gift wrap” the map around a cone tip (right).

out surface paths corresponding to radial lines in tangent space. This allows us to support multi-valued logarithmic maps as well, which can encode all the local geodesics that pass through a point on the surface (and smoothing could be disabled in such cases). As a proof of concept, we show some preliminary visualization results that take advantage of this property, by compositing multi-valued log map values in a sphere tracer. Compositing is done by taking all ray intersections with the log map and sorting the intersections (near the closest ray intersection to the underlying surface) based on user specification. The shortest radius mode reproduces global geodesics by taking the point with the smallest radius (reproducing the seam we would expect with global geodesics). Alternatively, we can “gift wrap” the map by taking the value with the smallest phase, which, to our knowledge, is not an effect that can be produced by other methods (Fig. 14).

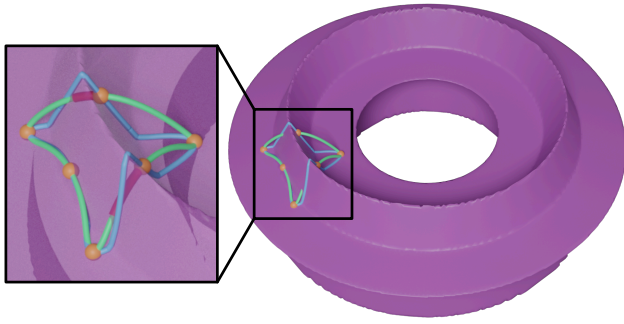


Fig. 15. Using local parameterizations, we can draw κ -curves directly on an isosurface (blue) by placing constraint points on the surface (orange) and optimizing control points in tangent coordinates. Solving in the ambient space with the same constraint points leads to intersections with the surface (green). The inset shows a translucent view of the curves, where the regions of the ambient curve that cut inside the surface are shown in red.

4.5.3 Curve Drawing on Surfaces. It is also useful to be able to “connect” nearby parameterizations by composing the inverse of one with the forward evaluation of another (i.e., $q_{p_2} \circ q_{p_1}^{-1}$), to convert surface points between different local coordinate systems, which we illustrate by lifting the 2D κ -curves [Yan et al. 2017] algorithm onto surfaces using our method. With local parameterizations we can solve for the curve’s control points in tangent coordinates, and evaluate each curve segment in a separate local map, enforcing that all points lie on the surface by construction. For details on how this is done, please see Appendix B. We show our results in Fig. 15; the curve produced with our method exactly conforms to the surface, while attempting the same process directly in \mathbb{R}^3 results in the curve cutting through the surface.

5 LIMITATIONS, FUTURE WORK AND CONCLUSION

The primary technical limitation of our method is that our maps are restricted to discs in tangent space, which also restricts their maximum radii. Unlike “log map-based” methods on meshes [Herholz and Alexa 2019; Sharp et al. 2019], which assign tangent coordinates to each vertex and indirectly create a star around the origin in tangent space, our method restricts the geometry of the domain and thus cannot expand into large maps across complex surfaces without intersecting or rapidly separating (adjacent) geodesics, which lead to a poorly interpolated spline surface. Even holonomy smoothing cannot entirely ameliorate this issue, particularly around large protrusions on the surface, because it will try to emulate a surface of constant Gaussian curvature that will eventually cause adjacent geodesics to run nearly parallel with each other (Fig. 9). However, even prior work such as the vector heat method struggles in these challenging scenarios; a challenging example from Fig. 11 is shown in Fig. 16 where both our method and the vector heat method produce distorted results. The parallel transport assumption along the current integration front ψ_j used in holonomy smoothing can also degrade in accuracy as the geodesics separate over several tracing iterations, though this can be remedied by tracing more geodesics to reduce the inter-geodesic separation distances. Thus, a method that

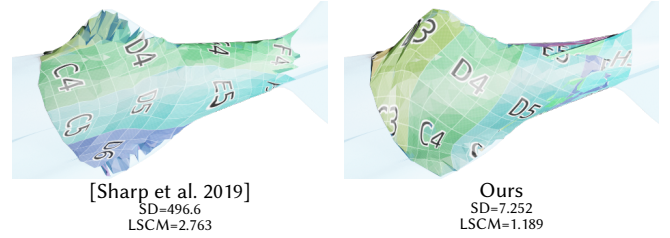


Fig. 16. A challenging example from the dataset of Myles et al. [2014] where our method’s log map triangulated from surface samples exhibits high distortion (right), with shearing near the shark’s dorsal fin, and wrapping around itself near the tail (unlike Fig. 14 which used a custom renderer, the Blender renderer used here does not support map-aware compositing). However, when the vector heat method [Sharp et al. 2019] (left) is applied to the same set of sample points, it also struggles to reproduce the dorsal fin, though it produces an injective map. The two distortion metrics from Fig. 11 are given as well, where “SD” represents symmetric Dirichlet energy, and “LSCM” represents LSCM energy.

can separate varying angular density between radial geodesics, varying termination steps for each geodesic, and an interpolation method that can accurately stitch such geodesics together, is required for a fully robust “exp map-based” solution to the local parameterization problem, though our much simpler approach already produces excellent results on many complex surfaces and previously unsupported geometric representations.

There are many more applications of local parameterizations beyond the ones we showed in this paper. For example, conducting simulation algorithms on surfaces, such as fluid simulation, particle dynamics, and even algorithms for more esoteric phenomena like ice crystal growth [Kim and Lin 2003], are a few examples that can be more easily unlocked by high-quality local parameterizations. Another application is in running generative models such as diffusion models [Song et al. 2020] and normalizing flows [Kobyzev et al. 2020] on surfaces. Although there has been some work on generalizing the latter to surfaces [Lou et al. 2020], it is restricted to surfaces where analytic exponential maps are known, which greatly limits its applicability. More broadly in computer graphics, surface-based algorithms are often relegated to cases where the surface is explicitly provided, such as with triangle meshes, but with the emergence of new neural geometric representations like neural implicits, neural radiance fields [Mildenhall et al. 2021], and Gaussian splats [Kerbl et al. 2023], their adoption in wider applications is limited by the need to design tailored algorithms for each representation. We therefore believe that general-purpose algorithms like ours are an important step towards making more geometry widely usable.

ACKNOWLEDGMENTS

This project was funded in part by an NSERC Discovery Grant (RGPIN-2023-05120) and an Ontario Early Researchers Award. The first author was funded by an NSERC Canada Graduate Scholarship – Doctoral. We thank Victor Rong, Honglin Chen, Silvia Sellán, and Towaki Takikawa for proofreading.

REFERENCES

- Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. 2019. Controlling neural level sets. *Advances in Neural Information Processing Systems* 32 (2019).
- Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.
- Mario Botsch and Leif Kobbelt. 2004. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 185–192.
- Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. 2022. Neural dual contouring. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–13.
- Zhiqin Chen and Hao Zhang. 2021. Neural marching cubes. *ACM Trans. Graph.* 40, 6, Article 251 (dec 2021), 15 pages. <https://doi.org/10.1145/3478513.3480518>
- Keenan Crane, Mathieu Desbrun, and Peter Schröder. 2010. Trivial connections on discrete surfaces. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1525–1533.
- Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. 2020. A survey of algorithms for geodesic paths and distances. *arXiv preprint arXiv:2007.10430* (2020).
- Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2017. The heat method for distance computation. *Commun. ACM* 60, 11 (2017), 90–99.
- Germund Dahlquist and Åke Björck. 2008. *Numerical methods in scientific computing, volume I*. SIAM.
- Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–11.
- Manfredo Perdigão Do Carmo. 1992. *Riemannian geometry*. Vol. 2. Springer.
- Michal Edelstein, Nestor Guillen, Justin Solomon, and Mirela Ben-Chen. 2023. A Convex Optimization Framework for Regularized Geodesic Distances. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–11.
- Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 216–224.
- Philipp Herholz and Marc Alexa. 2019. Efficient computation of smoothed exponential maps. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 79–90.
- Philipp Herholz, Timothy A. Davis, and Marc Alexa. 2017. Localized solutions of sparse linear systems for geometry processing. *ACM Transaction on Graphics (Proc. of Siggraph Asia)* 36, 6 (2017), 8. <https://doi.org/10.1145/3130800.3130849>
- Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.
- Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. <https://libigl.github.io/>.
- Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.
- Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* 42, 4 (2023), 1–14.
- Theodore Kim and Ming C Lin. 2003. Visual simulation of ice crystal growth. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Citeseer, 86–97.
- Ron Kimmel. 1998. Fast marching methods on triangulated domains. *Proc. Nat. Acad. Sci.* 95 (1998), 8431–8435.
- Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Globally optimal direction fields. *ACM Transactions on Graphics (ToG)* 32, 4 (2013), 1–10.
- Ivan Kobzyev, Simon JD Prince, and Marcus A Brubaker. 2020. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence* 43, 11 (2020), 3964–3979.
- Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Transactions on Graphics* 21, 3 (2002), 10–p.
- William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (1987), 163–169.
- Marco Lorenzi and Xavier Pennec. 2014. Efficient parallel transport of deformations in time series of images: from Schild’s to pole ladder. *Journal of mathematical imaging and vision* 50 (2014), 5–17.
- Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser Nam Lim, and Christopher M De Sa. 2020. Neural manifold ordinary differential equations. *Advances in Neural Information Processing Systems* 33 (2020), 17548–17558.
- Abhishek Madan and David IW Levin. 2022. Fast Evaluation of Smooth Distance Constraints on Co-Dimensional Geometry. *ACM Trans. Graph.* 41, 4 (2022).
- Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*. 37–45.
- Eivind Lyché Melvær and Martin Reimers. 2012. Geodesic polar coordinates on polygonal meshes. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 2423–2435.
- 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.
- Patrick Mullen, Yiyi Tong, Pierre Alliez, and Mathieu Desbrun. 2008. Spectral conformal parameterization. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1487–1494.
- Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)* 41, 4 (2022), 1–15.
- Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-aligned Global Parameterization. , Article No. 135 pages. <http://vcg.isti.cnr.it/Publications/2014/MPZ14>
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
- Hans Kōhling Pedersen. 1995. Decorating implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 291–300.
- Konrad Polthier and Markus Schmies. 2006. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH 2006 Courses* (Boston, Massachusetts) (*SIGGRAPH ’06*). Association for Computing Machinery, New York, NY, USA, 30–38. <https://doi.org/10.1145/1185657.1185664>
- Helmut Pottmann, Qixing Huang, Bailin Deng, Alexander Schiftner, Martin Kilian, Leonidas Guibas, and Johannes Wallner. 2010. Geodesic patterns. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 1–10.
- Ryan Schmidt. 2013. Stroke parameterization. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 255–263.
- Ryan Schmidt, Cindy Grimm, and Brian Wyvill. 2006. Interactive decal compositing with discrete exponential maps. In *ACM SIGGRAPH 2006 Papers*. 605–613.
- Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. Article 73, 11 pages.
- Nicholas Sharp and Keenan Crane. 2020. A laplacian for nonmanifold triangle meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 69–80.
- Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019. The vector heat method. *ACM Transactions on Graphics (TOG)* 38, 3 (2019), 1–19.
- Jason Smith and Scott Schaefer. 2015. Bijective parameterization with free boundaries. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–9.
- Yousuf Soliman, Dejan Slepčev, and Keenan Crane. 2018. Optimal Cone Singularities for Conformal Flattening. *ACM Trans. Graph.* 37, 4 (2018).
- Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456* (2020).
- Pratul P. Srinivasan, Stephan J. Garbin, Dor Verbin, Jonathan T. Barron, and Ben Mildenhall. 2023. Nuvo: Neural UV Mapping for Unruly 3D Representations. *arXiv* (2023).
- Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.* 24, 3 (July 2005), 553–560. <https://doi.org/10.1145/1073204.1073228>
- Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11358–11367.
- Francis Williams, Teso Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. 2019. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10130–10139.
- Andrew P Witkin and Paul S Heckbert. 1994. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 269–277.
- Zhipei Yan, Stephen Schiller, Gregg Wilensky, Nathan Carr, and Scott Schaefer. 2017. κ -curves: Interpolation at local maximum curvature. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–7.

A INSTABILITY OF STRIP HOLONOMY

Here we show that smoothing the holonomy of the outermost strips $S_{i,j}$ is unstable, or more precisely, we show that strip holonomy smoothing produces undesirably large rotation angles. The strip holonomy is given by

$$R(S_{i,j}) = (\theta_{i,j} + \phi_{i,j} - \theta_{i+1,j}) - (\theta_{i,j-1} + \phi_{i,j-1} - \theta_{i+1,j-1}), \quad (7)$$

and the smoothing optimization is

$$\Theta_j^* = \min_{\Theta_j} \sum_i R(S_{i,j})^2 + \frac{1}{\kappa} \|\Theta_j\|^2. \quad (8)$$

The solution to Eq. 8 is given by the linear system

$$\left(L + \frac{1}{\kappa} I\right) \Theta_j = \Phi_j - \Phi_{j-1} - L\Theta_{j-1}, \quad (9)$$

where L is the 1D (positive-semidefinite) Laplacian, I is the identity matrix, and Φ_j is defined coordinate-wise by $(\Phi_j)_i = (\phi_{i-1,j} - \phi_{i,j})$. However, we typically want $\frac{1}{\kappa}$ to be small for smoothing to be effective, so we will drop the $\frac{1}{\kappa} I$ term. Since $\Phi_0 = \Theta_0 = \mathbf{0}$, we have $L\Theta_1 = \Phi_1$, and then $L\Theta_2 = \Phi_2 - 2\Phi_1$, $L\Theta_3 = \Phi_3 - (2\Phi_2 - 2\Phi_1)$, and in general

$$L\Theta_j = \Phi_j - 2 \sum_{k=1}^{j-1} (-1)^{j-k-1} \Phi_k. \quad (10)$$

Since L^{-1} has eigenvalues $\gg 1$, and the upper bound of the right-hand side of Eq. 10, $\|\Phi_j\| + 2 \sum_{k=1}^{j-1} \|\Phi_k\|$, grows in magnitude as j increases, the upper bound of $\|\Theta_j\|$ also grows in magnitude as j increases. Although this is not a tight bound, from Eq. 6, we can interpret Φ_j as the difference in wedge holonomy between adjacent $\Delta_{i,j}$ before smoothing, and since the total (absolute) curvature contained in $\bigcup_i \Delta_{i,j}$ increases as j increases, we can expect adjacent wedge holonomies to diverge and the norm of holonomy differences to increase with j , with the exception of constant curvature surfaces where $(\Phi_j)_i = 0$. As such, $\|\Theta_j\|$ generally increases with j , which matches what we observed in practice, and thus strip holonomy smoothing is unsuitable for a procedure that is primarily intended to make small adjustments to radial paths.

In contrast, the solution to wedge holonomy smoothing in Eq. 4 drops the alternating summation:

$$\left(L + \frac{1}{\kappa} I\right) \Theta_j = \Phi_j. \quad (11)$$

As such, its behaviour is much more benign and only depends on Φ_j rather than an alternating sum of previous Φ_k .

B CURVE DRAWING ON SURFACES

To elaborate on our κ -curves implementation on surfaces, we replaced the global step of the κ -curves local-global solver with 10 iterations of a Gauss-Seidel solver, where every iteration solves for the c_1 control points in sequence (using the notation from Yan et al. [2017]). Each c_1 is associated with a local map and its corresponding origin \mathbf{p}_k , so they are initialized to be $\mathbf{0}$ in $\mathcal{T}_{\mathbf{p}_k} \mathcal{M}$ at the start of the solve. For the solve to work, each c_1 must also be covered by the adjacent local maps along the curve, and stay within those maps throughout the solve. Although we do not try to ensure that the optimization stays within the parameterized tangent disc for each map, we find that in practice we never encountered solver issues if an interactive user is careful to place each high-curvature constraint point (the \mathbf{p}_k) in regions that overlap the previous point in the sequence, as well as the very first point when closing the curve.