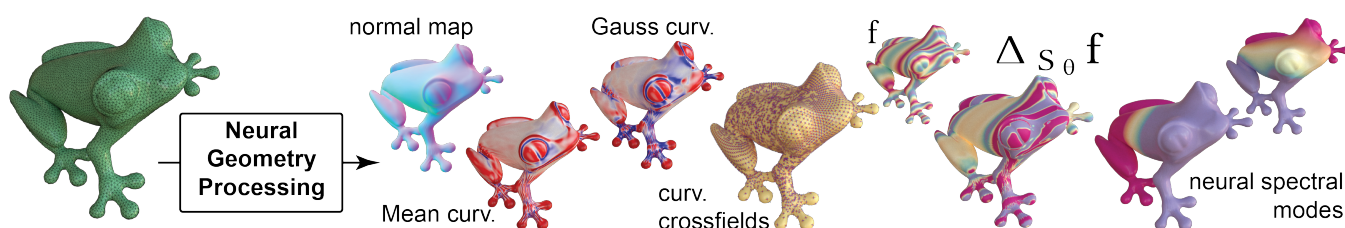# Neural Geometry Processing via Spherical Neural Surfaces

Romy Williamson[1] and Niloy J. Mitra[1,2]

[1]University College London   [2]Adobe Research

Figure 1: *Neural Geometry Processing. We encode input genus-0 surfaces as overfitted neural networks and propose operators on them. Specifically, we describe how to compute the normals and the First and Second Fundamental Forms, and hence compute curvatures. We also define a Laplace-Beltrami operator directly using the neural representation, thus enabling processing of scalar (or vector) fields on the underlying surface, and we find the smallest eigenmodes of the Laplace-Beltrami operator via a novel optimization scheme. We avoid unnecessary discretization in the estimates, as commonly encountered while using a traditional surface representation (e.g., a polygonal mesh).*

Abstract

Neural surfaces (e.g., neural map encoding, deep implicit, and neural radiance fields) have recently gained popularity because of their generic structure (e.g., multi-layer perceptron) and easy integration with modern learning-based setups. Traditionally, we have a rich toolbox of geometry processing algorithms designed for polygonal meshes to analyze and operate on surface geometry. Without an analogous toolbox, neural representations are typically discretized and converted into a mesh, before applying any geometry processing algorithm. This is unsatisfactory and, as we demonstrate, unnecessary. In this work, we propose a spherical neural surface representation for genus-0 surfaces and demonstrate how to compute core geometric operators directly on this representation. Namely, we estimate surface normals and first and second fundamental forms of the surface, as well as compute surface gradient, surface divergence and Laplace Beltrami operator on scalar/vector fields defined on the surface. Our representation is fully seamless, overcoming a key limitation of similar explicit representations such as Neural Surface Maps [MAKM21]. These operators, in turn, enable geometry processing directly on the neural representations without any unnecessary meshing. We demonstrate illustrative applications in (neural) spectral analysis, heat flow and mean curvature flow, and evaluate robustness to isometric shape variations. We propose theoretical formulations and validate their numerical estimates, against analytical estimates, mesh-based baselines, and neural alternatives, where available. By systematically linking neural surface representations with classical geometry processing algorithms, we believe this work can become a key ingredient in enabling neural geometry processing. Code is available via the project webpage.

1. Introduction

The use of neural networks to represent surfaces has gained rapid popularity as a generic surface representation. Many variants of this formulation [PFS*19, MON*18, GFK*18, QSMG] have been proposed, such as directly representing a single surface as an overfitted network, or a collection of surfaces, as encoder-decoder networks. The 'simplicity' of the approach lies in the network architectures (e.g., MLPs, UNets) being universal function approximators. Such deep representations can either be overfitted to given

meshes [DNJ20, MAKM21] or directly optimized from a set of posed images (i.e., images with camera information), as popularized by the NeRF formulation [MST*21]. Although there are some isolated and specialized attempts [NSS*22, CYW*23, YBHK21], no established set of operators directly works on a neural object, encoding an *explicit* surface, to facilitate easy integration with existing geometry processing tasks.

Traditionally, triangle meshes, more generally polygonal meshes, are used to represent surfaces. They are simple to define, easy to

store, and benefit from having a wide range of available algorithms supporting them. Most geometry processing algorithms rely on the following core operations: (i) estimating differential quantities (i.e., surface normals, and first and second fundamental forms) at any surface location; (ii) computing surface gradients and surface divergence on scalar/vector fields; and (iii) defining a Laplace Beltrami operator (i.e., the generalization of the Laplace operator on a curved surface) for analyzing the shape itself or functions on the shape.

Meshes, however, are discretized surface representations, and accurately carrying over definitions from continuous to discretized surfaces is surprisingly difficult. For example, the area of discrete differential geometry (DDG) [DPSS06] was entirely invented to meaningfully carry over continuous differential geometry concepts to the discrete setting. However, many commonly used estimates vary with the mesh quality (i.e., distribution of vertices and their connections), even when encoding the same underlying surface. In the case of the Laplace-Beltrami operator, the 'no free lunch' result [WMKG07] tells us that we cannot hope for any particular discretization of the Laplace-Beltrami operator to simultaneously satisfy all of the properties of its continuous counterpart. In practice, the most suitable choice of discretization, such as triangular or, more generally, polygonal meshes, can change depending on the specifics of the target application (see [BAB23] for a recent survey).

In the absence of native geometric operators for neural representations, the common workaround for applying a geometry processing algorithm to neural representations is discretizing these representations into an explicit mesh (e.g., using Marching cubes) and then relying on traditional mesh-based operators. Unfortunately, this approach inherits the disadvantages of having an explicit mesh representation, as discussed above. A similar motivation prompted researchers to pursue a grid-free approach for volumetric analysis using Monte Carlo estimation for geometry processing [SC20] or neural field convolution [NDS*23] to perform general continuous convolutions with continuous signals such as neural fields.

We propose a different approach to discretization-free geometry processing. First, we create a smooth and differentiable representation — *spherical neural surfaces (SNS)* — where we encode individual genus-0 surfaces using dedicated neural networks. Having a differentiable encoding allows one to directly compute surface normals, tangent planes, and Fundamental Forms directly with continuous differential geometry. The First Fundamental Form allows us to compute attributes of the parametrization, such as local area distortion. The Second Fundamental form allows us to compute surface attributes such as mean curvature, Gauss curvature, and principal curvature directions. Essentially, encoding surfaces as seamless neural maps unlocks definitions from differential geometry.

Other estimates require more thought. We implement two methods to apply the *continuous* Laplace Beltrami operator to any differentiable scalar or vector field (using Equations 13 and 16). Additionally, to enable spectral analysis on the resultant Laplace Beltrami operator, we propose a novel optimization scheme via a neural representation of scalar fields that allows us to extract the lowest k spectral modes, using the variational formulation of the Dirichlet eigenvalue problem. We use Monte Carlo estimates to approximate the continuous inner products and integration of functions on the (neural) surface. Our framework of spherical neural surfaces

makes optimizing many expressions involving surface gradients and surface integrals of smooth functions possible. We hope that our setup opens the door to solving other variational problems posed on smooth (neural) surfaces.

We evaluate our proposed framework on standard geometry processing tasks. In the context of mesh input, we assess the robustness of our estimates to different meshing of the same underlying surfaces (Figure 11) and isometrically related shapes (Figure 10). We compare the results against ground truth values when they are computable (e.g., analytical functions – see Figure 3 and Figure 4) and contrast them against alternatives. Specifically, we compare against traditional mesh-based methods (e.g., curvature estimates, cotan LBO, – see Figures 9, 11, 12) or neural alternatives (e.g., NGP [YBHK21] and neural implicits [NSS*22] – see Figure 7).

Our main contribution is introducing a novel representation and associated operators to enable neural geometry processing. We:

(i) introduce spherical neural surfaces as a representation to encode genus-0 surfaces seamlessly as overfitted networks;

(ii) compute surface normals, First and Second Fundamental Forms, curvature, continuous Laplace Beltrami operators without unnecessary discretization;

(iii) approximate the smallest spectral modes of the Laplace Beltrami operator on spherical neural surfaces;

(iv) show illustrative applications towards scalar field manipulation and neural mean curvature flow; and

(v) demonstrate the robustness and accuracy ours on different forms of input and quantitatively compare against alternatives.

2. Related Works

Neural Representations. Implicit neural representations, such as signed distance functions (SDFs) and occupancy networks, have recently been popularized to model 3D shapes. Notably, DeepSDF [PFS*19] leverages a neural network to represent the SDF of a shape, enabling continuous and differentiable shape representations that can be used for shape completion and reconstruction tasks; Davies et al. [DNJ20] use neural networks to overfit to individual SDFs; Occupancy Networks [MON*18] learn a continuous function to represent the occupancy status of points in space, providing a powerful tool for 3D shape modeling and reconstruction from sparse input data. Explicit neural representations, on the other hand, use neural networks to directly predict structured 3D data such as meshes or point clouds. Mesh R-CNN [GMJ19] extends Mask-RCNN to predict 3D meshes from images using a voxel-based representation followed by mesh refinement with a graph convolution network operating on the meshes' vertices and edges. In the context of point clouds, PointNet [QSMG] and its variants are widely used as backbones for shape understanding tasks such as shape classification and segmentation. Hybrid representations combine the strengths of both implicit and explicit methods. For example, Pixel2Mesh [WZL*18] generates 3D meshes from images by progressively deforming a template mesh using a graph convolutional network, integrating the detailed geometric structure typical of explicit methods with the continuous nature of implicit representations. More relevant to ours is AtlasNet [GFK*18] that models surfaces as a collection of parametric patches, balancing flexibility and precision in shape

representation, and Neural Surface Maps [MAKM21], which are overfitted to a flattened disc parametrization of surfaces to enable surface-to-surface mapping. Similarly to SNS, Explicit Neural Surfaces [WMVB23] were proposed in the context of 3D reconstruction from images, but the authors did not yet explore the properties of the differentiable surface representation.

Estimating Differential Quantities Traditionally, researchers have investigated how to carry over differential geometry concepts [dC76] to surface meshes, where differential geometry does not directly apply because mesh faces are flat, with all the 'curvature' being at sharp face intersections. Taubin [Tau95] introduced several signal processing operators on meshes. Meyer et al. [MDSB02] used averaging Voronoi cells and the mixed Finite-Element/Finite-Volume method; [CP05] used osculating jets; while [dGBD20] used discrete differential geometry [DPSS06] to compute gradients, curvatures, and Laplacians on meshes. Recently, researchers have used learning-based approaches to 'learn' differential quantities like normals and curvatures on point clouds [GKOM18, BSLF19, PFVM20].

More related to ours is the work by Novello et al. [NSS*22], who represent surfaces via implicit functions and analytically compute differential quantities such as normals and curvatures. We refer to this method as 'i3d' in our comparison 7. Later, Novello et al. proposed a related method, to evolve implicit functions [NdSS*23], and [MCR22] describes an alternative approach. With a similar motivation to Novello et al., Chetan et al. [CYW*23] fit local polynomial patches to obtain more accurate derivatives from pre-trained hybrid neural fields; they also use these as higher-order constraints in the neural implicit fitting stage. A similar idea was explored by Bednarik et al. [BPG*20], as a regularizer for the AtlasNet setup, by using implicit differential surface properties during training. The work by Yang and colleagues [YBHK21] is closest to ours in motivation - they also mainly focus on normal and curvature estimates but they additionally demonstrate how to perform feature smoothing and exaggeration using the local differential quantities in the target loss functions and thus driving the modification of the underlying neural fields. In Section 6, we compare our estimates against those obtained by others [NSS*22, YBHK21]. However, we go beyond normal and curvature estimates and focus on operators such as surface gradient, surface divergence and Laplace-Beltrami operators for processing scalar and vector fields defined on the surfaces.

Laplace Beltrami Operators on Meshes The Laplace-Beltrami operator (LBO) is an indispensable tool in spectral geometry processing, as it is used to analyze and manipulate the intrinsic properties of shapes. Spectral mesh processing [LZ10] leverages the eigenvalues and eigenfunctions of the LBO for tasks such as mesh smoothing and shape segmentation. Many subsequent efforts have demonstrated the use of LBO towards shape analysis and understanding (e.g., shapeDNA [RWP06], global point signatures [Rus07], heat kernel signature [SOG09]), eventually culminating into the functional map framework [OBCS*12]. An earlier signal processing framework [Tau95] has been revisited with the LBO operator [LZ10] for multi-scale processing applications (e.g., smoothing and feature enhancement). Although the Laplace-Beltrami operator on triangular meshes or, more generally, on polyhedral meshes, can be computed, these operators depend on the underlying discretization

of the surface (i.e., placement of vertices and their connectivity). See [BAB23] for a discussion. In the case of triangular meshes, the commonly used discretizations are the Uniform Laplace Operator and the cotan LBO. In Section 6, we compare our neural LBO with the cotan discretization and discuss their relative merits.

3. Spherical Neural Surfaces

We introduce a novel neural representation for continuous surfaces, called a 'Spherical Neural Surface' (SNS). An SNS is a Multi-Layer Perceptron (MLP) $S_\theta : \mathbb{R}^3 \to \mathbb{R}^3$, trained such that its restriction to the unit sphere ($\mathbb{S}^2 \subset \mathbb{R}^3$) is a continuous parametrization of a given surface. In terms of notation, we use S_θ or S as shorthand for the set $S_\theta(\mathbb{S}^2)$. In our current work, we primarily focus on creating SNSs from input triangle-meshes and analytically-defined surfaces, but there is potential to create SNSs from other representations such as pointclouds, SDFs and NeRFs. For example, Figure 15 shows initial results for generating an SNS from a neural SDF.

3.1. Overfitting an SNS to a Triangle Mesh

Given a genus-0 manifold triangle mesh, we first find an injective embedding of the mesh vertices onto the unit sphere, using a spherical embedding algorithm [SPK23, PH03]. We extend the embedding to a continuous embedding by employing barycentric coordinates, so that every point p_i on a (curved) triangle on the sphere corresponds to a unique point q_i on a (flat) triangle on the target surface. We overfit the network S_θ to this parametrization by minimizing the MSE between the ground truth and predicted surface positions:

$$\mathcal{L}_{MSE} := \frac{1}{N} \sum_{i=1}^{N} \| S_\theta(p_i) - q_i \|^2. \tag{1}$$

Further, to improve the fitting, we encourage the normals derived from the spherical map to align with the normals of points on the mesh, via a regularization term,

$$\mathcal{L}_{normal} := \frac{1}{N} \sum_{i=1}^{N} \| \mathbf{n}_{S_\theta}(p_i) - \mathbf{n}_{mesh}(q_i) \|^2 = \frac{2}{N} \sum_{i=1}^{N} (1 - \cos \alpha_i). \tag{2}$$

In this expression, $\mathbf{n}_{mesh}(q_i)$ is the (outwards) unit-normal on the mesh at q_i, and $\mathbf{n}_{S_\theta}(p_i)$ is the corresponding (outwards) unit-normal of the smooth parametrization at $S_\theta(p_i)$, which is derived analytically from the Jacobian of S_θ at p_i (see Section 3.2); the angle

Figure 2: *Spherical Neural Surface. Given an input surface — a mesh in this case — we progressively overfit an MLP to a spherical mesh-embedding, to represent the surface as S_θ. The loss term is simply the MSE between the ground truth and predicted surface positions, plus a (scaled) normals regularization term (Equation 2).*

α_i is the angle between $\mathbf{n}_{mesh}(q_i)$ and $\mathbf{n}_{S_\theta}(p_i)$. Figure 2 shows an example of an SNS over training stages.

3.2. Computing Differential Quantities using SNS

One of the advantages of Spherical Neural Surfaces as a geometric representation is that one can compute important quantities from continuous differential geometry - without any need for approximate numerical formulations or discretization. We just need to use the automatic differentiation functionality built into modern machine learning setups, and algebraically track some changes of variables, before applying the formulas from classical differential geometry.

For example, to compute the outwards-pointing normal $\mathbf{n}_S$, we first use `autodiff` to compute the 3×3 Jacobian of S,

$$\mathbf{J}(S) := \left[\frac{\partial S}{\partial x}, \quad \frac{\partial S}{\partial y}, \quad \frac{\partial S}{\partial z} \right],$$

and then turn this into a 3×2 Jacobian in local coordinates, by composing it with a 3×2 orthonormal matrix (R):

$$\mathbf{J}_S^{local} = \mathbf{J}(S)\,R. \tag{3}$$

The matrix R maps vectors in 2D local coordinates to tangent vectors on the sphere at a particular point, and the matrix $\mathbf{J}(S)$ maps tangent vectors on the sphere to tangent vectors on the surface S. If u and v are the local coordinates on the tangent plane of the sphere, then the columns of $\mathbf{J}_S^{local}$ are equal to the partial derivatives S_u and S_v respectively:

$$\mathbf{J}_S^{local} = \begin{pmatrix} | & | \\ S_u & S_v \\ | & | \end{pmatrix}, \tag{4}$$

and these vectors lie in the tangent plane. The normalized cross product of S_u and S_v is the outward-pointing unit normal:

$$\mathbf{n}_S = \frac{S_u \times S_v}{\|S_u \times S_v\|}. \tag{5}$$

In terms of standard spherical polar coordinates (θ and ϕ), we chose to use the following rotation matrix:

$$R = \begin{pmatrix} \cos\theta\cos\phi & -\sin\phi \\ \cos\theta\sin\phi & \cos\phi \\ -\sin\theta & 0 \end{pmatrix}. \tag{6}$$

The trigonometric quantities here are simple functions of x, y and z.

The matrix R is an orthonormal version of the usual spherical polar coordinates Jacobian. We avoid degeneracy at the poles by selectively re-labeling the x, y, and z coordinates so that θ is never close to zero or π. (This is equivalent to choosing a different chart for the points within some radius of the poles.)

Now, from $\mathbf{J}_S^{local}$, we can simply write the First Fundamental Form of the sphere-to-surface map as:

$$\mathbf{I}_S = \begin{pmatrix} E & F \\ F & G \end{pmatrix} = (\mathbf{J}_S^{local})^T \,\mathbf{J}_S^{local}. \tag{7}$$

To compute the Second Fundamental Form, we first compute the second partial derivatives of the parametrization with respect to the local coordinates, and then find the dot products of the second derivatives with the normal, $\mathbf{n} = \mathbf{n}_{S_\theta}$. The second derivatives are:

$$S_{uu} = \mathbf{J}(S_u)S_u, \quad S_{uv} = \mathbf{J}(S_u)S_v, \quad S_{vv} = \mathbf{J}(S_v)S_v. \tag{8}$$
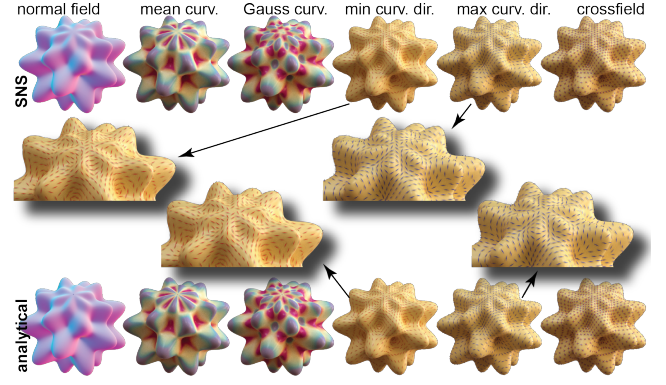
Figure 3: *First and Second Fundamental Forms. We show an analytically defined surface, `star` (bottom) and an SNS (top row) that has been overfitted to a mesh of the surface (with 40962 vertices). We show various differential quantities computed symbolically using Matlab (bottom) and computed on the SNS using our method (top). Our results align very closely with the analytically-computed results.*

We can now express the Second Fundamental Form as:

$$\mathbf{II}_S = \begin{pmatrix} e & f \\ f & g \end{pmatrix} = \begin{pmatrix} S_{uu} \cdot \mathbf{n} & S_{uv} \cdot \mathbf{n} \\ S_{uv} \cdot \mathbf{n} & S_{vv} \cdot \mathbf{n} \end{pmatrix}. \tag{9}$$

In terms of the Fundamental Forms, the Gauss curvature is

$$K = \frac{eg - f^2}{EG - F^2}, \tag{10}$$

and the mean curvature is

$$H = \frac{Eg - 2Ff + Ge}{2(EG - F^2)}. \tag{11}$$

Furthermore, the solutions to the quadratic equation $\det(\mathbf{I}_S - \lambda\mathbf{II}_S) = 0$ give us explicit expressions for the principal curvatures, that we then use to find the principal curvature directions directly on the Spherical Neural Surface. In Figure 3, we compare our differential estimates against analytical computations for an analytic surface; in Figure 6 we also show curvature estimates and principal curvature directions on other SNSs overfitted to non-analytic surfaces (ground truth values are not available). In Section 6, we evaluate the quality of our estimates and also evaluate how the estimates change as we fit SNSs to different mesh resolutions as proxies to the same underlying (continuous) surface.

4. Laplace-Beltrami Operator

This section briefly summarizes the relevant background for continuous surfaces that we need to develop a computation framework for Laplace-Beltrami and spectral analysis using SNS (Section 5).

4.1. Laplace-Beltrami Operator on Smooth Surfaces

The (Euclidean) Laplacian Δ of a scalar field $f : \mathbb{R}^n \to \mathbb{R}$ is defined as the *divergence* of the *gradient* of the scalar field. Equivalently,

Δf is the sum of the unmixed second partial derivatives of f:

$$\Delta f := \nabla \cdot \nabla f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}. \tag{12}$$

The Laplace-Beltrami operator, or *surface Laplacian*, is the natural generalization of the Laplacian to scalar fields defined on curved domains. If $\Sigma \subset \mathbb{R}^3$ is a regular surface and $f : \Sigma \to \mathbb{R}$ is a scalar field, then the Laplace-Beltrami operator on f is defined as the *surface divergence* of the *surface gradient* of f, and we denote this by $\Delta_\Sigma f$ to emphasize the dependence on the surface Σ. Thus,

$$\Delta_\Sigma f := \nabla_\Sigma \cdot \nabla_\Sigma f. \tag{13}$$

The surface gradient of the scalar field f (written $\nabla_\Sigma f$), and the surface divergence of a vector field $\mathbf{F} : \Sigma \to \mathbb{R}^3$ (written $\nabla_\Sigma \cdot \mathbf{F}$), can be computed by smoothly extending the fields to fields $\tilde{f}$ and $\tilde{\mathbf{F}}$ respectively, which are defined on a neighborhood of Σ in $\mathbb{R}^3$.

The surface gradient is defined as the orthographic projection of the Euclidean gradient of $\tilde{f}$ into the local tangent plane:

$$\nabla_\Sigma f = \nabla \tilde{f} - (\nabla \tilde{f} \cdot \mathbf{n})\mathbf{n}, \tag{14}$$

where $\mathbf{n}$ is the unit surface normal at the point. One can show that this definition is independent of the choice of extension. The surface divergence of the vector field $\mathbf{F}$ is defined as

$$\nabla_\Sigma \cdot \mathbf{F} = \nabla \cdot \tilde{\mathbf{F}} - \mathbf{n}^T \mathbf{J}(\tilde{\mathbf{F}})\mathbf{n}, \tag{15}$$

where $\mathbf{J}(\tilde{\mathbf{F}}) := \left[\frac{\partial \tilde{\mathbf{F}}}{\partial x_1}, \quad \frac{\partial \tilde{\mathbf{F}}}{\partial x_2}, \quad \frac{\partial \tilde{\mathbf{F}}}{\partial x_3} \right]$ is the Jacobian of $\mathbf{F}$. Since $\mathbf{J}(\tilde{\mathbf{F}})\mathbf{n}$ is the directional derivative of $\tilde{\mathbf{F}}$ in the normal direction, then $\mathbf{n}^T \mathbf{J}(\tilde{\mathbf{F}})\mathbf{n} = \mathbf{J}(\tilde{\mathbf{F}})\mathbf{n} \cdot \mathbf{n}$ can be thought of as the contribution to the three-dimensional divergence of $\tilde{\mathbf{F}}$ that comes from the normal component of $\tilde{\mathbf{F}}$, and by ignoring the contribution from the normal component, we get the two-dimensional 'surface divergence'. Although these expressions depend a-priori on the particular choice of extension, the surface gradient and surface divergence are in fact well-defined (i.e., any choice of extension will give the same result).

By expanding out the definitions of surface gradient and surface divergence, we can derive an alternative formula for the surface Laplacian ($\Delta_\Sigma f$) in terms of the (Euclidean) Laplacian ($\Delta \tilde{f}$), the gradient ($\nabla \tilde{f}$) and the Hessian ($\mathbf{H}(\tilde{f}) = \mathbf{J}(\nabla \tilde{f})^T$) as

$$\Delta_\Sigma f = \Delta \tilde{f} - 2H \nabla \tilde{f} \cdot \mathbf{n} - \mathbf{n}^T \mathbf{H}(\tilde{f})\mathbf{n}. \tag{16}$$

The dependence on the surface is captured by the normal function $\mathbf{n}$, and the mean curvature H. (Refer to [Rei82, XZ03] for a derivation.) In the case when $\tilde{f}$ is a 'normal extension' — i.e., it is locally constant in the normal directions close to the surface — then the second and third terms disappear, so that $\Delta_\Sigma f$ is consistent with $\Delta \tilde{f}$.

If we substitute in the coordinate function $\mathbf{x}$ for $\tilde{f}$, then the first and third terms disappear, and we are left with the familiar equation

$$\Delta_\Sigma \mathbf{x} = -2H\mathbf{n}. \tag{17}$$

This formula is often used in the discrete setting to compute the mean curvature from the surface Laplacian.

As in Section 3.2, these formulae are easily implemented for functions defined on an SNS. All that is required to implement Equation 16 is that we have access to an extension $\tilde{f}$ and its first and second derivatives. Therefore it can be applied when f is (the restriction to the surface of) a closed-form function on the embedding space (e.g., Figures 11 and 12), or when f and $\tilde{f}$ are defined by a network, allowing us to use `autodiff` (see Section 4.2).

We have provided Equation 16 as an alternative formula to the standard definition (Equation 13) because where a field evolves on a domain that remains constant, it allows us to precompute the mean curvature H and normal $\mathbf{n}$, and then only the Euclidean quantities ($\nabla \tilde{f}$, $\mathbf{H}(\tilde{f})$, and $\Delta \tilde{f}$) need to be updated. In Figure 9 (2nd and 3rd row) we demonstrate both methods and there is no visible difference.

4.2. Spectrum of the Laplace-Beltrami Operator

Given two scalar functions, f and g, defined on the same regular surface Σ, their L^2 inner product is

$$\langle f, g \rangle_{L^2(\Sigma)} := \int_\Sigma f g \, dA. \tag{18}$$

Then, the L^2 norm of a scalar function can be expressed as,

$$\|f\|_{L^2(\Sigma)} = \sqrt{\langle f, f \rangle_{L^2(\Sigma)}} = \sqrt{\int_\Sigma |f|^2 \, dA}. \tag{19}$$

The Laplace Beltrami operator, Δ_Σ, is a self-adjoint linear operator with respect to this inner product. This means that there is a set of eigenfunctions of Δ_Σ that form an orthonormal basis for the space $L^2(\Sigma)$ - the space of 'well-behaved' scalar functions on Σ that have a finite L^2-norm. This basis is analogous to the Fourier basis for the space of periodic functions on an interval.

In the discrete mesh case, the eigenfunctions of the Laplace-Beltrami operator are computed by diagonalizing a sparse matrix [LZ10, BAB23] based on the input mesh. In the continuous case, however, we no longer have a finite-dimensional matrix representation for Δ_Σ. Instead, we exploit a functional analysis result that describes the first k eigenfunctions of Δ_Σ as the solution to a minimization problem, as described next.

First, we define the Rayleigh Quotient of a scalar function f to be the Dirichlet energy of the scalar function, divided by the squared L^2-norm of the function:

$$Q_\Sigma(f) := \frac{\|\nabla_\Sigma f\|_{L^2(\Sigma)}^2}{\|f\|_{L^2(\Sigma)}^2}. \tag{20}$$
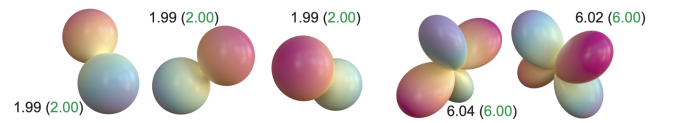
Figure 4: *Neural Spectral Basis against Analytical Solutions. The sphere is a genus-0 surface for which the eigenspaces of the Laplace-Beltrami operator are very well understood: the eigenspaces are the spaces spanned by linear combinations of the spherical harmonics for each frequency. This means that each of our optimized eigenfunctions should belong to one of these eigenspaces, and therefore, the Rayleigh quotient should match the energy level of the corresponding spherical harmonics. For evaluation, we compute the Rayleigh quotient using a large number of samples for Monte Carlo integration, since we cannot analytically compute the integral. For reference, we present the ground truth Rayleigh quotients, in green.*

Then, the first k eigenfunctions of Δ_Σ are the minimizers of the Rayleigh Quotient, as given by,

$$\Psi_0, \Psi_1, ..., \Psi_{k-1} = \text{argmin}_{\Psi_0^*, ..., \Psi_{k-1}^*} \sum_{i=0}^{k-1} Q_\Sigma(\Psi_i^*)$$

$$such\ that \quad \langle \Psi_i, \Psi_j \rangle_{L^2(\Sigma)} = 0 \quad for\ all \quad i \neq j. \quad (21)$$

We use Ψ_0 to denote the first eigenfunction, which is always a constant function. The constraint states that the eigenfunctions are orthogonal inside the inner-product space $L^2(\Sigma)$.

In addition, the Rayleigh Quotient of the eigenfunction Ψ_i is the positive of the corresponding (negative) eigenvalue, i.e.,

$$\Delta_\Sigma \Psi_i(x) = -Q_\Sigma(\Psi_i)\Psi_i(x) \quad \forall x \in \Sigma. \quad (22)$$

Physically, the eigenfunctions of Δ_Σ represent the fundamental modes of vibration of the surface Σ. The Rayleigh Quotient of Ψ_0 is zero. As the Rayleigh quotient increases, the energy increases, and the eigenfunctions produce higher frequency patterns.

5. Spectral Analysis using SNS

To find the *continuous* eigenmodes of the Laplace-Beltrami operator on an SNS, we require a continuous and differentiable representation for scalar functions defined on the surface. In our framework, we represent smooth scalar fields on the sphere $\mathbb{S}^2$ by MLPs, g_η, from $\mathbb{R}^3$ to $\mathbb{R}$, which are parametrized by η. We only ever evaluate the MLP g_η on $\mathbb{S}^2 \in \mathbb{R}^3$, but because the network's domain is $\mathbb{R}^3$ then it defines an extension of the scalar field, and this allows us to compute the Euclidean gradient ∇g_η. Then, if $S_\theta : \mathbb{R}^3 \to \mathbb{R}^3$ is an SNS, any smooth scalar field h whose domain is the surface defined by S_Θ can be defined implicitly by the equation

$$h \circ S_\theta = g_\eta. \quad (23)$$

Applying the chain rule, we can compute the gradient of h:

$$\nabla h = (\mathbf{J}(S_\theta)^T)^{-1} \nabla g_\eta, \quad (24)$$

which allows us to compute $\nabla_{S_\theta} h$ (Equation 14) and $Q_{S_\theta}(h)$ (Equation 20) *without* explicitly computing h.

To optimize for a (smooth) scalar function g_{η_k} so that h approximates the kth non-constant eigenfunction of the Laplace-Beltrami
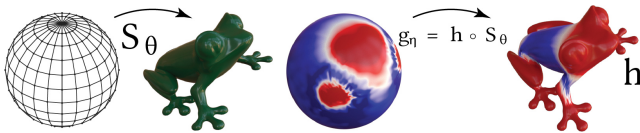
Figure 5: *Encoding Scalar Fields on Neural Surfaces. We encode scalar fields on the sphere (and corresponding extensions into $\mathbb{R}^3$) as MLPs $g_\eta : \mathbb{R}^3 \to \mathbb{R}$, with parameters η. Because S_θ, considered as a parametrization, is bijective, then we can implicitly define any smooth scalar field h on the surface in terms of one of the functions g_η (restricted to $\mathbb{S}^2$). We visualize the one-to-one correspondence between scalar fields defined on the surface, and scalar fields defined on the sphere, by using vertex-colours to display the fields.*

operator, we use gradient descent to optimize the weights, η_k. Specifically, we use a combination of two loss terms:

$$\mathcal{L}_{Rayleigh} := Q_S(h) \quad and \quad \mathcal{L}_{ortho} := \sum_{i=0}^{k-1} \langle h, h_i \rangle_{L^2(S)}^2, \quad (25)$$

where we sequentially compute the eigenfunctions, and h_i is the ith smallest eigenfunction. Recall that we represent each such eigenfunction using a dedicated MLP (except for h_0, which is the constant eigenfunction). The loss term $\mathcal{L}_{ortho}$ encourages the scalar field h to be orthogonal to all previously-found eigenfunctions (including the constant eigenfunction). The quantity $Q_{S_\theta}(h)$ depends on the Jacobian of S_θ, through $\nabla_{S_\theta} h$ and ∇h. However, since the surface does not change during optimization and θ stays fixed, we precompute $(\mathbf{J}(S_\theta)^T)^{-1}$. Note that, if we were to optimize over the subspace of the unit-norm functions in $L^2(\Sigma)$, then we could replace the Rayleigh Quotient with the Dirichlet energy. However, it is difficult to construct a function, based on an MLP, that produces only unit-norm functions (in the L^2 sense). In initial experiments, we used a unit-norm loss in combination with the Dirichlet Energy and orthogonality loss, but we observed some robustness issues, such as a tendency for the functions to collapse to zero. We found that robustness increased when we replaced the Dirichlet Energy by the Rayleigh Quotient, and we think that this is the Rayleigh Quotient, by cancelling out the effect of magnitude, is less in opposition to the unit-norm loss than the Dirichlet energy is. The Rayleigh Quotient design choice greatly reduces the importance of the unit-norm loss, but we retained it as a regularizer, to prevent functions from collapsing to zero.

Although functions h are not required to have unit norm, we used a regularization term to prevent the L^2-norm of h from either exploding or vanishing:

$$\mathcal{L}_{reg} := (\|h\|_{L^2(S)} - 1)^2. \quad (26)$$

The combined loss term is expressed as,

$$\mathcal{L} = \mathcal{L}_{Rayleigh} + \lambda_{ortho}\mathcal{L}_{ortho} + \lambda_{reg}\mathcal{L}_{reg}. \quad (27)$$

We choose large initial values for the coefficients because we find that this improves stability, and then we reduce the coefficients so that the Rayleigh Quotient is the dominant term during the later stages of optimization (see Section 6 for details).

Finally, since we cannot exactly compute the integrals for the inner products in these loss terms, we approximate the inner products using Monte-Carlo sampling. Specifically, we approximate,

$$\langle h, h_i \rangle_{L^2(S)} \approx \frac{Area(S)}{N} \sum_{j=1}^{N} h(q_j)h_i(q_j), \quad (28)$$

where $\{q_j\}_{1 \leq j \leq N}$ is a set of uniformly distributed points on the surface S_θ, corresponding to points $\{p_j\}_{1 \leq j \leq N}$ on the sphere; $h(q_j)$ and $h_i(q_j)$ are evaluated by applying scalar field MLPs g_η and g_{η_i} to p_j. (In the overfitting stage, we normalize all our Spherical Neural Surfaces to have an area equal to 4π.) For the Monte Carlo step, we generate N uniformly distributed points on the surface S_θ via the following steps (rejection sampling):

(i) Generate $M \gg N_{target}$ samples (p_j) from the 3D normal distribution $\mathcal{N}(\mu = \mathbf{0}, \Sigma = \mathbf{I})$. Normalize, to unit norm, to produce a dense uniform distribution of points on the sphere $\mathbb{S}^2 \subset \mathbb{R}^3$.
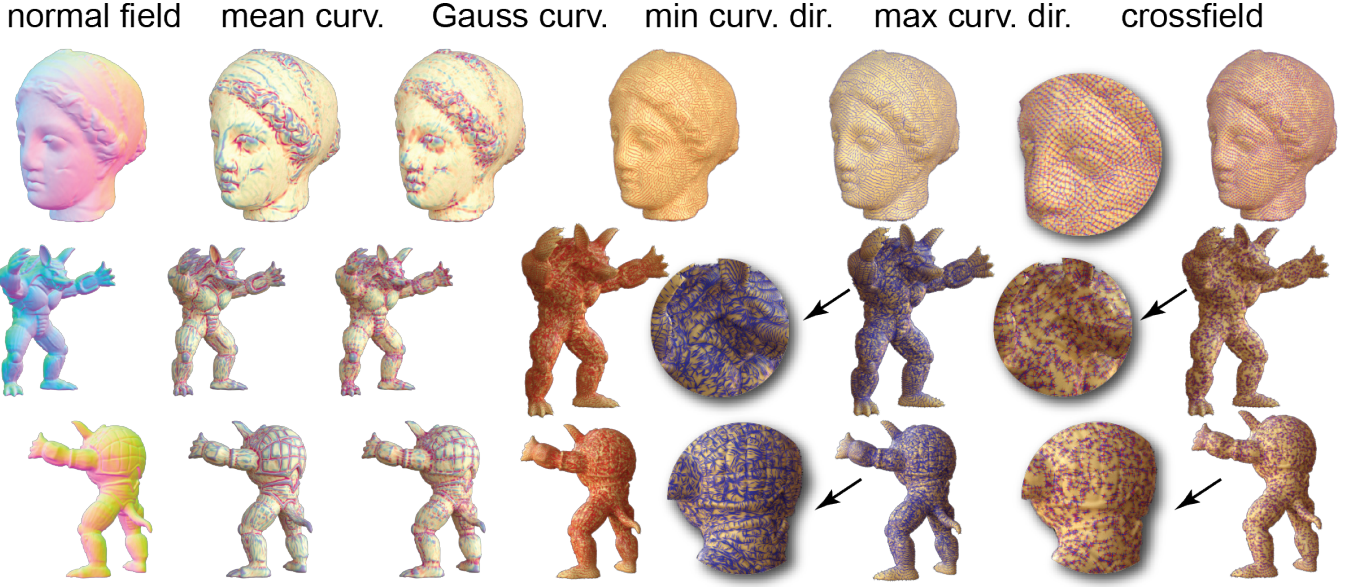
Figure 6: *First and Second Fundamental Forms. An SNS represents a smooth, seamless parametrization of a genus-0 surface, and we can compute the partial derivatives of this parametrization using automatic differentiation. We compute the normals and the First Fundamental Form of the parametrization from the first derivatives. Subsequently, computing the second derivatives enables us to construct the Second Fundamental Form of the parametrization. Hence, we compute Mean Curvature, Gaussian Curvature, and Principal Curvature Directions at any point without any unnecessary discretization error. To render scalar fields, we took samples at the 2.6 million vertices of an icosphere.*

(ii) Compute the local area distortion ($d_j = \sqrt{E_j G_j - F_j^2}$), for each point p_j, using the First Fundamental Form.

(iii) Perform rejection sampling, such that the probability of keeping point p_k is equal to $\frac{d_k}{\sum_{j=1}^{M} d_j} N_{target}$.

(iv) Push the samples p_j (on the sphere) through the map S_θ.

Following this process, the *expected* number of sample points will equal N_{target} — a specified parameter — however, the actual number of sample points (N) may vary.

6. Evaluation

Dataset We evaluate our method on various meshes of the unit sphere, various meshes of the Igea model, and four analytic shapes. In order to evaluate the robustness of our method, we included several 'bad' meshes (e.g. with many thin triangles) and unusual meshes such as those with a very sharp contrast in triangle size across different regions, and two polygonal meshes. (See Figures 11 and 12.)

The four analytic shapes, which we designed for this experiment, are: `mushroom`, `flower`, `bobble`, and `star`. In Figure 7, we show results on the `mushroom` only. Refer to the supplemental, for the analytic parametrizations of all four shapes, and for additional results on the `flower`, `bobble`, and `star`. The advantage of using analytic shapes for evaluation is that we can compute the ground truth for any derived geometric quantity (such as curvature). We used symbolic Matlab to compute the ground truth differential quantities, given the analytic parametrization. We provide text files, in the supplemental, which contain the symbolic expressions.

**Evaluation Metrics** To evaluate the quality of the surface representation, we measure:

(i) The distance from each point on the reconstructed surface, to the analytic surface (geometric error).

(ii) The angle deviation of the surface normal at each (reconstructed) surface point, compared to the ground truth surface normal at the the closest analytic surface point.

(iii) The angle deviation of the minimum curvature direction at each (reconstructed) surface point, compared to the ground truth minimum curvature direction at the the closest analytic surface point.

(iv) The absolute error of the Gaussian curvature and Mean curvature at each (reconstructed) surface point, compared to the ground truth curvatures at the the closest analytic surface point.

For the Laplace Beltrami operator, we compare our LBO to other LBOs when applied to a frequency-varying sine-wave type scalar field, and in the case of a sphere, we compute the absolute error with respect to the analytic LBO (see supplemental for details).

Comparison Methods We compare the SNS against three alternative methods for estimating geometric quantities. (These methods do not support LBO computation or spectral analysis.)

- *Monge-fitting [PC24]]* is a classical mesh-based method that uses osculating jets to approximate differential quantities of the underlying curved surface. We used the official implementation available via CGAL.
- *NFGP [YBHK21]* is a method with similar motivation to ours, in the sense that they fit a neural representation to a mesh and
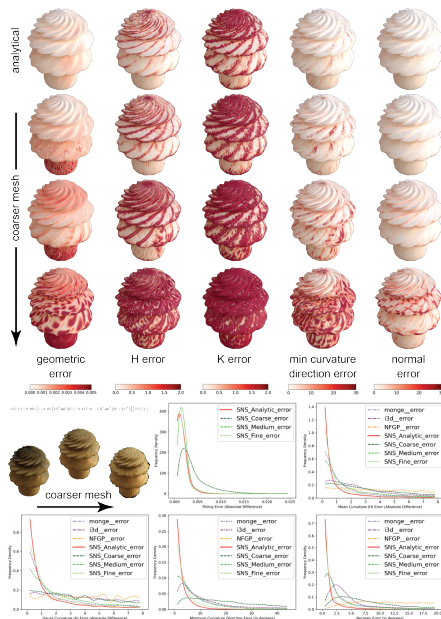
Figure 7: *Accuracy of Differential Quantity Estimation. As shown by the colourmaps, the quality of the estimation of differential quantities (mean curvature, Gauss curvature, minimum curvature direction and normals) decreases gradually as mesh-resolution decreases. We show results for an SNS overfitted to analytical parametrization, with 102400 sample points (top), and SNSs fitted to meshes with 40962, 10242 and 2562 vertices. (The errors are sampled at a much higher resolution, at 655362 points.) The level of error for the finest level mesh is close to the level of error for the 'analytical' SNS and we argue that this demonstrates the 'interpolation' capability of our method: it shows that fitting an SNS to an appropriately high-resolution mesh successfully interpolates the underlying surface (even for meshes where the analytic parametrization is not available). We also show plots of the error on the various differential quantities, compared against three additional methods (when functions are available): Monge-fitting, NFGP and i3d. A minimum curvature direction function was not available for NFGP.*

Figure 8: *Spectral Modes on Neural Surfaces. We propose how to compute the lowest few spectral modes of the neural Laplace Beltrami operator Δ_{S_θ} via minimization of the Rayleigh quotient, without requiring any unnecessary discretization. Here we show the lowest few spectral modes of different SNSs. The lowest spectral mode is constant, and not included in this figure. Each of the spectral modes is represented by a dedicated MLP (see Equation 23).*

compute differential quantities in order to perform edits in the shape. However, they use a deep-implicit representation, so their differential quantities additionally depend on the representation's adherence to the Eikonal property, whereas we use an explicit representation.

- *i3D [NSS*22]* is also a deep implicit method that computes differential quantities, and it improves the overfitting process itself by encouraging the curvature of the deep implicit to match the computed curvature of the input mesh. Because our method, NFGP, and Monge fitting do not require the curvature information from the mesh, we did not provide the mesh-curvature information to this method either and we found that the method suffered as a result.

(i) What is the effect on the estimates when we fit the SNS to different resolution meshes of an analytic shape? In Figure 7, we fit SNSs to analytic shapes (`mushroom`), and plot the absolute error of derived geometric quantities (against the ground truth, which is computed from the analytic parametrization using symbolic Matlab). We fit SNSs to a coarse, medium and fine mesh (with 2562, 10242 and 40962 vertices, respectively) and, for reference, we also fit an SNS directly to the analytic parametrization (without meshing).

As expected, the accuracy of the derived quantities is highest when we fit the SNS directly to the analytic parametrization, and the accuracy of the mesh-fitted SNSs approaches the analytically-fitted SNS as the mesh resolution increases. Since the accuracy of the SNS fitted to the fine-resolution mesh (40962 vertices) is consistently close to the accuracy of the analytically-fitted SNS, we argue that fitting the SNS to a reasonably fine mesh is a good way to find a smooth approximation of an underlying smooth surface, when no analytic parametrization is available. We also show the distribution of errors for one of the shapes (`mushroom`), for SNSs fitted to each resolution of the mesh, and for the three comparison methods: Monge-fitting, NFGP and i3D.
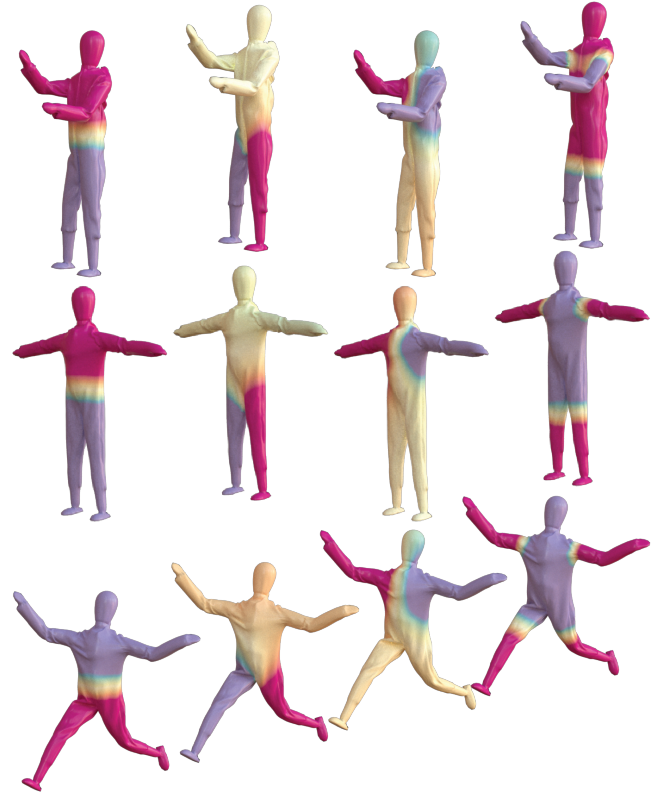
(ii) How do the position, normals and curvature estimates of SNS compare to those from other methods? In Figure 7, alongside the SNS errors, we show the absolute errors of computing the
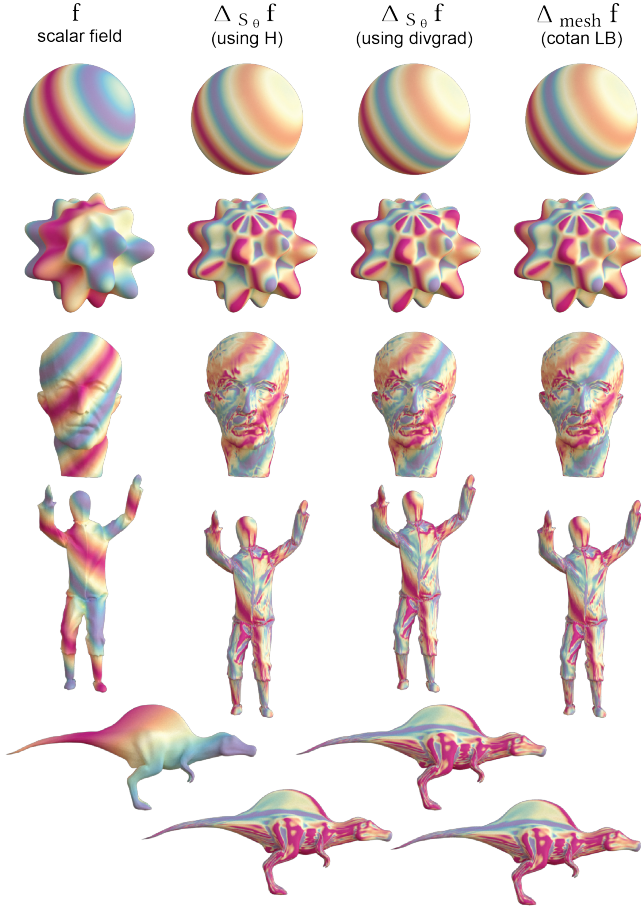
Figure 9: *Laplace Beltrami on Neural Scalar Fields. We present two ways to compute Laplace Beltrami operators on any smooth scalar field, f, defined on a neural surface S_θ (top): first, using mean curvature estimates (top-middle) (Equation 16) and second, using the divergence of gradient definition (divgrad) (bottom-middle) (Equation 13). We represent scalar fields on surfaces using dedicated MLPs. (Bottom) For comparison, we also show results using the cotan LBO (with a lumped mass matrix) on a dense mesh, whose vertices are the image of the vertices in a dense icosphere mesh under the transformation S_θ.*

Figure 10: *Spectral SNS Modes across Isometric Poses. If we compute the first few eigenfunctions on SNSs that represent near-isometric surfaces, and order the functions according to their Rayleigh quotients, we see a clear correspondence between corresponding eigenfunctions on each surface. Although our computation of the LBO and its spectrum involves extrinsic terms such as mean curvature and normals, the Laplace Beltrami operator is intrinsic so its spectrum does not depend on the particular embedding of a surface into $\mathbb{R}^3$, therefore the eigenmodes of isometric SNSs should appear similar (up to a possible sign change).*

same geometric quantities on the fine-resolution mesh, using three alternative methods. The first method is Monge-fitting, which fits a piecewise polynomial surface to the mesh and allows us to compute the geometric quantities on the vertices. The other two methods (i3d and NFGP) are deep implicit methods, which fit a neural signed-distance function (a deepSDF) to the mesh. The i3D method has the option to use mesh-curvature information to improve the fitting, but for consistency, we did not provide this, because the other neural methods (ours, and NFGP) do not require additional mesh-curvature information. We use the CGAL implementation for Monge-fitting, and the original author-implementations for the two deep-implicit methods. The geometric quantities on a deepSDF are computed using automatic differentiation. (Although this appears at first glance

to be similar to computing geometric quantities on an SNS, the computations are different due to deepSDF being an implicit representation while SNS is explicit. In contrast to SNS, deepSDFs use a soft constraint enforced by an Eikonal loss term, without which the deepSDF can only be used for mesh-reconstruction and not for the estimation of geometric quantities.)

We observe that given only a fine-resolution mesh, the most accurate method is consistently the SNS, except on the `flower` shape where it is Monge-fitting. The second most accurate method appears, on average, to be the Monge-fitting, but impressively the medium-resolution SNS is sometimes able to perform similarly well, showing that SNS has can robustly interpolate from limited geometric data.

On all shapes, i3D performed worse than the fine-resolution SNS and the Monge-fitting. The NFGP method produces consistently poor results. We believe that this method struggles to capture finer surface details (e.g. on the `mushroom`) and thin structures (e.g. on
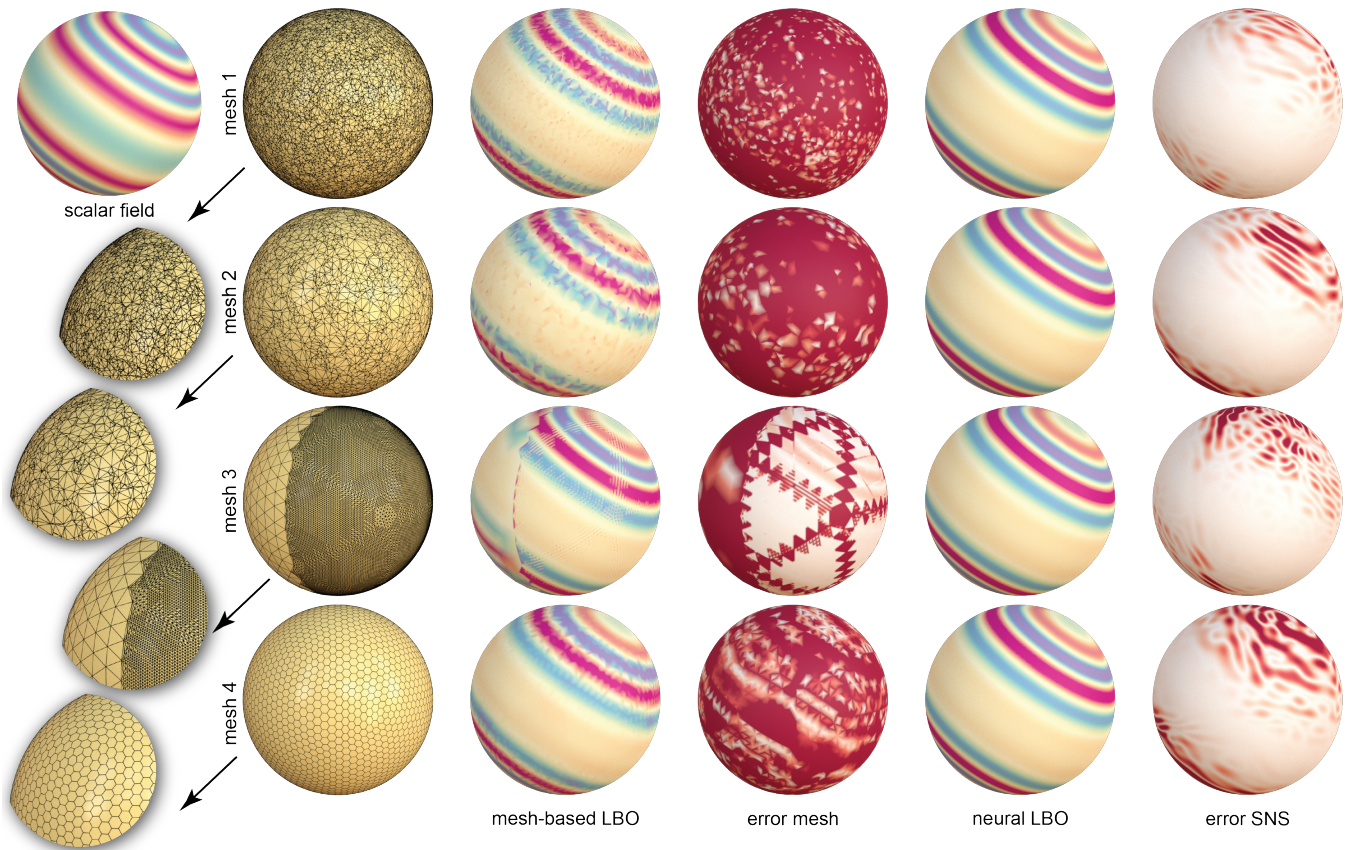
Figure 11: *Robustness of Neural LBO (Analytic). We demonstrate that our neural LBO is largely unaffected by differences in mesh density and mesh quality on the starting mesh. While the common mesh-based LBOs are sensitive to the quality and the density of the mesh, the neural LBO is more consistent when applied to a scalar field defined on an SNS that is generated from different meshes of the same surface. The mesh-based LBO is the cotan LBO for the triangle-meshes and the virtual-refinement Laplace [BHKB20] for polygonal meshes. The scalar field is analytic (a variable frequency sine-wave) on a sphere (for which the mean-curvature is one and the normals are trivial), because this allows us to compute hessian, gradient, normals and mean curvature analytically, so that we can compare against the ground truth LBO using equation 16. Please refer to the supplemental for further details. The error colourmap runs from white (no error) to red (higher error).*

the flower), at least with the default settings. However, we were limited by compute resources and it is possible that better results could be achieved given a longer training time (please see the 'NFGP Note' in the Supplemental). Observations of the reconstructed meshes show that neither of the deep implicit methods capture the input geometry well, leading to poor results in the estimation of geometric quantities.

(iii) How consistent is the SNS LBO, when we fit an SNS to different meshes of the same shape? In Figure 11 and 12, we put a test-function (a variable-frequency sine-wave scalar field) on two different surfaces: the sphere and Igea, respectively. We fit the SNS to various different meshings of these shapes, and in each case compute the result of the applying the SNS LBO to the scalar field. To fit an SNS to a polygonal mesh, we first convert to a triangle mesh by splitting the faces.

For reference, we also show the result of applying a mesh-based LBO to the scalar field (sampled at the vertices). For triangle meshes,

we use the cotan LBO (with vertex area as 1/3 the one-ring area, and a lumped mass matrix). For polygonal meshes, we use the Virtual Refinement Laplace operator from Polygon Laplacian Made Simple [BHKB20], with the default Area Minimizer setting.

Visually, the SNS LBO produces more consistent and less noisy results across different meshings. This is in part because the SNS LBO is discretization-free, so we can sample at any point (not just vertices of the original mesh), and partly because the SNS LBO is able to leverage information from the derivatives of the scalar field (which is always available when the scalar field is analytic, or neural) whereas the mesh-based methods *cannot* use this additional available information.

(iv) How does the SNS LBO compare to the ground truth LBO, when the analytic operator is available? In the case of a sphere, we know the mean curvature and normals analytically at every point, so we can use Equation 16 to compute the ground-truth result of applying the LBO to the (analytically-defined) scalar field. We show

Figure 12: *Robustness of Neural LBO (Non-Analytic). For non-analytic surfaces, we do not have access to a ground-truth LBO, because the precise underlying surface geometry is ambiguous. However, we show that the neural LBO is more consistent than mesh-based LBOs when fitted to different meshes of the same surface. This is because the smooth geometry of the SNS remains quite consistent when fitted to different meshes of the same surface, and the neural LBO is defined everywhere on the smooth surface - not just at the mesh-vertices. The neural LBO on mesh 2 does display some visible differences, however this is because the hair of Igea contains some high-frequency details. This is a signal which was lost in the lower-resolution meshes, so it is not an artefact or noise.*

this result in Figure 11 , and also display the error maps of the mesh-based LBOs and the neural LBO. Clearly, the error of the neural LBO is smaller than the error of the mesh-based LBO, and the error is smallest where the scalar field is low-frequency.

(v) Neural LBO on scalar fields on non-analytic surfaces. To evaluate the accuracy of our Laplace-Beltrami operator on SNSs, we generated random scalar fields using sine waves, and visually compared the results of applying three different forms of the LBO, on five different surfaces (see Figure 9). To compute $\Delta_{S_\theta} f$ using the mean curvature formulation (Equation 16), we computed the gradient and Hessians of the scalar field analytically and combined these with the SNS estimates of the normals and mean curvature. We also computed $\Delta_{S_\theta} f$ using the 'divergence of gradient' definition (Equation 13) - in this version, all of the differential calculations were carried out by `autograd`. We show the results using vertex-colours, on a dense icosphere mesh.

Finally, we computed the result of applying the cotan LBO to the scalar field (sampled at the vertices of the dense icosphere mesh).

At this resolution, all three versions of the LBO are extremely close, however, we notice that the cotan estimate on the `star` (row 2) contains a small discontinuity, not present in the SNS estimates. From a smooth shape and a smooth scalar field, the resulting Δ_S should be smooth, which is the case for the SNS estimates.

(vi) What is the effect of different levels of distortion in the spherical mesh embedding? Figure 13 shows area-distortion colourmaps for four different spherical embeddings of the frog mesh. Spherical embeddings were computed by performing explicit deformations of the mesh-embedding produced by multi-resolution embedding [SPK23]. The distortion colourmap is shown on the original mesh. The area-distortion at a single vertex is computed as the one-ring area on the sphere mesh divided by the corresponding one-ring area on the surface mesh. The second and third rows show the SNS trained on each parametrization, with colouring based on the normals and the mean curvature. We see that first order estimates (normals) are more consistent than second order estimates (curvature). We also see that, although the geometry is close to the input surface in all cases, artefacts such as 'ripples' occur more frequently in white areas, where the MLP maps a large area on the sphere to a small area on the surface.

Figure 13: *We show area-distortion colourmaps for four different spherical embeddings of the frog mesh, and demonstrate the effect of different patterns of distortion on the accuracy of fitting, and normals and curvature estimates.*

Implementation Details All of our networks in our experiments were trained on Nvidia GeForce P8 GPUs. The SNS network is a Residual MLP with input and output of size three, and eight hidden layers with 256 nodes each. The networks used to represent scalar fields (in the eigenfunction and heat flow experiments) are very small Residual MLPs with input size three, output size three and two hidden layers with 10 nodes each. (To make the output value a scalar, we take the mean of the three output values.) This rather small network-size appears to be sufficient to represent the smallest eigenfunctions, however it might be necessary to add more nodes if we aim to represent highly complex scalar fields. For both scalar fields and SNS, we use `softplus` as the activation function. We use the RMSProp optimizer with a learning rate of 10^{-4} and a momentum of 0.9. We trained each SNS for up to 20,000 epochs (eight to ten hours), and fixed the normal regularization coefficient to be 10^{-4}. The time per epoch increases slightly with the number of vertices in the mesh, but for simpler shapes (such as the sphere) the optimization converges in fewer epochs.

Can our optimization process reproduce eigenfunctions of the LBO when the analytic solution is known? In the eigenfunc-

tion experiment, we used the initial parameter values $\lambda_{ortho} = 10^3$ and $\lambda_{unit} = 10^4$ and then we reduced the coefficients linearly to $\lambda_{ortho} = 1$ and $\lambda_{unit} = 10^2$, over 10,000 epochs. These settings prevent the scalar field from collapsing to the zero-function during the initial stages of optimization, whilst allowing the Rayleigh Quotient to dominate the loss in the later stages. We optimized each eigenfunction for up to 40,000 epochs (approximately six hours on our setup).

We generate $M = 100,000$ points for the initial uniform distribution on the sphere, and we use $N_{target} = 10,000$ in the rejection sampling process. The points stay fixed during each optimization stage. We believe that this is a possible reason why the Rayleigh Quotient computed on the training points is often slightly lower than the Rayleigh Quotient that is computed with a different, larger set of uniform samples, and a better sampling strategy might improve the accuracy of our optimization.

Heat Flow and Mean Curvature Flow We can approximate the evolution of any initial scalar field f on an SNS S_θ, according to the process defined by the heat equation, $f_t = \Delta_{S_\theta} f$. In our implementation, we represent the field f implicitly, using Equation 23. In each time step, we compute $\Delta_{S_\theta} f$ for the current scalar field f, using the divergence of gradient formula in Equation 13. Then we select a small value ($d = 10^{-3}$) and we compute $f + d\Delta_{S_\theta} f$ at some fixed sample points (in Figure 14-left, we used 10,242 points). We then update the scalar field, by finetuning the network for up to 100 epochs to fit the new sample points, using a simple MSE loss. Qualitatively, the experiment aligns with our expectations. The scalar field
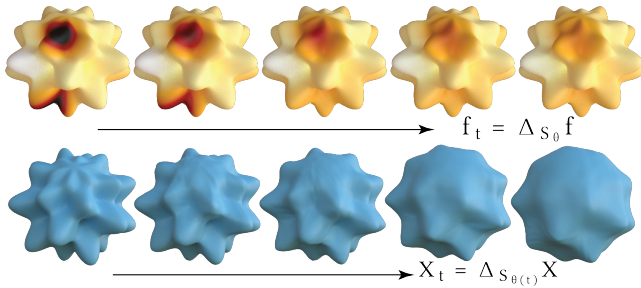
Figure 14: *Heat Flow (top) and Mean Curvature Flow (bottom). (Top) We can evolve a given scalar field, f, specified over the surface of an SNS, S_θ, using the Partial Differential Equation (PDE) $f_t = \Delta_{S_\theta} f$ (the heat equation). We represent the evolving scalar field as a small MLP, whose weights are 'finetuned' for up to 100 epochs after every time step. Darker colours denote low values (cold) with lighter colours being high (hot). (Bottom) Taking advantage of the differentiable nature of our representation, we can also compute a mesh-free Mean Curvature Flow (MCF) in which the Mean Curvature and normals are updated at every iteration (the coordinate function evolves according to the PDE $X_t = \Delta_{S_{\theta(t)}} X$). We update the Spherical Neural Surface using up to 100 finetuning steps, after every iteration of the flow. This formulation of MCF prevents singularities from forming, without any special handling to prevent them [KSBC12].*

gradually becomes more smooth over time, and cold areas warm up to a similar heat level as the surroundings.

Similarly, we approximate a Mean Curvature Flow of an SNS by updating the SNS to $S_\theta - dH\mathbf{n}$ at each time step, and finetuning for up to 100 epochs, so that the surface evolves according to the PDE $X_t = \Delta_{S_{\theta(t)}} X$ The results in Figure 14-right show the spiky analytic shape becoming closer to a sphere, over 150 iterations.

SNS   normal map   cross field   SNS   normal map   cross field
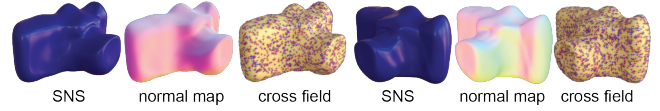
Figure 15: *Handling Neural Implicits. Spherical Neural Surfaces can be generated from other neural representations, such as DeepSDF. Here, we have produced an SNS from a DeepSDF representation for a camera, by overfitting the SNS to a low-resolution marching-cubes mesh reconstruction and then finetuning this SNS to better align with the SDF.*

Neural Implicits To demonstrate robustness to the input representation, we provide an illustrative example of an SNS generated from a DeepSDF [PFS*19], and show the normals and principal curvature directions on its SNS. We took a pretrained DeepSDF network and the optimized latent code for the camera shape [Com23]. Then, as initialization, we overfitted an SNS to a coarse marching-cubes mesh. Then we took a set of samples on the SNS and projected them to deepSDF representation defined by the signed distance function (SDF), by moving them in the direction of the gradient of the SDF (which approximates the surface normal) by the signed distance at that point, and use them to finetune the SNS.

7. Conclusion

We have presented spherical neural surfaces as a novel representation for genus-0 surfaces. Our smooth and differentiable neural surface representation inherently supports the computation of continuous differential geometric quantities and operators such as normals, First and Second Fundamental Forms), surface gradients and surface divergence. We also presented how to compute a continuous Laplace Beltrami operator and its lowest spectral modes.

We demonstrate that SNSs, by avoiding the pitfalls of discretization, produce robust and consistent differential geometric estimates for surfaces both across meshing parameters such as sampling and vertex connectivity, and when compared to alternate neural and mesh-based surface representations.

Limitations and Future Work

Beyond Genus-0 Surfaces Since our SNSs rely on a sphere for the parametrization, it limits us to genus-0 surfaces. We chose the sphere as a domain, because it allows us to process many common shapes without needing to 'cut' them into a disc-topology (as shown in, e.g., Neural Surface Maps [MAKM21]). If we could choose from a range of canonical surfaces (e.g., torus), this would allow us to seamlessly process higher genus shapes and, possibly, to also produce neural surfaces without such high distortions. Another option would be to rely on local parametrizations, but then we would have to consider blending across overlapping parametrizations.

Solving Variational Problems We are optimistic that the techniques demonstrated in our work have the potential to allow us to solve other variational problems, beyond the example of computing a spectral basis via energy minimization. We would like to know whether similar energy minimizations may be used on neural surface-paths, to compute geodesics, and on scalar- and vector-fields, to compute heat-flow/mean-curvature flow without fine-tuning the network at every step.

Speed A limitation of our current realization is its long running time. While we expect that better and optimized implementations would increase efficiency significantly, we also need to make changes to our formulation. Specifically, our current spectral estimation is sequential, which is not only slow, but leads to an accumulation of errors for later spectral modes. Hence, in the future, we would like to jointly optimize for multiple spectral modes.

SNS from Neural Representations We demonstrated our setup mainly on mesh input and also on neural input in the form of deepSDF. In the future, we would like to extend our SNS to other neural representations in the form of occupancy fields or radiance fields. However, this will require locating and sampling points on the surfaces, which are implicitly encoded – we need the representation to provide a projection operator. We would also like to support neural surfaces that come with level-of-detail. Finally, an interesting direction would be to explore end-to-end formulation for dynamic surfaces encoding temporal surfaces with SNS, and enabling optimization with loss terms involving first/second fundamental forms as well as Laplace-Beltrami operators (e.g., neural deformation energy).

References

[BAB23] BUNGE A., ALEXA M., BOTSCH M.: Discrete Laplacians for general polygonal and polyhedral meshes. In *SIGGRAPH Asia 2023 Courses* (2023), Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3610538.3614620. 2, 3, 5

[BHKB20] BUNGE A., HERHOLZ P., KAZHDAN M., BOTSCH M.: Polygon Laplacian made simple. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 303–313. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/cgforum.13931. 10

[BPG*20] BEDNARIK J., PARASHAR S., GUNDOGDU E., SALZMANN M., FUA P.: Shape reconstruction by learning differentiable surface representations. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2020). doi:10.1109/CVPR42600.2020.00477. 3

[BSLF19] BEN-SHABAT Y., LINDENBAUM M., FISCHER A.: Nesti-net: Normal estimation for unstructured 3D point clouds using convolutional neural networks. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2019), IEEE, pp. 10112–10120. URL: https://doi.org/10.48550/arXiv.1812.00709. 3

[Com23] COMI M.: DeepSDF-minimal. https://github.com/maurock/DeepSDF/, 2023. 12

[CP05] CAZALS F., POUGET M.: Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design 22*, 2 (2005), 121–146. URL: https://inria.hal.science/inria-00097582v1. 3

[CYW*23] CHETAN A., YANG G., WANG Z., MARSCHNER S., HARIHARAN B.: Accurate differential operators for hybrid neural fields, 2023. arXiv:2312.05984. 1, 3

[dC76] DO CARMO M. P.: *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. 3

[dGBD20] DE GOES F., BUTTS A., DESBRUN M.: Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics TOG 39* (2020), 110:1 – 110:14. URL: https://api.semanticscholar.org/CorpusID:221105828. 3

[DNJ20] DAVIES T., NOWROUZEZAHRAI D., JACOBSON A.: Overfit neural networks as a compact shape representation. *CoRR abs/2009.09808* (2020). URL: https://arxiv.org/abs/2009.09808, arXiv:2009.09808. 1, 2

[DPSS06] DESBRUN M., POLTHIER K., SCHRÖDER P., STERN A.: Discrete differential geometry - an applied introduction. In *ACM SIGGRAPH course notes* (2006). 2, 3

[GFK*18] GROUEIX T., FISHER M., KIM V. G., RUSSELL B., AUBRY M.: AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2018). URL: https://doi.org/10.48550/arXiv.1802.05384. 1, 2

[GKOM18] GUERRERO P., KLEIMAN Y., OVSJANIKOV M., MITRA N. J.: PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum 37*, 2 (2018), 75–85. doi:10.1111/cgforum.13343. 3

[GMJ19] GKIOXARI G., MALIK J., JOHNSON J.: Mesh R-CNN. *CoRR abs/1906.02739* (2019). URL: http://arxiv.org/abs/1906.02739, arXiv:1906.02739. 2

[KSBC12] KAZHDAN M., SOLOMON J., BEN-CHEN M.: Can mean-curvature flow be made non-singular?, 2012. arXiv:1203.6819. 12

[LZ10] LÉVY B., ZHANG H. R.: Spectral mesh processing. In *ACM SIGGRAPH 2010 Courses* (2010), SIGGRAPH '10. 3, 5

[MAKM21] MORREALE L., AIGERMAN N., KIM V. G., MITRA N. J.: Neural surface maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 4639–4648. URL: https://doi.org/10.48550/arXiv.2103.16942. 1, 3, 12

[MCR22] MEHTA I., CHANDRAKER M., RAMAMOORTHI R.: A level set theory for neural implicit evolution under explicit flows. *ECCV* (2022). 3

[MDSB02] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *International Workshop on Visualization and Mathematics* (2002). URL: https://api.semanticscholar.org/CorpusID:11850545. 3

[MON*18] MESCHEDER L. M., OECHSLE M., NIEMEYER M., NOWOZIN S., GEIGER A.: Occupancy networks: Learning 3D reconstruction in function space. *CoRR abs/1812.03828* (2018). URL: https://doi.org/10.48550/arXiv.1812.03828. 1, 2

[MST*21] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: NeRF: representing scenes as neural radiance fields for view synthesis. *Commun. ACM 65*, 1 (Dec 2021), 99–106. URL: https://doi.org/10.48550/arXiv.2003.08934. 1

[NDS*23] NSAMPI N. E., DJEACOUMAR A., SEIDEL H.-P., RITSCHEL T., LEIMKÜHLER T.: Neural field convolutions by repeated differentiation. *ACM Trans. Graph. 42*, 6 (Dec 2023). URL: https://doi.org/10.1145/3618340, doi:10.1145/3618340. 2

[NdSS*23] NOVELLO T., DA SILVA V., SCHARDONG G., SCHIRMER L., LOPES H., VELHO L.: Neural implicit surface evolution. *ICCV* (2023). 3

[NSS*22] NOVELLO T., SCHARDONG G., SCHIRMER L., DA SILVA V., LOPES H., VELHO L.: Exploring differential geometry in neural implicits. *Computers and Graphics 108* (2022), 49–60. URL: https://doi.org/10.48550/arXiv.2201.09263. 1, 2, 3, 8

[OBCS*12] OVSJANIKOV M., BEN-CHEN M., SOLOMON J., BUTSCHER A., GUIBAS L.: Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics TOG 31*, 4 (2012), 1–11. URL: https://dl.acm.org/doi/10.1145/2185520.2185526. 3

[PC24] POUGET M., CAZALS F.: Estimation of local differential properties of point-sampled surfaces. In *CGAL User and Reference Manual*, 5.6.1 ed. CGAL Editorial Board, 2024. URL: https://doc.cgal.org/5.6.1/Manual/packages.html#PkgJetFitting3. 7

[PFS*19] PARK J. J., FLORENCE P. R., STRAUB J., NEWCOMBE R. A., LOVEGROVE S.: DeepSDF: Learning continuous signed distance functions for shape representation. *CoRR abs/1901.05103* (2019). URL: https://arxiv.org/abs/1901.05103. 1, 2, 12

[PFVM20] PISTILLI F., FRACASTORO G., VALSESIA D., MAGLI E.: Point cloud normal estimation with graph-convolutional neural networks. In *2020 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* (2020), IEEE, pp. 1–6. doi:10.1109/ICMEW46912.2020.9105972. 3

[PH03] PRAUN E., HOPPE H.: Spherical parametrization and remeshing. *ACM Transactions on Graphics TOG 22*, 3 (jul 2003), 340–349. doi:10.1145/882262.882274. 3

[QSMG] QI C. R., SU H., MO K., GUIBAS L. J.: PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint*. URL: https://doi.org/10.48550/arXiv.1612.00593. 1, 2

[Rei82] REILLY R. C.: Mean curvature, the Laplacian, and soap bubbles. *The American Mathematical Monthly 89* (1982), 180–98. URL: https://api.semanticscholar.org/CorpusID:203043073. 5

[Rus07] RUSTAMOV R.: Laplace-Beltrami eigenfunctions for deformation invariant shape representation. In *Proceedings of the Symposium on Geometry Processing* (2007), Eurographics Association, pp. 225–233. URL: https://dl.acm.org/doi/10.5555/1281991.1282022. 3

[RWP06] REUTER M., WOLTER F.-E., PEINECKE N.: Laplace–Beltrami spectra as 'shape-DNA' of surfaces and solids. *Computer-Aided Design 38*, 4 (2006), 342–366. Symposium on Solid and Physical Modeling 2005. URL: https://doi.org/10.1016/j.cad.2005.10.011. 3

[SC20] SAWHNEY R., CRANE K.: Monte carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph. 39*, 4 (aug 2020). URL: https://dl.acm.org/doi/abs/10.1145/3386569.3392374. 2

[SOG09] SUN J., OVSJANIKOV M., GUIBAS L.: A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum 28*, 5 (2009), 1383–1392. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2009.01515.x. 3

[SPK23] SCHMIDT P., PIEPER D., KOBBELT L.: Surface maps via adaptive triangulations. *Computer Graphics Forum 42*, 2 (2023). URL: https://doi.org/10.1111/cgforum.14747. 3, 11

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, Association for Computing Machinery, p. 351–358. URL: https://doi.org/10.1145/218380.218473, doi:10.1145/218380.218473. 3

[WMKG07] WARDETZKY M., MATHUR S., KÄLBERER F., GRINSPUN E.: Discrete Laplace operators: No free lunch. vol. 07, pp. 33–37. URL: https://doi.org/10.2312/SGP/SGP07/033-037. 2

[WMVB23] WALKER T., MARIOTTI O., VAXMAN A., BILEN H.: Explicit neural surfaces: Learning continuous geometry with deformation fields, 2023. arXiv:2306.02956. 3

[WZL*18] WANG N., ZHANG Y., LI Z., FU Y., LIU W., JIANG Y.: Pixel2mesh: Generating 3D mesh models from single RGB images. *CoRR abs/1804.01654* (2018). URL: http://arxiv.org/abs/1804.01654. 2

[XZ03] XU J., ZHAO H.: An Eulerian formulation for solving partial differential equations along a moving interface. *Journal of Scientific Computing 19* (2003), 573–594. doi:10.1023/A:1025336916176. 5

[YBHK21] YANG G., BELONGIE S., HARIHARAN B., KOLTUN V.: Geometry processing with neural fields. In *Thirty-Fifth Conference on Neural Information Processing Systems* (2021). URL: https://dl.acm.org/doi/10.5555/3540261.3541983. 1, 2, 3, 7