DOI: 10.1111/cgf.70354

EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Mesh Processing Non-Meshes via Neural Displacement Fields

Yuta Noma[1], Zhecheng Wang[1], Chenxi Liu[1], Karan Singh[1], Alec Jacobson[1,2]

[1]University of Toronto, [2]Adobe Research

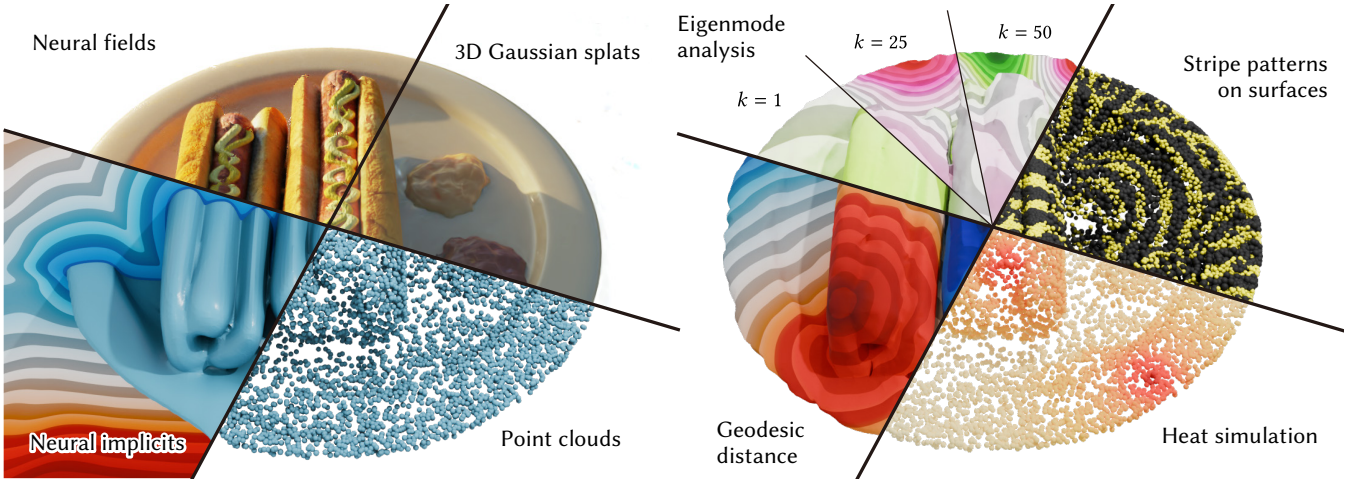


Figure 1: *We propose a compact neural map that allows fast and accurate computation of geometry processing tasks on non-mesh surfaces. Our method supports various surface representations, including neural implicits, point clouds, and 3D Gaussian splats.*

Abstract

Mesh processing pipelines are mature, but adapting them to newer non-mesh surface representations—which enable fast rendering with compact file size—requires costly meshing or transmitting bulky meshes, negating their core benefits for streaming applications.

We present a compact neural field that enables common geometry processing tasks across diverse surface representations. Given an input surface, our method learns a neural map from its coarse mesh approximation to the surface. The full representation totals only a few hundred kilobytes, making it ideal for lightweight transmission. Our method enables fast extraction of manifold and Delaunay meshes for intrinsic shape analysis, and compresses scalar fields for efficient delivery of costly precomputed results. Experiments and applications show that our fast, compact, and accurate approach opens up new possibilities for interactive geometry processing.

CCS Concepts

• ***Computing methodologies*** → ***Shape analysis; Mesh models; Mesh geometry models;***

1. Introduction

Neural fields have emerged as powerful representations for visualizing 3D geometry. Neural Signed Distance Fields (SDFs) and Neural Radiance Fields (NeRFs) enable high-fidelity rendering at interactive frame rates [MESK22] while being significantly more compact than traditional representations [TMND*23, LSW*22]. The interactive speed and compactness are especially valuable for streaming applications like games and interactive apps, where the client must download data from the server and render it in real time.

Despite the impact of neural fields in rendering, the whole area of

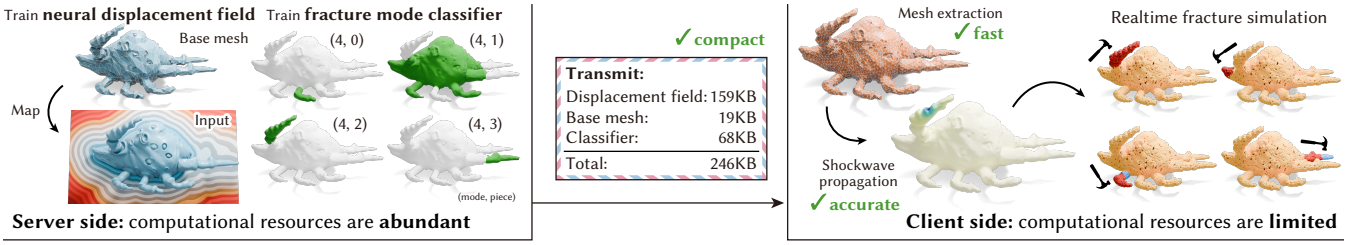


Figure 2: *One application of our method is realtime fracture simulation [SLMDS*23]. On the server side, we train the neural displacement field and the fracture mode classifier using our method. The neural networks are compact enough (246KB) to be sent to the client side, where we run mesh extraction and a sparse Cholesky factorization in under a second. This allows realtime fracture simulation that can be done in milliseconds.*

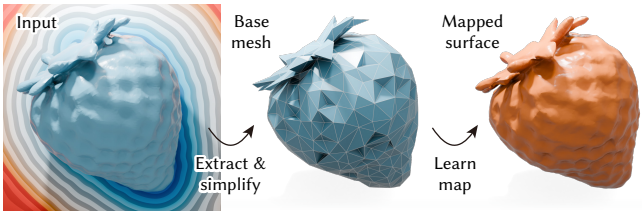


Figure 3: *Given an input neural implicit surface, we extract a* base mesh *that coarsely approximates the input surface, and learn a* neural displacement field *that maps a base mesh to the input surface.*

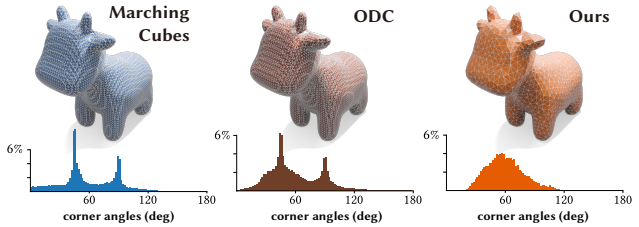


Figure 4: *Our mesh is free of thin sliver triangles unlike isosurfacing methods [LC87, HS24], leading to accurate and efficient computation on the mesh.*

geometric computing spans beyond visualizing surfaces. In particular, intrinsic shape analysis, such as geodesic computation or Laplacian geometry processing, is central to applications like physics-based animation. However, these pipelines typically rely on manifold and Delaunay triangle meshes for robust processing. Applying them to neural surface representations requires mesh extraction: either on the client side, involving heavy remeshing algorithms, or on the server side, necessitating transmission of bulky meshes.

We propose a method tailored for geometry processing in streaming applications, where server resources are abundant but client compute and bandwidth are limited. Our goal is to minimize computational and transmission burdens on the client, while retaining high accuracy in geometry processing tasks. To this end, we learn a *neural displacement field* on the server side that is both compact and fast to infer. This field maps a coarse mesh approximation—referred to as the *base mesh*—to the input surface (Figure 3). Inspired by displacement maps for surface multigrid methods [LZBCJ21, GGH02, LSS*98, JSZP20, JZH*21, LJG*24], the displacement field acts as a quasi one-to-one correspondence between the base mesh and the implicit surface. This enables fast, accurate extraction of manifold and Delaunay meshes on the client side. Additionally, our method compresses scalar fields into compact neural representations, enabling heavy preprocessing to be offloaded to the server while keeping transmission overhead minimal.

Another key advantage of our method is its representation-agnostic design. It supports a wide range of shape representations: not only neural implicit surfaces, but also oriented point clouds and SDFs, with partial support for NeRFs, Gaussian splats, and non-oriented point clouds. This flexibility enables visualization using the original representation, while leveraging our extracted mesh for robust geometric computation.

Figure 2 shows the overview of our pipeline where:

1. the server side trains a neural displacement field and a scalar field that requires heavy computation,
2. the compact MLPs and the base mesh are transmitted to the client side, and
3. the client side subdivides the mesh and runs the simulation in an interactive runtime.

We demonstrate that our fast, compact, and accurate method enables new types of interactive streaming applications on non-mesh surfaces, including geodesic computation, Laplacian-based processing, fracture simulation, surface sampling, and more.

2. Related Work

We discuss three alternative methods for our pipeline: mesh extraction on the client side, mesh extraction on the server side, and meshless methods.

2.1. Mesh extraction on the client side

To perform geometry processing tasks on neural surfaces, one solution is to extract the triangle mesh on the client side. This approach can avoid transmitting bulky meshes over the internet. Although there are hundreds of optimization-based mesh extraction methods [dALJ*15, HWW*24, BTS*17], here we only focus on

Figure 5: *Meshes obtained by Marching Cubes [LC87] suffer from thin sliver triangles, leading to catastrophic failure in analyzing intrinsic properties even after intrinsic Delaunay refinement [SSC19]. In contrast, meshes obtained using our method qualitatively preserve the pattern even with a significantly smaller number of vertices.*

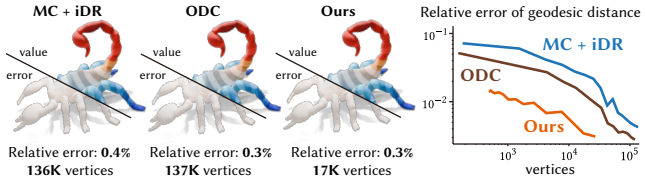


Figure 6: *Our method creates a mesh that is accurate in computing geodesic distances with a significantly smaller number of vertices than isosurfacing methods [HS24, LC87].*

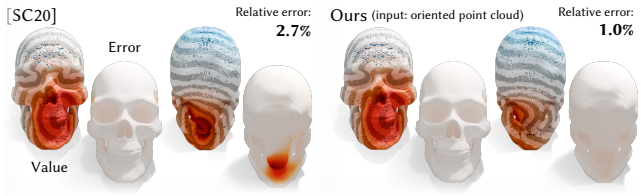


Figure 7: *Unlike the point cloud Laplacian of Sharp and Crane [SC20b], our mesh can distinguish the connectivity of thin parts, leading to accurate geodesic computation.*

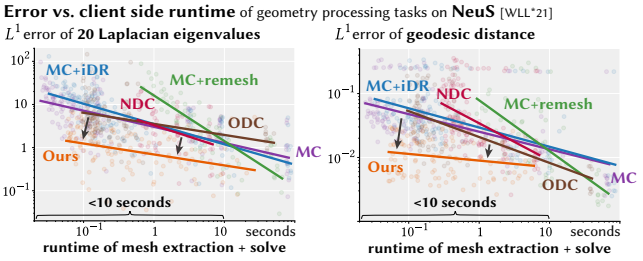


Figure 8: *One alternative to our method is to extract the mesh on the client side using isosurfacing methods ‡, which has a trade-off between speed and accuracy. By additionally transmitting our representation, which is only 174KB in this setup, we extend the Pareto frontier towards the bottom left of the plot, especially where the runtime is below 10 seconds.*

methods that do not require additional training or optimization on the client side, allowing mesh extraction in seconds.

Thanks to their robustness and simplicity, Marching Cubes [LC87] or Dual Contouring [JLSW02] or their variants [DK91, SW04, LB03, SJW07] often serve as the final component in surface reconstruction pipelines. Recently, several isosurfacing approaches have been proposed to leverage learned priors from a large dataset [CZ21, CTFZ22, SKO24], while another method harnesses the behavior of the underlying implicit function [HS24]. Also, Sharp and Crane [SC20b] find local connectivity of a point cloud with a simple proximity search.

However, most of these meshes have thin, poor-quality triangles (Figure 4), inducing error in eigenvalue analysis (Figure 5) or leading to longer runtime due to the excessive number of vertices (Figure 6). Furthermore, many methods miss the local connectivity of the surface, leading to larger errors in geodesic computation (Figure 7). Finally, some isosurfacing methods (e.g., NDC [CTFZ22]) produce non-manifold edges for most cases, making the numerical instability even worse.

Another tempting approach is to remesh poor quality meshes using, e.g., isotropic remeshers [BK04] or intrinsic Delaunay triangulation [SSC19]. However, the edge flip p rocedures i n these methods are mostly $\mathcal{O}(n \log n)$, leading to notoriously long runtimes due to the excessive number of vertices in a bad quality mesh. Furthermore, such edge flip procedures are not easily parallelizable [CNGT14].

In Figure 8, we show the time and error trade-off of the existing approaches. Raw isosurfacing methods (MC, ODC) or neural-powered methods (NDC) are fast but prone to higher error. Using an isotropic remesher [BK04] decreases the error sharply but suffers from a longer runtime to remesh. In contrast, our representation can extract manifold and Delaunay meshes with computationally cheap operations, leading to faster runtime with lower error.

2.2. Mesh extraction on the server side

Another solution is to extract the mesh on the server side and transmit it to the client. As meshes are often bulky, significant research has focused on mesh compression [MLDH15]. Early approaches include lossless schemes that preserves the triangle connectivity [AD01, Dee95, Ros01, SKR01, TR98, TG98, Goo17] and surface simplification techniques [GH97, SG03, SRK02]. However, their final compressed file size is always proportional to the number of vertices on the mesh, causing a trade-off between file size and feature preservation.

Instead, recent state-of-the-art methods decompress meshes by

‡ **MC**: Marching Cubes [LC87], **ODC**: Occupancy-Based Dual Contouring [HS24], **NDC**: Neural Dual Contouring [CTFZ22], **iDR**: intrinsic Delaunay refinement [SSC19], **remesh**: isotropic remeshing [BK04].

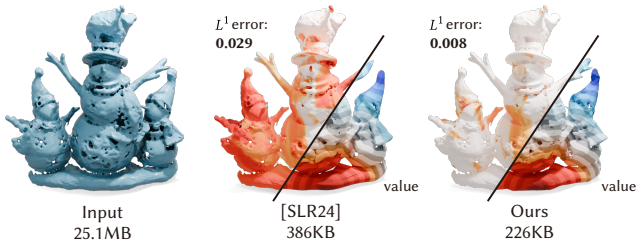


Figure 9: *A state-of-the-art mesh compression method [SLR24] cannot keep the accuracy of the geodesic distance. In contrast, our method achieves a 3.6x error reduction with a smaller file size.*

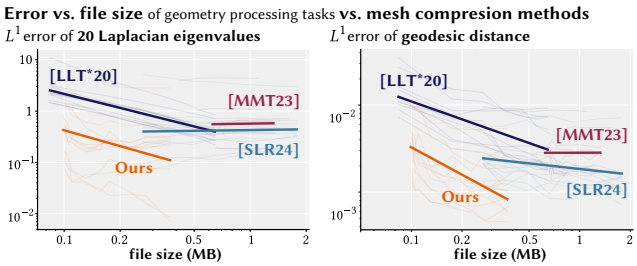


Figure 10: *Another alternative to our method is to extract a mesh on the server side and transmit it to the client side, using surface simplification methods [LLT*20] or state-of-the-art mesh compression methods [SLR24, MMT23]. We extend the Pareto frontier towards the bottom left of the plot, especially where the file size is below 500KB.*

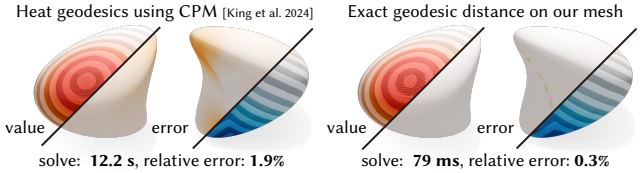


Figure 11: *Running the Closest Point Method [KSA*24] on the client side suffers from not only longer computation time but also larger error. Ours, while requiring training on the server side, is significantly faster and more accurate on the client side.*

subdividing a coarse base mesh, inspired by displacement map generation techniques [LZBCJ21, GGH02, LSS*98, JSZP20, JZH*21, LJG*24]. Recent methods guide their mesh subdivision by a learned prior [CKAJ23, LKC*20], a per-vertex displacement vector [MMT23, DZJ*24], a neural field that minimizes differentiable rendering losses [SLR24], and a neural field overfitted to a self-parametrization [PPB25]. However, these methods are primarily designed to preserve visual appearance and often fail to retain local geometric connectivity, which is critical for intrinsic shape analysis (Figure 9). Furthermore, these methods are not robust to many types of inputs; for example, [PPB25] does not accept non-manifold meshes. Most importantly, non-mesh representations must first be converted to meshes, introducing additional discretization error before compression even begins (Figures 10).

We argue that the true value of a mesh lies in its utility for computation. Triangle meshes provide a convenient finite-element basis for solving PDEs, and are supported by a rich ecosystem of software libraries (e.g., CGAL [CGA24], LIBIGL [JP*18]). Prior work has explored mesh simplification tailored to preserve specific computational properties—for instance, Otaduy and Lin [OL03] focused on preserving haptic features, while Lescoat et al. [LLT*20] targeted spectral characteristics. As shown in Figure 10, our method significantly outperforms Lescoat et al. [LLT*20], achieving lower geodesic and spectral error with smaller file sizes. This reinforces the idea that compact representations can still enable high-quality computation.

Furthermore, our method accomodates desirable features for interactive applications, which mere mesh compression methods do not have. One is its ability for progressive refinement, which allows users to balance their mesh resolution based on their computational budget (Section 3.4). Unlike Chen et al. [CKAJ23], the client side can determine the mesh resolution without requiring additional data to be transmitted. Another is compressing scalar fields on surfaces (e.g., the fracture mode classifier in Figure 2), enabling heavy preprocessing to be offloaded to the server. Finally, we support several useful extensions and applications, which we explain in Section 5.2.

2.3. Meshless methods

The third category circumvents the need for high-quality meshing and directly solves the problem on the surface.

Many works use neural networks to solve their geometry processing tasks, akin to our approach. Several early attempts try to derive the local properties like normal directions and curvatures on non-oriented point clouds [GKOM18, BSLF19, PFVM20, BPG*20] or accurately derive these properties on neural SDFs [CYW*23, NSS*22]. In order to support a broader set of tasks, several works [YBHK21, WM25] use neural networks to represent a vector field on a surface and solve various geometry processing problems by regressing a loss. However, these works require additional training to run their task, making it infeasible to add interactive control on the client side (e.g., changing the position of the boundary condition). Other methods represent surface maps [MAKM21] or the Laplace-Beltrami operator [PZL*24] using a neural network, but these works require the ground truth value computed on a discretized mesh, and little is said about inference time or compression.

Alternatively, Monte Carlo approaches [SC20a] or the closest point method (CPM) [KSA*24, LOW*23, SKHB24] bypass the need for discretizing the boundary domain. However, naive Monte Carlo approaches require a precomputed conformal parametrization [SSJC22, Section 6.5], while CPM-based approaches fall short due to extremely slow computation on the client side (see Figure 11).

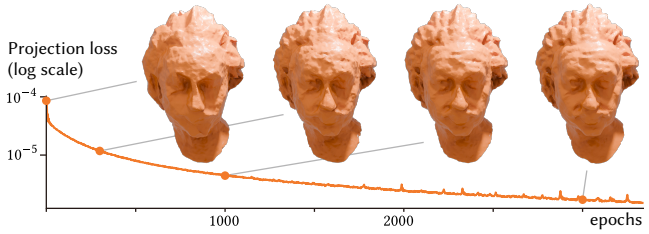


Figure 12: Ω_θ *learns to approximate* Ω *as training proceeds.*

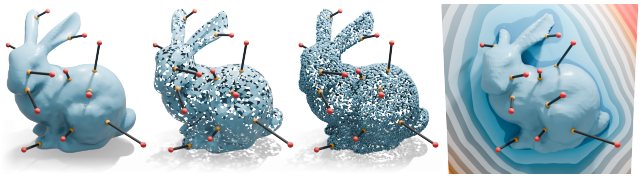


Figure 13: *Our projection operator* $\mathcal{P}$ *supports various surface representations, such as (left to right) meshes, polygon soups, point clouds, or neural occupancy functions where the Eikonal equation is not necessarily met (right).*

3. Method

The input of our method is a target surface $\Omega \subset \mathbb{R}^3$ represented as a non-mesh. Our output will be (1) a manifold base mesh $\Omega' \subset \mathbb{R}^3$ that approximates the shape of Ω and (2) a *neural displacement field* $g_\theta : \mathbb{R}^3 \mapsto \mathbb{R}^3$ parametrized by weights θ, learned to approximate a map $f : \Omega' \mapsto \Omega$.

To achieve this, we extract a base mesh using existing procedures (Section 4.1), coarsen it to a target face number $N_{\Omega'}$ using QEM [GH97], and finally learn g_θ. The compressibility relies on $N_{\Omega'}$, and thus one may change this to trade off file size and accuracy. Also, we use a fairly small MLP to represent g_θ (64 neurons with 2 hidden layers), allowing inference to be done in several milliseconds.

We also use an intrinsic encoding scheme that further enhances the accuracy of our neural displacement field (Section 3.3). Using the base mesh Ω' and the neural displacement field g_θ, one may extract the final mesh with cheap computation (Section 3.4), or learn scalar fields on surfaces (Section 3.5).

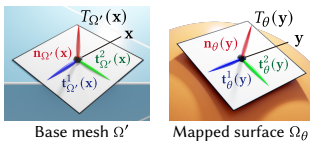
In what follows, let $\mathbf{n}_{\Omega'}(\mathbf{x}) \in \mathbb{R}^3$ be the normal vector at $\mathbf{x} \in \Omega'$, $T_{\Omega'}(\mathbf{x})$ be the tangent plane at $\mathbf{x} \in \Omega'$, and $\mathbf{t}^1_{\Omega'}(\mathbf{x}), \mathbf{t}^2_{\Omega'}(\mathbf{x}) \in \mathbb{R}^3$ be the orthogonal tangent vectors spanning $T_{\Omega'}(\mathbf{x})$. Please refer to the supplemental material for additional technical details.

3.1. Querying Geometric Quantities

The mapped surface $\Omega_\theta := g_\theta(\Omega')$ has the same topology as Ω', and as the training proceeds, Ω_θ approximates Ω (Figure 12). Before showing how to train the neural displacement field g_θ, let us explain how we can access to important geometric quantities on Ω_θ, which our computation of g_θ relies on.

3.1.1. Normals and tangent vectors

Given a point $\mathbf{x} \in \Omega'$, one useful quantity on Ω_θ is the unit normal vector at $\mathbf{y} = g_\theta(\mathbf{x}) \in \Omega_\theta$ which we denote as $\mathbf{n}_\theta(\mathbf{y}) \in \mathbb{R}^3$. With the normal vector $\mathbf{n}_\theta(\mathbf{y})$ in hand, we may also define the tangent plane $T_\theta(\mathbf{y})$ and two randomly selected unit orthogonal vectors $\mathbf{t}^1_\theta(\mathbf{y}), \mathbf{t}^2_\theta(\mathbf{y}) \in \mathbb{R}^3$ spanning it (see inset).



Let

$$\mathbf{J}_\theta(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} g_\theta(\mathbf{x}) \in \mathbb{R}^{3\times 3} \tag{1}$$

be the Jacobian of g_θ at $\mathbf{x}$. Then, the tangent plane on $\mathbf{y}$ is spanned by two vectors $\mathbf{J}_\theta(\mathbf{x})\mathbf{t}^1_{\Omega'}(\mathbf{x})$ and $\mathbf{J}_\theta(\mathbf{x})\mathbf{t}^2_{\Omega'}(\mathbf{x})$, leading to

$$\mathbf{n}_\theta(\mathbf{y}) = \frac{\mathbf{J}_\theta(\mathbf{x})\mathbf{t}^1_{\Omega'}(\mathbf{x}) \times \mathbf{J}_\theta(\mathbf{x})\mathbf{t}^2_{\Omega'}(\mathbf{x})}{\|\mathbf{J}_\theta(\mathbf{x})\mathbf{t}^1_{\Omega'}(\mathbf{x}) \times \mathbf{J}_\theta(\mathbf{x})\mathbf{t}^2_{\Omega'}(\mathbf{x})\|_2}. \tag{2}$$

3.1.2. Local Jacobians

Viewing g_θ as a map from Ω' to Ω_θ, we can also define the *pushforward* $dg^{\mathbf{x}}_\theta : T_{\Omega'}(\mathbf{x}) \mapsto T_\theta(\mathbf{y})$ which defines how the local neighborhood at $T_{\Omega'}(\mathbf{x})$ is mapped to the local neighborhood at $T_\theta(\mathbf{y})$. The pushforward can be described as a mapping from a local coordinate space spanned by $\mathbf{t}^1_{\Omega'}, \mathbf{t}^2_{\Omega'}$ to one spanned by $\mathbf{t}^1_\theta, \mathbf{t}^2_\theta$, where the transformation can be given as a 2×2 matrix

$$\mathbf{J}_f(\mathbf{x}) = \left[\mathbf{t}^1_\theta(\mathbf{y}), \mathbf{t}^2_\theta(\mathbf{y})\right]^T \mathbf{J}_\theta(\mathbf{x}) \left[\mathbf{t}^1_{\Omega'}(\mathbf{x}), \mathbf{t}^2_{\Omega'}(\mathbf{x})\right] \tag{3}$$

which we call the *local Jacobian* matrix.

3.2. Loss Function

Our loss function contains two terms, which can all be computed using the same s samples $\mathbf{x}_1, \ldots, \mathbf{x}_s \in \Omega'$ sampled on the base mesh Ω'. In order to integrate the loss on Ω_θ instead of Ω', we draw them such that the distribution of the points on Ω' is proportional to $\left|\det \mathbf{J}_f(\mathbf{x})\right|$ using rejection sampling. See the supplemental material for other optional losses and training details.

To make our neural displacement field work as a quasi 1-to-1 mapping between the base mesh and the fine surface, we also learn the inverse neural map g_ϕ parametrized by weights ϕ. The inverse map is only used for training, and thus does not need to be sent to the client side.

3.2.1. Projection loss

The first loss ensures that the mapped surface Ω_θ matches the input surface Ω. In order to compute the loss, the surface representation of Ω must be able to define a *projection operator* $\mathcal{P}(\mathbf{p})$ (Figure 13) that takes a point $\mathbf{p} \in \mathbb{R}^3$ as input and returns

$$\mathcal{P}(\mathbf{p}) = \begin{cases} \mathbf{p} & (\mathbf{p} \in \Omega), \\ \mathbf{q} \in \Omega & (\mathbf{p} \notin \Omega). \end{cases} \tag{4}$$

We will discuss later in Section 4 on how to perform this projection on various surface representations.

We want $\mathcal{P}(g_\theta(\mathbf{x}))$ to match $g_\theta(\mathbf{x})$ for every $\mathbf{x} \in \Omega'$, and thus we

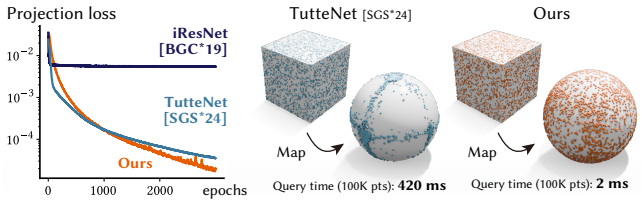


Figure 14: *Invertible neural networks [BGC*19] do not have enough expressivity to learn even the simplest map from a cube to a sphere (left). TutteNet [SGS*24] can learn this map, but the map induces large distortions which makes mesh extraction infeasible, and the query time is slow (middle).*

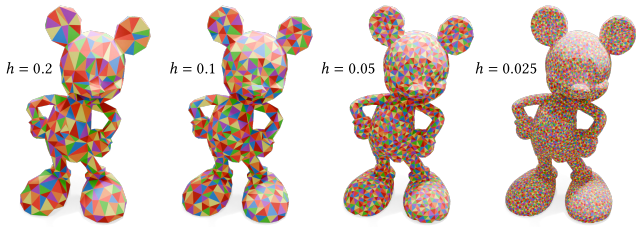


Figure 15: *Our neural displacement field can progressively refine the mesh, allowing users to trade off runtime and accuracy in their computation tasks.*

add the following loss

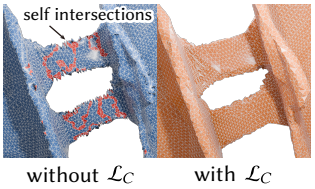
$$\mathcal{L}_P(\theta) = \frac{1}{s}\sum_{i=1}^{s} \|g_\theta(\mathbf{x}_i) - \mathcal{P}(g_\theta(\mathbf{x}_i))\|_2^2. \tag{5}$$

3.2.2. Cycle consistency loss

The second loss encourages g_ϕ to be the inverse of g_θ. This can be encouraged by adding the following loss

$$\mathcal{L}_C(\theta,\phi) = \frac{1}{s}\sum_{i=1}^{s} \|g_\phi(g_\theta(\mathbf{x}_i)) - \mathbf{x}_i\|_2^2. \tag{6}$$

We also found that this loss contributes to having a smoother surface free of self-intersections (see inset).



We opt for regressing a loss to encourage cycle consistency in a weak sense. This design choice is because existing injective neural networks are either not expressive enough [BGC*19] or slow [SGS*24] (see Figure 14). Although cyclic consistency is not guaranteed, we observed that this loss becomes sufficiently low ($< 10^{-6}$) and can achieve high accuracy in computational tasks as we explain in Section 5.

3.2.3. Total loss

Given all the losses at hand, we now define our final optimization problem

$$\underset{\theta,\phi}{\operatorname{argmin}}\ \lambda_C\mathcal{L}_C(\theta,\phi) + \lambda_P\mathcal{L}_P(\theta), \tag{7}$$

where λ_C, λ_P are hyperparameters.

3.3. Neural Network Architecture

We model the residual of our neural displacement field $g_\theta(\mathbf{x}) - \mathbf{x}$ and its inverse $g_\phi(\mathbf{x}) - \mathbf{x}$ using two MLPs with 2 hidden layers, 64 internal hidden units, and ReLU activation functions. We also applied periodic positional encoding [MST*20] with 8 frequencies.

For the neural displacement field g_θ, we also apply an intrinsic encoding that involves learnable d-dimensional feature vectors assigned to each vertex. This helps to learn scalar or vector fields on the surface with finer details (see Section 3.5). The idea itself is similar to previous mesh compression works [SLR24], but ours is a more general representation that extends naturally to base triangle meshes instead of quad meshes.

Suppose that a point $\mathbf{x}_i \in \Omega'$ lies on a face that contains vertices $\mathbf{v}_i^1, \mathbf{v}_i^2, \mathbf{v}_i^3 \in \mathbb{R}^3$ on the base mesh. For each vertex, we have a learnable d-dimensional feature vector $\mathbf{u}_i^1, \mathbf{u}_i^2, \mathbf{u}_i^3 \in \mathbb{R}^d$. Let $[w_i^1, w_i^2, w_i^3]$ be the barycentric coordinate of $\mathbf{x}_i$, we interpolate the feature vectors as

$$\mathbf{u}_i = w_i^1\mathbf{u}_i^1 + w_i^2\mathbf{u}_i^2 + w_i^3\mathbf{u}_i^3 \in \mathbb{R}^d \tag{8}$$

and feed this vector to the MLP in addition to the positional encoding vectors.

3.4. Mesh Extraction

Given the neural displacement field g_θ, one may extract a mesh that is suitable for geometry processing. We achieve this by subdividing the base mesh and mapping all the vertices to the target surface Ω_θ using g_θ.

Given a user-specified parameter h (Figure 15), we subdivide the edge on the base mesh Ω' at the midpoint if the mapped edge is longer than h, and remove one of its incident vertices if it is shorter than $h/4$. In order to retain the geometry of the base mesh, we allow vertex removal only for the ones that were added during midpoint subdivision. We also conduct (extrinsic) edge flips if it reduces the ODT energy [CX04], which is the sum of the squared edge lengths weighted by its incident triangle areas. To keep the bijection between the base mesh and the mapped mesh, we only allow edge flips if the edge can be flipped intrinsically (see Sharp et al. [SSC19] Section 4.1). We iterate these three procedures 5 times, and after that we run intrinsic Delaunay triangulation [SSC19] to ensure the Delaunay condition. The subdivided mesh is homeomorphic to the base mesh, and has a one-to-one correspondence with the base mesh. As the base mesh is guaranteed to be manifold, the subdivided mesh is also guaranteed to be manifold.

This mesh can be used for geometry processing tasks such as computing geodesic distances or Laplacian geometry processing. The solution computed on the mesh can be queried by simply taking the closest point query from the fine surface to the subdivided mesh and barycentrically interpolating the per-vertex value. This can be combined with rendering algorithms that get the explicit ray intersection position with the surface (e.g., sphere tracing).

Table 1: *We support various surface representations.*

Surface Representation	(1) Base mesh Ω' ▶ Section 4.1	(2) Projection $\mathcal{P}(\mathbf{p},\mathbf{d})$ ▶ Section 4.2
Analytic SDFs	Marching Cubes	SDF projection
Grid-interpolated implicits	Marching Cubes	Newton's method
Neural implicits/NeuS	Marching Cubes	Newton's method
Polygon soups/ Oriented point clouds	Marching Cubes on the Generalized Winding Number (GWN) field [BDS*18]	Ray tracing the GWN field using Harnack Tracing [GYBC24]
NeRFs (partial support)	Marching Cubes on the TSDF [ZBS*22]	Ray tracing the density field
Non-oriented point clouds/Gaussian splats (partial support)	Dilation	Closest point query

Akin to progressive physics simulation methods [ZJK24], one may also progressively refine the mesh to have a *preview* at earlier runtime and a high-quality result later (Figure 15). This allows the client application to trade off the computational budget and accuracy based on their demand.

3.5. Learning Scalar Fields

Given the trained neural displacement field g_θ, we may also train a neural network to represent a scalar field on the surface Ω_θ. This is especially useful when one wants to compress a precomputed scalar field that is expensive to compute on the client side, like the object's fracture modes (Figure 2).

Let $u : \Omega_\theta \mapsto \mathbb{R}$ be a scalar field computed on the mesh of Ω_θ. We overfit a neural network $u_\varphi : \Omega' \mapsto \mathbb{R}$ such that u_φ approximates $u \circ g_\theta$. As u_φ is a scalar field defined on the coarse mesh Ω', we use the encoding scheme we introduced in Section 3.3 to parametrize the function intrinsically to the coarse mesh.

Let $\mathbf{z}_1, \ldots, \mathbf{z}_s$ be uniformly sampled points on the mesh approximating Ω. If u is a continuous function, we regress a binary cross entropy loss

$$\mathcal{L}_\varphi = \frac{1}{s}\sum_i^s \|u_\varphi(\mathbf{z}_i) - u(g_\theta(\mathbf{z}_i))\|^2. \tag{9}$$

Likewise, if u gives a binary value, we regress

$$\mathcal{L}_\varphi = -\frac{1}{s}\sum_i^s \mathcal{L}_{\mathrm{BCE}}(u_\varphi(\mathbf{z}_i), u(g_\theta(\mathbf{z}_i))), \tag{10}$$

where $\mathcal{L}_{\mathrm{BCE}}(x,y) = y\ln\big(1/(1+e^{-x})\big) + (1-y)\ln\big(e^{-x}/(1+e^{-x})\big)$.

4. Supporting Various Representations

Our method requires the surface representation to be able to (1) extract the base mesh Ω' and (2) define a projection operator $\mathcal{P}$. We outline the supported surface representations in Table 1.

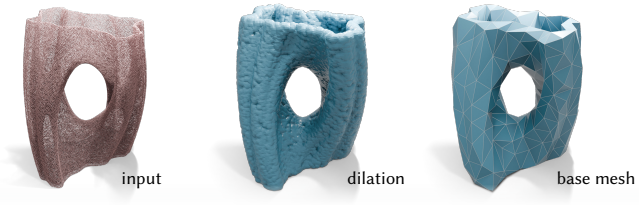


Figure 16: *On a non-oriented point cloud where the occupancy function is unavailable, one may dilate the points for a certain radius by adding an offset to the distance function on a regular grid.*

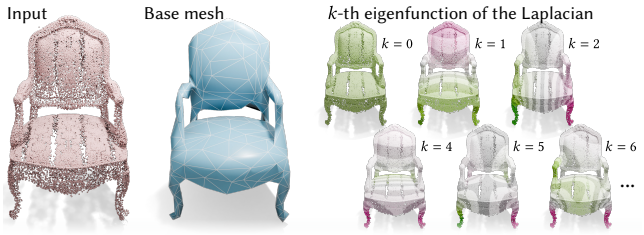


Figure 17: *Our method partially supports 3D Gaussian splats [KKLD23] by treating them as a volumetric point cloud. The mapped mesh is clean enough to have a positive definite Laplacian, accommodating applications like eigenmode analysis.*

4.1. Extracting Base Mesh Ω'

Most surface representations support inside/outside occupancy functions, like **analytic SDFs**, **grid- interpolated implicits**, **neural implicits** [PFS*19], **NeuS** [WLL*21], and the Generalized Winding Number [JKSH13, BDS*18] for **oriented point clouds**. For such cases, we apply Marching Cubes [LC87] on an upsampled grid and coarsen it with surface simplification algorithms [GH97].

NeRFs, **Non-oriented point clouds**, or **3D Gaussian splats** do not have a definitive boundary of the occupancy function, and we thus only claim partial support. However, one may still use an existing method that estimates the occupancy (e.g., TSDF [ZBS*22] for NeRFs) or dilate a point cloud and coarsen it (Figure 16). We also note that several neural representations have a well-behaved occupancy function for novel view synthesis data (e.g., NeuS [WLL*21]).

4.2. Projection Operator $\mathcal{P}$

Our second requirement for Ω is the ability to define the projection operator (Eq. 4).

4.2.1. Implicit functions

For shapes represented as the zero level set of an implicit function $\phi : \mathbb{R}^3 \mapsto \mathbb{R}$ including **neural implicits**, **NeuS**, and **grid-interpolated implicits**, we use Newton's method that undergoes an iterative procedure

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \phi(\mathbf{p}_i)\frac{\nabla\phi(\mathbf{p}_i)}{\|\nabla\phi(\mathbf{p}_i)\|^2},\ \mathbf{p}_0 = \mathbf{p}, \tag{11}$$

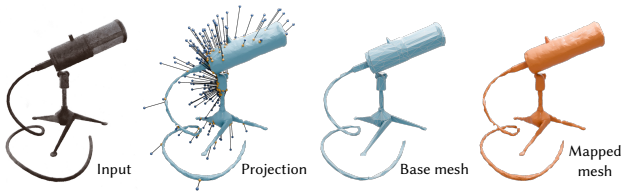


Figure 18: *Our method partially supports Neural Radiance Fields [MST*20]. The projection is done by shooting a ray towards a direction* **d** *and getting its first hit point with the levelset of the density function.*

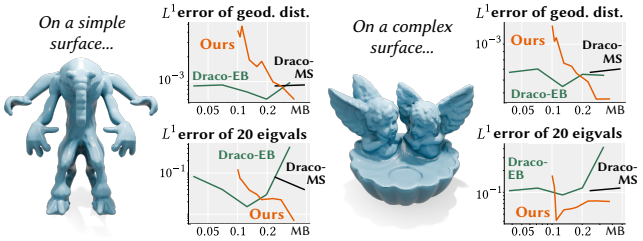


Figure 19: *For file sizes of $>$ 250KB, our method outperforms Draco [Goo17] combined with mesh simplification [GH97] that was run on a high resolution marching cube mesh (400^3 grid). Our win increases on complex implicit surfaces with intricate features (right), as the discretization error becomes more significant.* ***Draco-EB*** *uses the mesh encoding scheme in Edgebreaker [Ros99], while* ***Draco-MS*** *uses a less efficient sequential connectivity encoding.*

which is also used for sampling level sets on neural implicits [AHY*19].

4.2.2. Closest point queries

A closest point query to Ω can be directly used as our $\mathcal{P}$, an operator commonly required in representation-agnostic PDE solvers [SC20a, KSA*24]. For **analytic SDFs** $\phi_S : \mathbb{R}^3 \mapsto \mathbb{R}$, the projection of $\mathbf{p}$ is obtained by $\mathbf{p} - \phi_S(\mathbf{p})\nabla\phi_S(\mathbf{p})$ [MSLJ23], which we call SDF projection. For **non-oriented point clouds** and **3D Gaussian splats** sampled on Ω, we take the closest point in the point cloud or the center positions of the Gaussians (see Figure 17).

For **oriented point clouds** or **polygon soups**, we use Harnack tracing [GYBC24] and terminate the ray where the Generalized Winding Number (GWN) equals 0.5 [JKSH13]. To get the ray direction, we apply a Gaussian kernel to the normal vectors of the eight nearest neighbors of $\mathbf{p}$ in the point cloud and negate them if the GWN is smaller than 0.5. This is similar in spirit to the vector field definition in Poisson Surface Reconstruction [KBH06, Section 4.2]. These adjustments will provide a more accurate estimate of the closest point to the surface than a simple nearest neighbor search (see the supplemental material for details). We note that this projection operator may not converge, and in such a case, we simply omit the nonconvergent samples and only use samples that succeed.

4.2.3. Ray traceable geometry

For **Neural Radiance Fields (NeRFs)** [MST*20], we shoot two rays $\mathbf{p} + t\mathbf{n}_\theta(\mathbf{p})$ and $\mathbf{p} - t\mathbf{n}_\theta(\mathbf{p})$ to get the closest intersection point, where $\mathbf{n}_\theta(\mathbf{p})$ is the normal direction of Ω_θ at $\mathbf{p}$. The ray intersection is where the density function equals a threshold (Figure 18). In our implementation, we naively sample points uniformly along the ray to find the first intersection.

5. Results

In this section, we report the core results of our method in terms of speed, compressibility, accuracy, and versatility. Please see the supplemental material for other details such as baseline setup, ablations, hyperparameters, and extensions.

5.1. Experiments and Ablations

5.1.1. Mesh extraction

Our mesh extraction tackles the multiobjective problem regarding (1) runtime, (2) file size, and (3) error of computation tasks (geodesics/eigenvalue analysis). As there are three axes, it is difficult to project all baseline methods on a single 2D plot and do a fair comparison. We thus show three versions of plots: comparison with fast mesh extraction methods (Figure 8), with mesh compression methods (Figure 10), and a 3D plot with all of the axes (see the attached HTML file in the supplemental material).

Ideally, one would want shorter runtime, compact file size, and lower error in their computation tasks. In our plots, we show that our method pushes the Pareto frontier towards this goal. For example, our method outperforms existing mesh extraction methods in regions where the runtime is below 10 seconds (Figure 8) . Likewise, our method outperforms state-of-the-art mesh compression methods in the range of $\sim$500KB (Figure 10).

In Figure 19, we also compare our method with Draco [Goo17], a mesh compression library of industry standards, combined with mesh simplification [GH97]. To use mesh compression methods for implicit surfaces, one must discretize the surface first, which is vulnerable to discretization errors. We show that our method outperforms this strategy, especially on complex surfaces with intricate features. We note that while we cannot get a large performance improvement on simple input surfaces, we still have many benefits such as progressive refinement, scalar field compression, and many extensions (Section 5.2).

We note that our base mesh extraction does not guarantee the base mesh to have the same topology as the input surface. In fact, 3 out of 11 test cases in Figure 10 had a different genus than the input surface. However, even in such cases, our method had lower error with higher compression rate (Figure 20). This is because other methods also cannot escape from topological mismatches, as long as discretization is required.

We also qualitatively show that our method can extract a compact mesh with lower error (Figures 6, 4), preserves the spectral

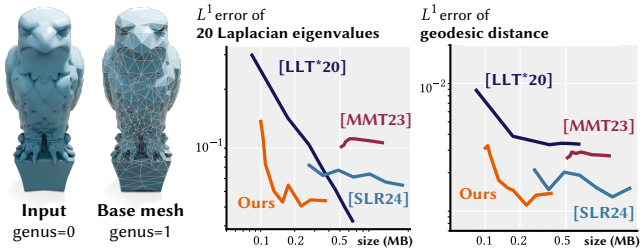


Figure 20: *Our method outperforms other methods, even when the base mesh does not match the topology of the input surface.*

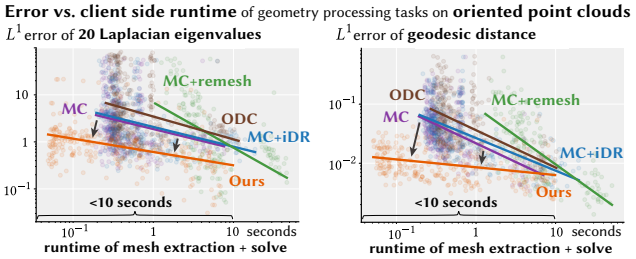


Figure 21: *Our method also shows the time- and error-reducing effect for oriented point clouds. Comparison methods were run on the fast generalized winding number [BDS*18].*

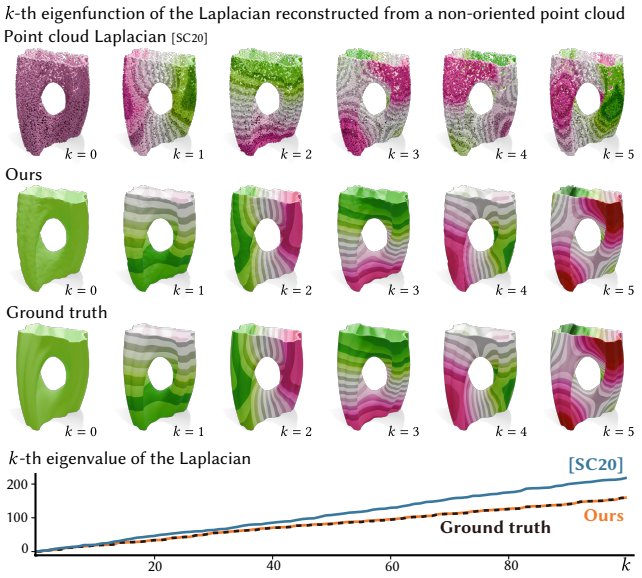


Figure 22: *On a thin shell, the spectral pattern of the point cloud Laplacian [SC20b] significantly differs from the ground truth. In contrast, our mesh successfully reconstructs its spectral pattern.*

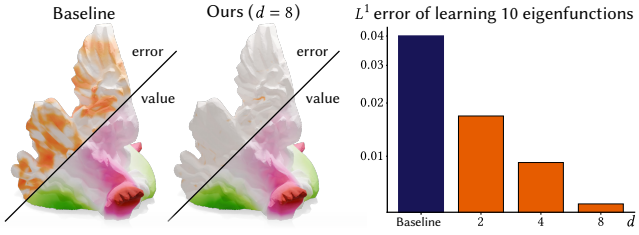


Figure 23: *Using our map and our intrinsic encoding, one may train a neural network that represents a scalar function on a surface, like the eigenfunctions of the Laplacian (left). By increasing the dimension of the feature vectors d, our method can achieve lower error without changing the number of layers/neurons of the MLP (right).*

pattern of the ground truth surface (Figures 5), and has a compact representation compared to mesh compression methods (Figure 9).

5.1.2. Other representations

In Figure 21, we show that our method extends well for oriented point clouds, showing the same runtime-reducing effect as in neural implicits. We also qualitatively show that our method can preserve the spectral pattern of the surface for point clouds (Figure 22).

5.1.3. Learning scalar fields

To validate the performance of our scalar field learning algorithm in Section 3.5, we learned the smallest 10 eigenfunctions of a shape to a single neural network by registering them to the latent code (Figure 23). We compared our method to a baseline MLP that uses the same parameters (2 hidden layers, 32 neurons, 8 positional encoding dimensions) but takes a 3D point on the fine surface as input and outputs a function value. By using our intrinsic encoding with $d = 8$, we achieved 3.6x lower error compared to the baseline, without increasing the number of neurons of the MLP.

5.2. Extensions and Applications

The accurate connectivity of our generated mesh can accommodate applications that require manifold and Delaunay meshes, such as vector field design [KCPS13] or stripe patterns on surfaces [KCPS15] (Figure 24). Having a good-quality mesh also contributes to interactively editing the design properties, like the heat source positions for heat simulation (Figure 1).

Our method can be used for realtime fracture simulation [SLMDS*23] (Figure 2). In their method, fracture mode decomposition takes minutes, which we want to avoid running on the client side. Using our intrinsic encoding, we train a binary classifier that tells whether a point $\mathbf{p} \in \Omega$ belongs to a certain piece of a certain fracture mode. Thanks to this compact classifier (68KB) and our efficient mesh extraction, the client side only required 0.6 seconds of precomputation, allowing us to run the fracture simulation in 7 ms.

Our method can handle cases where the shape of the base mesh and the final geometry are significantly different, like the torus and Bob (Figure 25). Given several pairs of anchor points to guide the training, our neural displacement field can successfully learn the mapping (see the supplemental for details). This can allow swapping the texture of the two surfaces simultaneously on the client side [SPK23], but without sending the full mesh.

Figure 24: *One may use our map for visualizing vector fields [KCPS13] or stripe patterns [KCPS15] on the client side.*

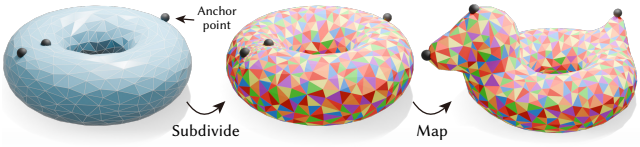


Figure 25: *Our method allows base meshes with different shapes but with the same topology.*

Finally, one may run different strategies to sample points on the mapped surface (Figure 26), which could be useful for visualization. As the determinant of our local Jacobian $\det \mathbf{J}_f$ is proportional to the distortion of our map, we achieve white noise sampling by uniformly generating samples on the base mesh and accepting samples at a rate proportional to it. Likewise, we achieve blue noise sampling by slightly modifying Poisson disk sampling [Bri07] to evaluate the radius of the annulus on the mapped points instead of on the base mesh.

6. Conclusion and Discussion

We propose a representation-agnostic method that learns a neural displacement field from a base mesh to the input surface. Our method addresses the requirements for streaming applications to be fast, compact, and accurate.

The cost of our fast and compact mesh extraction is its low visual quality. This is due to our design choice to prioritize speed and accuracy in intrinsic shape analysis, where Delaunay and non-manifold triangles matter more than their extrinsic appearance. We thus suppose our representation to be used as an *add-on* to existing representations that already excel in visual quality.

One remaining fundamental problem is how to visualize deformation or fracture on neural surfaces. To just visualize it, one may sample point clouds on the fine surface using our method (Figure 26), but this cannot leverage the existing neural rendering pipeline. One potential solution is to distort the rays accordingly to the deformation [SJNJ19, SGS*24], but it is unclear how to generalize them to various surface representations.

Another future direction is incorporating our method into an actual client-server setup. We anticipate that this would also require solving additional challenges (e.g., networking, streaming), and solving them would be a promising future direction.

We acknowledge that our method does not support some representations, such as unsigned distance functions sampled on a grid

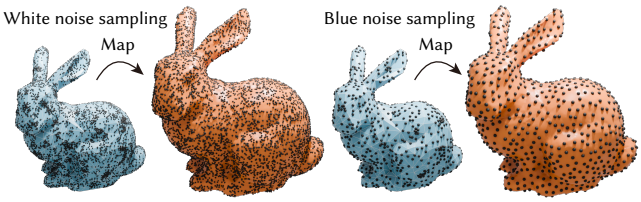


Figure 26: *Our map can facilitate different surface sampling strategies like white noise and blue noise sampling.*

(e.g., [CTFZ22]), where defining a projection operator is difficult due to gradient discontinuities.

Our method only supports mesh extraction for surface meshes, but many physics-based animation tasks work better with volumetric tetrahedral meshes. Thus, fast extraction of them using neural networks will be an exciting future direction.

Finally, we hope our method can empower the computer graphics community and beyond to run their interactive mesh processing applications on non-meshes.

Acknowledgements

We thank Jonathan Panuelos and Abhishek Madan for insightful discussions, and Thor Vestergaard Christiansen, Joonho Kim, and Victor Rong for proofreading. Our research is funded in part by NSERC Discovery (RGPIN–2022–04680), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program and gifts by Adobe Inc.

References

[AD01] ALLIEZ P., DESBRUN M.: Valence-driven connectivity encoding for 3d meshes. *Computer Graphics Forum 20*, 3 (2001), 480–489. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00541, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00541, doi:https://doi.org/10.1111/1467-8659.00541. 3

[AHY*19] ATZMON M., HAIM N., YARIV L., ISRAELOV O., MARON H., LIPMAN Y.: *Controlling neural level sets*. Curran Associates Inc., Red Hook, NY, USA, 2019. 8

[BDS*18] BARILL G., DICKSON N. G., SCHMIDT R., LEVIN D. I. W., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Trans. Graph. 37*, 4 (jul 2018). URL: https://doi.org/10.1145/3197517.3201337, doi:10.1145/3197517.3201337. 7, 9

[BGC*19] BEHRMANN J., GRATHWOHL W., CHEN R. T. Q., DUVENAUD D., JACOBSEN J.-H.: Invertible residual networks, 2019. URL: https://arxiv.org/abs/1811.00995, arXiv:1811.00995. 6

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (New York, NY, USA, 2004), SGP '04, Association for Computing Machinery, p. 185–192. URL: https://doi.org/10.1145/1057432.1057457, doi:10.1145/1057432.1057457. 3

[BPG*20] BEDNARIK J., PARASHAR S., GUNDOGDU E., SALZMANN

M., FUA P.: Shape reconstruction by learning differentiable surface representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 4716–4725. 4

[Bri07] BRIDSON R.: Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches* (New York, NY, USA, 2007), SIGGRAPH '07, Association for Computing Machinery, p. 22–es. URL: https://doi.org/10.1145/1278780.1278807, doi:10.1145/1278780.1278807. 10

[BSLF19] BEN-SHABAT Y., LINDENBAUM M., FISCHER A.: Nesti-net: Normal estimation for unstructured 3d point clouds using convolutional neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 10104–10112. doi:10.1109/CVPR.2019.01035. 4

[BTS*17] BERGER M., TAGLIASACCHI A., SEVERSKY L. M., ALLIEZ P., GUENNEBAUD G., LEVINE J. A., SHARF A., SILVA C. T.: A survey of surface reconstruction from point clouds. *Computer Graphics Forum 36*, 1 (2017), 301–329. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12802, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12802, doi:https://doi.org/10.1111/cgf.12802. 2

[CGA24] CGAL, Computational Geometry Algorithms Library, 2024. http://www.cgal.org. 4

[CKAJ23] CHEN Y.-C., KIM V., AIGERMAN N., JACOBSON A.: Neural progressive meshes. In *ACM SIGGRAPH 2023 Conference Proceedings* (New York, NY, USA, 2023), SIGGRAPH '23, Association for Computing Machinery. URL: https://doi.org/10.1145/3588432.3591531, doi:10.1145/3588432.3591531. 4

[CNGT14] CAO T.-T., NANJAPPA A., GAO M., TAN T.-S.: A gpu accelerated algorithm for 3d delaunay triangulation. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D '14, Association for Computing Machinery, p. 47–54. URL: https://doi.org/10.1145/2556700.2556710, doi:10.1145/2556700.2556710. 3

[CTFZ22] CHEN Z., TAGLIASACCHI A., FUNKHOUSER T., ZHANG H.: Neural dual contouring. *ACM Trans. Graph. 41*, 4 (July 2022). URL: https://doi.org/10.1145/3528223.3530108, doi:10.1145/3528223.3530108. 3, 10

[CX04] CHEN L., XU J.-C.: Optimal delaunay triangulations. *Journal of Computational Mathematics* (2004), 299–308. 6

[CYW*23] CHETAN A., YANG G., WANG Z., MARSCHNER S., HARIHARAN B.: Accurate differential operators for hybrid neural fields, 2023. URL: https://arxiv.org/abs/2312.05984, arXiv:2312.05984. 4

[CZ21] CHEN Z., ZHANG H.: Neural marching cubes. *ACM Trans. Graph. 40*, 6 (Dec. 2021). URL: https://doi.org/10.1145/3478513.3480518, doi:10.1145/3478513.3480518. 3

[dALJ*15] DE ARAÚJO B. R., LOPES D. S., JEPP P., JORGE J. A., WYVILL B.: A survey on implicit surface polygonization. *ACM Comput. Surv. 47*, 4 (May 2015). URL: https://doi.org/10.1145/2732197, doi:10.1145/2732197. 2

[Dee95] DEERING M.: Geometry compression. 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. 13–20. URL: https://doi.org/10.1145/218380.218391, doi:10.1145/218380.218391. 3

[DK91] DOI A., KOIDE A.: An efficient method of triangulating equivalued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information E74-D*, 1 (January 1991), 214–224. 3

[DZJ*24] DOU Y., ZHENG Z., JIN Q., SHI R., LI Y., NI B.: Differentiable micro-mesh construction. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2024), pp. 4294–4303. doi:10.1109/CVPR52733.2024.00411. 4

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. *ACM Trans. Graph. 21*, 3 (July 2002), 355–361. URL: https://doi.org/10.1145/566654.566589, doi:10.1145/566654.566589. 2, 4

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. URL: https://doi.org/10.1145/258734.258849, doi:10.1145/258734.258849. 3, 5, 7, 8

[GKOM18] GUERRERO P., KLEIMAN Y., OVSJANIKOV M., MITRA N. J.: Pcpn<scp>et</scp> learning local shape properties from raw point clouds. *Computer Graphics Forum 37*, 2 (May 2018), 75–85. URL: http://dx.doi.org/10.1111/cgf.13343, doi:10.1111/cgf.13343. 4

[Goo17] GOOGLE: Draco: 3d data compression, 2017. URL: https://github.com/google/draco. 3, 8

[GYBC24] GILLESPIE M., YANG D., BOTSCH M., CRANE K.: Ray tracing harmonic functions. *ACM Trans. Graph. 43*, 4 (July 2024). URL: https://doi.org/10.1145/3658201, doi:10.1145/3658201. 7, 8

[HS24] HWANG J., SUNG M.: Occupancy-based dual contouring. In *SIGGRAPH Asia 2024 Conference Papers* (New York, NY, USA, 2024), SA '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3680528.3687581, doi:10.1145/3680528.3687581. 2, 3

[HWW*24] HUANG Z., WEN Y., WANG Z., REN J., JIA K.: Surface reconstruction from point clouds: A survey and a benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024). 2

[JKSH13] JACOBSON A., KAVAN L., SORKINE-HORNUNG O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4 (July 2013). URL: https://doi.org/10.1145/2461912.2461916, doi:10.1145/2461912.2461916. 7, 8

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Trans. Graph. 21*, 3 (July 2002), 339–346. URL: https://doi.org/10.1145/566654.566586, doi:10.1145/566654.566586. 3

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 4

[JSZP20] JIANG Z., SCHNEIDER T., ZORIN D., PANOZZO D.: Bijective projection in a shell. *ACM Trans. Graph. 39*, 6 (Nov. 2020). URL: https://doi.org/10.1145/3414685.3417769, doi:10.1145/3414685.3417769. 2, 4

[JZH*21] JIANG Z., ZHANG Z., HU Y., SCHNEIDER T., ZORIN D., PANOZZO D.: Bijective and coarse high-order tetrahedral meshes. *ACM Trans. Graph. 40*, 4 (July 2021). URL: https://doi.org/10.1145/3450626.3459840, doi:10.1145/3450626.3459840. 2, 4

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (2006), vol. 7. 8

[KCPS13] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Globally optimal direction fields. *ACM Trans. Graph. 32*, 4 (July 2013). URL: https://doi.org/10.1145/2461912.2462005, doi:10.1145/2461912.2462005. 9, 10

[KCPS15] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Stripe patterns on surfaces. *ACM Trans. Graph. 34*, 4 (July 2015). URL: https://doi.org/10.1145/2767000, doi:10.1145/2767000. 9, 10

[KKLD23] KERBL B., KOPANAS G., LEIMKUEHLER T., DRETTAKIS G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph. 42*, 4 (July 2023). URL: https://doi.org/10.1145/3592433, doi:10.1145/3592433. 7

[KSA*24] KING N., SU H., AANJANEYA M., RUUTH S., BATTY C.: A closest point method for pdes on manifolds with interior boundary conditions for geometry processing. *ACM Trans. Graph. 43*, 5 (Aug. 2024). URL: https://doi.org/10.1145/3673652, doi:10.1145/3673652. 4, 8

[LB03] LOPES A., BRODLIE K.: Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (Jan. 2003), 16–29. URL: https://doi.org/10.1109/TVCG.2003.1175094, doi:10.1109/TVCG.2003.1175094. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. URL: https://doi.org/10.1145/37402.37422, doi:10.1145/37402.37422. 2, 3, 7

[LJG*24] LIU S., JI Y., GUO J.-P., LIU L., FU X.-M.: Smooth bijective projection in a high-order shell. *ACM Trans. Graph. 43*, 4 (July 2024). URL: https://doi.org/10.1145/3658207, doi:10.1145/3658207. 2, 4

[LKC*20] LIU H.-T. D., KIM V. G., CHAUDHURI S., AIGERMAN N., JACOBSON A.: Neural subdivision. *ACM Trans. Graph. 39*, 4 (Aug. 2020). URL: https://doi.org/10.1145/3386569.3392418, doi:10.1145/3386569.3392418. 4

[LLT*20] LESCOAT T., LIU H.-T. D., THIERY J.-M., JACOBSON A., BOUBEKEUR T., OVSJANIKOV M.: Spectral mesh simplification. *Computer Graphics Forum 39*, 2 (2020), 315–324. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13932, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13932, doi:https://doi.org/10.1111/cgf.13932. 4

[LOW*23] LI M., OWENS M., WU J., YANG G., CHERN A.: Closest point exterior calculus. In *SIGGRAPH Asia 2023 Posters*. 2023, pp. 1–2. 4

[LSS*98] LEE A. W. F., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: Maps: multiresolution adaptive parameterization of surfaces. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, Association for Computing Machinery, p. 95–104. URL: https://doi.org/10.1145/280814.280828, doi:10.1145/280814.280828. 2, 4

[LSW*22] LI L., SHEN Z., WANG Z., SHEN L., BO L.: Compressing volumetric radiance fields to 1 mb, 2022. URL: https://arxiv.org/abs/2211.16386, arXiv:2211.16386. 1

[LZBCJ21] LIU H.-T. D., ZHANG J. E., BEN-CHEN M., JACOBSON A.: Surface multigrid via intrinsic prolongation. *ACM Trans. Graph. 40*, 4 (July 2021). URL: https://doi.org/10.1145/3450626.3459768, doi:10.1145/3450626.3459768. 2, 4

[MAKM21] MORREALE L., AIGERMAN N., KIM V., MITRA N. J.: Neural surface maps. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 4637–4646. doi:10.1109/CVPR46437.2021.00461. 4

[MESK22] MÜLLER T., EVANS A., SCHIED C., KELLER A.: Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics 41*, 4 (July 2022), 1–15. URL: http://dx.doi.org/10.1145/3528223.3530127, doi:10.1145/3528223.3530127. 1

[MLDH15] MAGLO A., LAVOUÉ G., DUPONT F., HUDELOT C.: 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv. 47*, 3 (Feb. 2015). URL: https://doi.org/10.1145/2693443, doi:10.1145/2693443. 3

[MMT23] MAGGIORDOMO A., MORETON H., TARINI M.: Micro-mesh construction. *ACM Trans. Graph. 42*, 4 (July 2023). URL: https://doi.org/10.1145/3592440, doi:10.1145/3592440. 4

[MSLJ23] MARSCHNER Z., SELLÁN S., LIU H.-T. D., JACOBSON A.: Constructive solid geometry on neural signed distance fields. In *SIGGRAPH Asia 2023 Conference Papers* (New York, NY, USA, 2023), SA '23, Association for Computing Machinery. URL: https://doi.org/10.1145/3610548.3618170, doi:10.1145/3610548.3618170. 8

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV* (2020). 6, 8

[NSS*22] NOVELLO T., SCHARDONG G., SCHIRMER L., DA SILVA V., LOPES H., VELHO L.: Exploring differential geometry in neural implicits. *Comput. Graph. 108*, C (Nov. 2022), 49–60. URL: https://doi.org/10.1016/j.cag.2022.09.003, doi:10.1016/j.cag.2022.09.003. 4

[OL03] OTADUY M. A., LIN M. C.: Sensation preserving simplification for haptic rendering. *ACM Trans. Graph. 22*, 3 (July 2003), 543–553. URL: https://doi.org/10.1145/882262.882305, doi:10.1145/882262.882305. 4

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation, 2019. URL: https://arxiv.org/abs/1901.05103, arXiv:1901.05103. 7

[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), pp. 1–6. doi:10.1109/ICMEW46912.2020.9105972. 4

[PPB25] PENTAPATI S. K., PHILLIPS G., BOVIK A. C.: Mesh compression with quantized neural displacement fields, 2025. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.70074, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.70074, doi:https://doi.org/10.1111/cgf.70074. 4

[PZL*24] PANG B., ZHENG Z., LI Y., WANG G., WANG P.-S.: Neural laplacian operator for 3d point clouds. *ACM Trans. Graph. 43*, 6 (Nov. 2024). URL: https://doi.org/10.1145/3687901, doi:10.1145/3687901. 4

[Ros99] ROSSIGNAC J.: Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61. doi:10.1109/2945.764870. 8

[Ros01] ROSSIGNAC J.: 3d compression made simple: Edgebreaker with zip&wrap on a corner-table. In *Proceedings of the International Conference on Shape Modeling & Applications* (USA, 2001), SMI '01, IEEE Computer Society, p. 278. 3

[SC20a] 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://doi.org/10.1145/3386569.3392374, doi:10.1145/3386569.3392374. 4, 8

[SC20b] SHARP N., CRANE K.: A laplacian for nonmanifold triangle meshes. *Computer Graphics Forum 39*, 5 (2020), 69–80. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14069, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14069, doi:https://doi.org/10.1111/cgf.14069. 3, 9

[SG03] SURAZHSKY V., GOTSMAN C.: Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Goslar, DEU, 2003), SGP '03, Eurographics Association, p. 20–30. 3

[SGS*24] SUN B., GROUEIX T., SONG C., HUANG Q., AIGERMAN N.: Tuttenet: Injective 3d deformations by composition of 2d mesh deformations. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition 2024* (2024). 6, 10

[SJNJ19] SEYB D., JACOBSON A., NOWROUZEZAHRAI D., JAROSZ W.: Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Trans. Graph. 38*, 6 (Nov. 2019). URL: https://doi.org/10.1145/3355089.3356502, doi:10.1145/3355089.3356502. 10

[SJW07] SCHAEFER S., JU T., WARREN J.: Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (May 2007), 610–619. URL: https://doi.org/10.1109/TVCG.2007.1012, doi:10.1109/TVCG.2007.1012. 3

[SKHB24] SUGIMOTO R., KING N., HACHISUKA T., BATTY C.: Projected walk on spheres: A monte carlo closest point method for surface pdes. In *SIGGRAPH Asia 2024 Conference Papers* (New York, NY, USA, 2024), SA '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3680528.3687599, doi:10.1145/3680528.3687599. 4

[SKO24] SUNDARARAMAN R., KLOKOV R., OVSJANIKOV M.: Self-supervised dual contouring, 2024. URL: https://arxiv.org/abs/2405.18131, arXiv:2405.18131. 3

[SKR01] SZYMCZAK A., KING D., ROSSIGNAC J.: An edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry 20*, 1 (2001), 53–68. Selected papers from the 12th Annual Canadian Conference on. URL: https://www.sciencedirect.com/science/article/pii/S0925772101000359, doi:https://doi.org/10.1016/S0925-7721(01)00035-9. 3

[SLMDS*23] SELLÁN S., LUONG J., MATTOS DA SILVA L., RAMAKRISHNAN A., YANG Y., JACOBSON A.: Breaking good: Fracture modes for realtime destruction. *ACM Trans. Graph. 42*, 1 (Mar. 2023). URL: https://doi.org/10.1145/3549540, doi:10.1145/3549540. 2, 9

[SLR24] SIVARAM V. E., LI T.-M., RAMAMOORTHI R.: Neural geometry fields for meshes. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, 2024), SIGGRAPH '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3641519.3657399, doi:10.1145/3641519.3657399. 4, 6

[SPK23] SCHMIDT P., PIEPER D., KOBBELT L.: Surface maps via adaptive triangulations. *Computer Graphics Forum 42*, 2 (2023), 103–117. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14747, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14747, doi:https://doi.org/10.1111/cgf.14747. 9

[SRK02] SZYMCZAK A., ROSSIGNAC J., KING D.: Piecewise regular meshes: Construction and compression. *Graphical Models 64*, 3 (2002), 183–198. URL: https://www.sciencedirect.com/science/article/pii/S1524070302905771, doi:https://doi.org/10.1006/gmod.2002.0577. 3

[SSC19] SHARP N., SOLIMAN Y., CRANE K.: Navigating intrinsic triangulations. *ACM Trans. Graph. 38*, 4 (July 2019). URL: https://doi.org/10.1145/3306346.3322979, doi:10.1145/3306346.3322979. 3, 6

[SSJC22] SAWHNEY R., SEYB D., JAROSZ W., CRANE K.: Grid-free monte carlo for pdes with spatially varying coefficients. *ACM Trans. Graph. 41*, 4 (July 2022). URL: https://doi.org/10.1145/3528223.3530134, doi:10.1145/3528223.3530134. 4

[SW04] SCHAEFER S., WARREN J.: Dual marching cubes: Primal contouring of dual grids. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (USA, 2004), PG '04, IEEE Computer Society, p. 70–76. 3

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada* (June 1998), pp. 26–34. URL: http://graphicsinterface.org/wp-content/uploads/gi1998-4.pdf. 3

[TMND*23] TAKIKAWA T., MÜLLER T., NIMIER-DAVID M., EVANS A., FIDLER S., JACOBSON A., KELLER A.: Compact neural graphics primitives with learned hash probing. In *SIGGRAPH Asia 2023 Conference Papers* (New York, NY, USA, 2023), SA '23, Association for Computing Machinery. URL: https://doi.org/10.1145/3610548.3618167, doi:10.1145/3610548.3618167. 1

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *ACM Trans. Graph. 17*, 2 (Apr. 1998), 84–115. URL: https://doi.org/10.1145/274363.274365, doi:10.1145/274363.274365. 3

[WLL*21] WANG P., LIU L., LIU Y., THEOBALT C., KOMURA T., WANG W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689* (2021). 7

[WM25] WILLIAMSON R., MITRA N. J.: Neural geometry processing via spherical neural surfaces, 2025. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.70021, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.70021, doi:https://doi.org/10.1111/cgf.70021. 4

[YBHK21] YANG G., BELONGIE S., HARIHARAN B., KOLTUN V.: Geometry processing with neural fields. In *Advances in Neural Information Processing Systems* (2021), Ranzato M., Beygelzimer A., Dauphin Y., Liang P., Vaughan J. W., (Eds.), vol. 34, Curran Associates, Inc., pp. 22483–22497. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/bd686fd640be98efaae0091fa301e613-Paper.pdf. 4

[ZBS*22] ZHANG X., BI S., SUNKAVALLI K., SU H., XU Z.: Nerfusion: Fusing radiance fields for large-scale scene reconstruction, 2022. URL: https://arxiv.org/abs/2203.11283, arXiv:2203.11283. 7

[ZJK24] ZHANG J. E., JAMES D., KAUFMAN D. M.: Progressive dynamics for cloth and shell animation. *ACM Trans. Graph. 43*, 4 (July 2024). URL: https://doi.org/10.1145/3658214, doi:10.1145/3658214. 7