Discontinuity-Aware 2D Neural Fields

YASH BELHE, University of California San Diego, USA
MICHAËL GHARBI, Adobe Research, USA
MATTHEW FISHER, Adobe Research, USA
ILIYAN GEORGIEV, Adobe Research, UK
RAVI RAMAMOORTHI, University of California San Diego, USA
TZU-MAO LI, University of California San Diego, USA

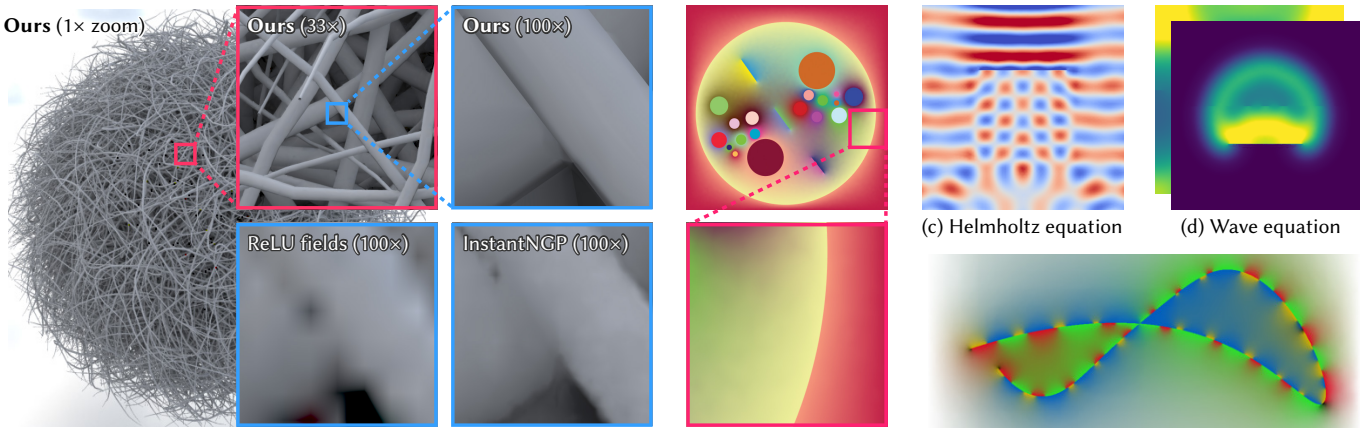


Fig. 1. We introduce a neural-mesh-based representation that can preserve sharp discontinuities (linear and curved) provided as input, at arbitrary resolution for a variety of two-dimensional signals. (a) Rendering comparison to InstantNGP and ReLU fields at 100× zoom, showing how we maintain sharp structures while previous methods blur the image. (b-e) Example uses of our method for fitting solutions to walk-on-spheres-based diffusion curves, partial differential equations, and physics-informed neural networks. In all cases, discontinuities are specified by curves. See Section 6 for more comparisons to prior works.

Neural image representations offer the possibility of high fidelity, compact storage, and resolution-independent accuracy, providing an attractive alternative to traditional pixel- and grid-based representations. However, coordinate neural networks fail to capture discontinuities present in the image and tend to blur across them; we aim to address this challenge. In many cases, such as rendered images, vector graphics, diffusion curves, or solutions to partial differential equations, the locations of the discontinuities are known. We take those locations as input, represented as linear, quadratic, or cubic Bézier curves, and construct a feature field that is discontinuous across these locations and smooth everywhere else. Finally, we use a shallow multi-layer perceptron to decode the features into the signal value. To construct the feature field, we develop a new data structure based on a curved triangular mesh, with features stored on the vertices and on a subset of the edges that are marked as discontinuous. We show that our method can be used to compress a $100,000^2$-pixel rendered image into a 25MB file; can be used as a new diffusion-curve solver by combining with Monte-Carlo-based methods or directly supervised by the diffusion-curve energy; or can be used for compressing 2D physics simulation data.

CCS Concepts: • **Computing methodologies** → **Image compression**; **Image representations**; **Reconstruction**; **Neural networks**; **Computer graphics**.

Additional Key Words and Phrases: discontinuous, neural fields, infinite resolution, walk on spheres, physics informed neural networks.

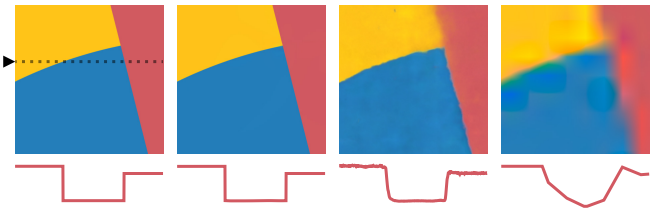
ACM Reference Format:
Yash Belhe, Michaël Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Discontinuity-Aware 2D Neural Fields. *ACM Trans. Graph.* 41, 3, Article 1 (November 2023), 11 pages. https://doi.org/10.1145/3504002

Authors' addresses: Yash Belhe, University of California San Diego, USA, ybelhe@ucsd.edu; Michaël Gharbi, Adobe Research, USA, mgharbi@adobe.com; Matthew Fisher, Adobe Research, USA, matfishe@adobe.com; Iliyan Georgiev, Adobe Research, UK, igeorgiev@adobe.com; Ravi Ramamoorthi, University of California San Diego, USA, ravir@ucsd.edu; Tzu-Mao Li, University of California San Diego, USA, tzli@ucsd.edu.

0730-0301/2023/11-ART1 $15.00
https://doi.org/10.1145/3504002

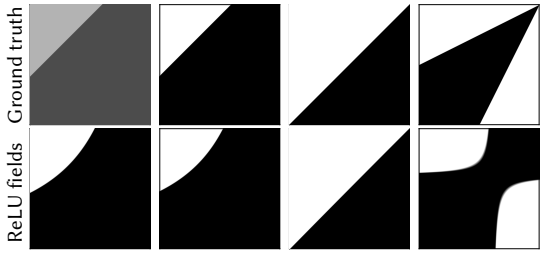
1 INTRODUCTION

Images are traditionally represented either as pixels (raster images) or simple geometric curves (vector graphics). Recent advances in neural fields [Xie et al. 2022] hint at the possibility of representing images using coordinate neural networks [Song et al. 2016; Chen et al. 2021; Müller et al. 2022] to preserve rich details in a resolution-independent fashion. Unfortunately, directly representing an image



(a) Ground truth (b) **Ours** (c) InstantNGP (d) ReLU fields

Fig. 2. Existing grid and neural architectures (c, d) continuously map input features to output colors. They cannot represent discontinuities (a) and blur across them (top row); our feature field preserves sharp boundaries (b) by changing discontinuously across them. Bottom row: 1D cross-section of the red channel at the marked horizontal scanline shows continuous color variation in (c,d) across edges and discontinuous variation for ours (b).



(a) Non-binary (b) Binary (c) Binary (d) Two edges

Fig. 3. (a) ReLU fields cannot represent jump discontinuities if one side is not zero (since it relies on the ReLU clamping). (b, c) it can represent a single binary edge in the cell reasonably well. (d) but the approximation quality deteriorates when there are more edges in the cell.

using a coordinate neural network fails to model discontinuities, because these representations are continuous by construction, which leads to blurring over edges (see Figs. 1 and 2).

In many situations, such as path-traced renders, vector graphics, diffusion curves [Orzan et al. 2008], or partial differential equations, the 2D locations of discontinuities are known *a priori*. We take these discontinuity locations, represented as lines or quadratic/cubic Bézier curves as common in vector graphics, and produce a hybrid neural-mesh-based data structure that lets us model 2D signals containing both discontinuities and rich, smooth color variations.

To design a resolution-independent representation that is only discontinuous at the curve locations, and otherwise smooth almost everywhere, we extend the classical feature-based textures [Ramanarayanan et al. 2004; Sen 2004; Tumblin and Choudhury 2004; Tarini and Cignoni 2005; Parilov and Zorin 2008; Pavić and Kobbelt 2010]. Instead of using a regular grid to store and interpolate features, we propose a new data structure based on a curved triangular mesh [Hu et al. 2019]. We store neural feature vectors at the mesh vertices, and mark a subset of the mesh edges as discontinuous. We devise an interpolation scheme to reconstruct a feature field that is discontinuous *only* on the marked edges (Fig. 4 and Section 4). The interpolated features are fed to a multi-layer perceptron to obtain the desired output signal. Our core contribution is making the neural features discontinuous across input discontinuity curves, and as such, our method can be paired with different neural-network architectures and loss functions.

We demonstrate our approach on many tasks and compare it to existing representations (Fig. 1, Section 6). We show that with the same size as existing neural representations [Müller et al. 2022; Karnewar et al. 2022], our method produces significantly more accurate results (sometimes > 10dB) when zooming in the signal. We apply our method for: (1) compressing a $100,000^2$-pixel Monte Carlo path-traced raster image from 33GB to 25MB; (2) rendering diffusion curves using a novel combination of neural representation and Monte Carlo walk on spheres [Muller 1956; Sawhney and Crane 2020] that reduces the sampling variance by an order of magnitude; (3) training physics-informed neural networks [Raissi et al. 2019] with results 9dB more accurate than alternatives; and (4) compressing finite-element/difference solutions for Helmholtz and wave equations by a factor of 20–50×.

2 RELATED WORK

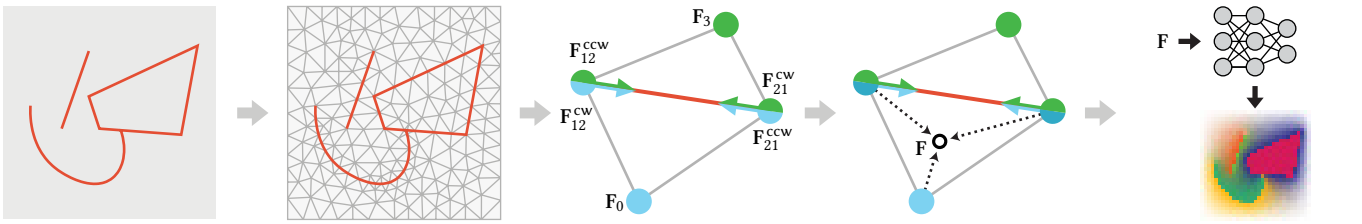
Mesh/curves representation for images. Early rendering work represents global illumination as a triangle mesh in image space with each edge marking the discontinuity locations [Pighin et al. 1997]. To add detail to the interior of the vector graphics, a variety of hybrid data structures with both pixels and edges have been proposed [Salisbury et al. 1996; Sen et al. 2003; Sen 2004; Ramanarayanan et al. 2004; Tumblin and Choudhury 2004; Tarini and Cignoni 2005; Parilov and Zorin 2008; Pavić and Kobbelt 2010; Reshetov and Luebke 2016], with work extending similar ideas to volume rendering [Agus et al. 2010]. All these works assume that the "smooth" part of the signal is represented by a grid, and the discontinuities are represented by lines or curves, and apply an edge-preserving reconstruction to interpolate. Assumptions have to be made about the topology of the curve network for the edge-preserving reconstruction to work. We revisit this class of techniques with neural representations, leading to significantly different design decisions. Crucially, apart from the use of neural networks, we store the feature vectors on the vertices of a curved triangular mesh, instead of a grid.

Diffusion curves [Orzan et al. 2008] represent images using curves and solve a Laplace equation using Dirichlet boundary conditions defined on the curves. We show that our representation can be used for both solving the Laplace equation or fitting a walk-on-spheres solution [Muller 1956; Sawhney and Crane 2020], and achieves significantly better results compared to existing neural representations.

Edge-based and mesh-based representations of images have been used for smoothly reconstructing rendered images [Bala et al. 2003; Velazquez-Armendariz et al. 2006], representing videos [Chen et al. 2011] and light-fields [Broxton et al. 2020].

Some image-vectorization works [Selinger 2003; Xie et al. 2014], as well as early work on discontinuity meshing in radiosity [Lischinski et al. 1992; Heckbert 1992] extract discontinuities from signals. Our work currently assumes the discontinuity locations are known a priori, and they can be obtained through these methods.

Neural fields. There is a surge of interest using neural networks to augment representations in graphics [Xie et al. 2022]. Song et al. [2016], to our knowledge, pioneered the idea of representing images using coordinate neural networks. Encoding *giga images* [Kopf et al.



(a) Discontinuity curves (b) Curved triangulation (c) Feature assignment (d) Discontinuity-aware feature interpolation (e) MLP feature decoding

Fig. 4. Overview. (a) Our method takes in a set of discontinuity curves, and (b) applies existing robust curved triangulation [Hu et al. 2019] to them. (c) We then assign one or more features to every vertex. For vertices next to discontinuity curves (red lines), we assign a feature for each side of every discontinuity. (d) Inference at a query point begins by interpolating the vertex features, respecting the discontinuities, to obtain a feature vector F. (e) The vector F is then passed to a shallow multi-layer perceptron to decode the signal value at the query point. We train the vertex features and the multi-layer perceptron together.

2007] is a popular application for recent neural fields [Chen et al. 2021; Mehta et al. 2021; Müller et al. 2022; Martel et al. 2021].

We build our work on the recent *feature fields* neural representations [Chen et al. 2021; Müller et al. 2022; Martel et al. 2021]. They store feature vectors on grid (or tetrahedron [Shen et al. 2021]) vertices, and pass the smoothly interpolated features to a shallow neural network to produce the reconstructed signal. We store the features on a curved triangular mesh, and interpolate them in a discontinuity-aware manner.

Adaptive grids are often used for neural representation [Ren et al. 2013; Yu et al. 2021; Martel et al. 2021]. However, since grids are not aligned with discontinuities, their precision will always be limited.

Our goal is highly related to that of ReLU fields [Karnewar et al. 2022]. ReLU fields directly store the smooth signals on a grid, and use non-linear clamping on top of bilinear interpolation to reconstruct the signal. The clamping itself does *not* introduce discontinuities, but it allows reconstruction of high-frequency signals. ReLU fields are extremely efficient and expressive. However, the clamping to zero and one means that they cannot represent jump discontinuities where the values on the two sides are not zero and one. Empirically, we also find that gradient-based optimization does not allow the network to precisely locate the discontinuities (see Fig. 3).

Physics-informed neural networks. An emerging approach to solve partial differential equations is to fit a neural network using a loss directly coming from the differential equation [Raissi et al. 2019; Zehnder et al. 2021; Sitzmann et al. 2020]. We found that when there are jump discontinuities not at the domain boundaries, naïve physics-informed neural networks cannot preserve the sharp transitions. Our representation can be used as a basis for new physics-informed neural networks for solving low-dimensional partial differential equations.

Discontinuities in FEM. Our work is related to extended finite-element (XFEM) methods [Moës et al. 1999] which extend the approximation space of finite-element methods from continuous basis functions to discontinuous ones, significantly improving convergence rates near discontinuities in the simulation domain. Similarly, by making feature fields discontinuous, we are able to represent details that continuous feature fields cannot capture.

3 OVERVIEW

Given a target continuous-scale signal and a set of curves that denote its discontinuity locations as input (see Fig. 4a), our goal is to construct a compact resolution-independent approximation to the target, such that it can represent sharp changes across the discontinuities while maintaining continuity everywhere else.

We want to adopt a modern neural-feature-field appoach. This means that we need to sample points in image space, store some feature vectors on them, and interpolate the features to feed to a neural network. It might be tempting to sample on a regular/adaptive grid like existing work, but this would then require us to interpolate the grid values without smoothing over the discontinuities. We show in supplementary document that no matter how finely one subdivides the grids, an edge-aware interpolation on the grid can never represent (curved) discontinuities at points with large valence.

Our key idea is to make the feature-field points aligned with the discontinuity curves. Our representation is a hybrid neural-mesh-based data structure in which the mesh vertices store features and the neural network decodes the interpolated features. We first perform a curved triangulation of the domain such that all input discontinuity curves form a subset of the edges of our mesh (Fig. 4b); we mark these (curved) edges as discontinuous. Next, we place features on the triangle vertices and the marked edges (Fig. 4c). We then perform a discontinuity-aware interpolation that ensures a sharp transition in features across discontinuities and smooth transition elsewhere (Fig. 4d). Finally, the interpolated features are fed to a shallow multi-layer perceptron (MLP) to produce the signal value (Fig. 4e).

4 METHOD

Below we present a mathematical definition of our desired continuity criteria (Section 4.1). Next, we describe the curved triangulation (Section 4.2) and discontinuity-aware interpolation (Section 4.3) to set up our features. Finally we show how our method satisfies the continuity criteria (Section 4.4).

4.1 Continuity criteria

We illustrate our continuity criteria in Fig. 5 on a signal computed using diffusion curves [Orzan et al. 2008] which solves a Laplace equation with boundary conditions defined on a Bézier curve. We

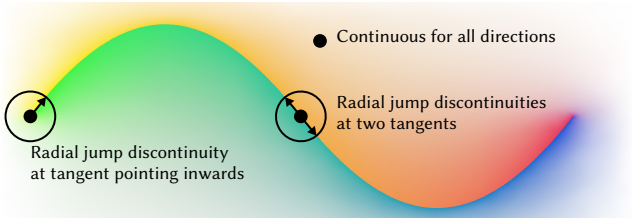
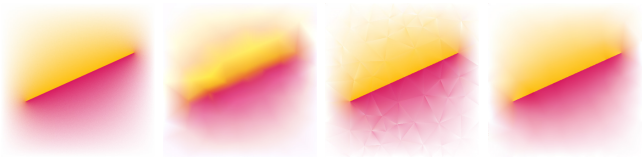


Fig. 5. Continuity criteria. A 2D signal with discontinuities for motivating our approach. Points in the domain can be categorized into three classes: (1) At points outside the curve the signal is continuous; (2) For a point on the curve interior, walking on a small surrounding circle will see two jump discontinuities at both tangent directions; (3) For a curve endpoint, walking on a small circle will see one jump discontinuity. Our goal is to have a data structure, such that when we query with continuous 2D coordinates, the returned values would preserve these criteria (Section 4.1).



(a) Reference (b) All continuous (c) All discontinuous (d) Discontinuous across edge (**ours**)

Fig. 6. Naïve interpolation introduces artifacts. We fit a diffusion-curve image (a) to three different triangle-mesh-based representations using walk-on-spheres Monte Carlo samples (see Section 6.2). (b) Storing a single feature per vertex results in a fully continuous representation which blurs the edge. (c) Storing vertex features independently for all triangles introduces spurious artifacts at edges between triangles (zoom in). (d) Our method places extra features on vertices that lie on the discontinuity and smoothly interpolates them except across discontinuities, producing the desired result.

are given a two-dimensional domain Ω and a set of one-dimensional discontinuity curves $\Gamma = \{\gamma_0, ...\gamma_{n-1}\}$, $\gamma_i \subset \Omega \quad \forall i$. We denote the set of endpoints of the curves as $\partial\Gamma$. We assume that the curves do not intersect with each other, except at their endpoints. In practice, we rely on the curved triangulation process (Fig. 4b) to resolve intersections and split the curves. We want to construct a function $f : \Omega \to \mathbb{R}^K$ such that

(1) if $\mathbf{x} \in \Omega \setminus \Gamma$, f is continuous at $\mathbf{x}$;
(2) if $\mathbf{x} \in \Gamma \setminus \partial\Gamma$, $\mathbf{x}$ is *directionally discontinuous* for the two tangent directions and continuous in all other directions;
(3) if $\mathbf{x} \in \partial\Gamma$, $\mathbf{x}$ is directionally discontinuous at the tangent direction pointing inwards to the curve(s) that end(s) at $\mathbf{x}$ and continuous in all other directions.

By *directionally discontinuous* over a point $\mathbf{x}$ and direction $\mathbf{d}$, we mean that if we walk on a circle with a small radius around $\mathbf{x}$, the function would have a jump discontinuity at direction $\mathbf{d}$. Below we formally define the meaning of directional continuity.

Definition 4.1. **Directional limit.** The directional limit of a function f at a point $\mathbf{x} \in \mathbb{R}^2$ along the direction $\mathbf{d} \in \mathbb{S}^1$ is given by $h(\mathbf{x}, \mathbf{d}) = \lim_{t \to 0^+} f(\mathbf{x} + t\mathbf{d})$.

Definition 4.2. **Directional continuity.** A function f is defined to be directionally continuous at point $\mathbf{x}$ and the direction $\mathbf{d}$ if $\lim_{\mathbf{d}' \to \mathbf{d}} h(\mathbf{x}, \mathbf{d}') = h(\mathbf{x}, \mathbf{d})$. If the limit does not exist for some pair $(\mathbf{x}, \mathbf{d})$, we call f *directionally discontinuous* at $(\mathbf{x}, \mathbf{d})$.

4.2 Curved triangulation

Given a set of discontinuity lines or Bézier curves (Fig. 4a), we first perform a curved triangulation using the input curves as constraints to obtain a triangular mesh (Fig. 4b). We build on the recent advance in robust curved triangulation and apply TriWild [Hu et al. 2019]. The triangulation process resamples[1] the curves into a set $\Gamma = \{\gamma_0, ...\gamma_{n-1}\} \subset \Omega$ to ensure that the resulting mesh satisfies:

(1) If γ_i is a line, then it is an edge of a triangle;
(2) If γ_i is a Bézier curve, then it is fully contained inside a triangle with two end points being two of the triangle vertices.

[1]During the resampling, TriWild can slightly modify the input curve depending on the tolerance parameters. In practice we have found the difference to be imperceptible and assume that TriWild's resampling perserves the discontinuity curves.

After the triangulation process, we mark an edge to be *discontinuous* if it was part of the input discontinuity curves. The curves are provided in the problem definition for applications such as boundary conditions of partial differential equations (Sections 6.2 to 6.4), and must be extracted for applications like rendering (Section 6.1).

4.3 Discontinuity-aware feature interpolation

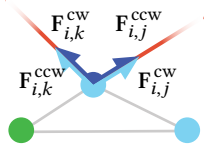
Given the triangle mesh, we want to interpolate features between vertices while respecting discontinuities (Section 4.1). To achieve this, we first assign to each vertex a set of features (Fig. 4c), then interpolate them without smoothing across the discontinuity curves (Fig. 4d). For simplicity, we fist focus on the case of linear edges. Even in this simple case, naïve methods would not work: storing a single feature per vertex and interpolating using barycentric coordinates would smooth across the discontinuities (Fig. 6b), while storing three features per triangle and interpolating would create spurious discontinuities at edges not marked as discontinuous (Fig. 6c).

4.3.1 Dual features at discontinuous vertices. Our idea for the feature assignment is to store two sets of features on each pair of a vertex and a discontinuous edge. At a vertex i, we store two features on each side of every discontinuous edge (i, j) – a *clockwise* feature $\mathbf{F}_{i,j}^{\text{cw}}$ and a *counterclockwise* feature $\mathbf{F}_{i,j}^{\text{ccw}}$. If there are no discontinuous edges associated with the vertex, then we store a single vertex feature $\mathbf{F}_i$. Fig. 4c illustrates the feature assignment. All features are multi-dimensional vectors; we use 5D vectors in all our examples.

Given the features and a query point inside a triangle with barycentric coordinates b_1, b_2, we want to barycentrically interpolate:

$$\mathbf{F} = (1 - b_1 - b_2)\hat{\mathbf{F}}_0 + b_1\hat{\mathbf{F}}_1 + b_2\hat{\mathbf{F}}_2. \tag{1}$$

However, how do we get the features $\hat{\mathbf{F}}_i$ to interpolate? For example, for the gray-edged triangle on the right, vertex i at the top has four possible values: how do we pick between the four?



4.3.2 Radial interpolation of discontinuous features. Inspired by the directional continuity definition (Section 4.1), our idea is to *radially interpolate* features. Given a query point in a triangle, we want to determine the feature $\hat{\mathbf{F}}_i$ at each triangle vertex i. If vertex i is

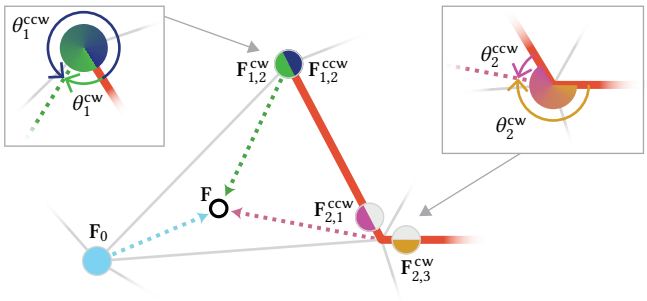


Fig. 7. Discontinuity-aware feature interpolation. For a query point inside a triangle, we barycentrically interpolate between the features at vertices 0 (left), 1 (top), and 2 (right) to obtain a feature vector $\mathbf{F}$. If a vertex is associated with one or more discontinuity edges, its feature varies directionally. The vertex then stores one feature for each side of each such edge. For the query point, we radially interpolate the nearest two edge features, in clockwise and counterclockwise directions respectively. Here, vertex 0 has no associated discontinuity edges and thus stores a single, isotropic feature $\mathbf{F}_0$. Vertex 1 stores two features, $\mathbf{F}_{1,2}^{cw}$ and $\mathbf{F}_{1,2}^{ccw}$, for one (open) discontinuity edge. Vertex 2 stores four features for two discontinuity edges; here we show only the two relevant ones, $\mathbf{F}_{2,1}^{ccw}$ and $\mathbf{F}_{2,3}^{cw}$. We then linearly interpolate using the angles θ_1^{cw} and θ_1^{ccw} between the direction from vertex to query point and the edges to obtain the directionally-varying feature for vertex 1, and using angles θ_2^{cw} and θ_2^{ccw} for vertex 2.

not associated with any discontinuous edge, then $\hat{\mathbf{F}}_i = \mathbf{F}_i$, which is the case for (blue) vertex $i = 0$ in Fig. 7. Otherwise, we find the two nearest discontinuous edges in clockwise and counterclockwise directions w.r.t. the direction from vertex i to the query vertex, with features $\mathbf{F}_i^{cw}$ and $\mathbf{F}_i^{ccw}$ respectively. (For vertex 1 in Fig. 7, which has only one (open) discontinuity edge, the features are $\mathbf{F}_{1,2}^{cw}$ and $\mathbf{F}_{1,2}^{ccw}$.) We then radially interpolate between them (see insets in Fig. 7):

$$\hat{\mathbf{F}}_i = \mathbf{F}_i^{cw} \frac{\theta_i^{ccw}}{\theta_i^{cw} + \theta_i^{ccw}} + \mathbf{F}_i^{ccw} \frac{\theta_i^{cw}}{\theta_i^{cw} + \theta_i^{ccw}}, \tag{2}$$

where θ_i^{cw} and θ_i^{ccw} are the angles between vector $\mathbf{x} - \mathbf{v}_i$ and the corresponding edges, with $\mathbf{x}$ being the query point and $\mathbf{v}_i$ the position of triangle vertex i. For vertex 1, this will simply radially interpolate the clockwise and counter-clockwise features as desired, with a discontinuity in only one direction along the edge from vertex 1 to vertex 2 (corresponding the leftmost curve endpoint in Fig. 5). Vertex 2 has four features, and the closest clockwise/counterclockwise features correspond to two different edges, $\mathbf{F}_{2,1}^{ccw}$ and $\mathbf{F}_{2,3}^{cw}$.

4.3.3 Curved discontinuities. Extending the method above to handle curved discontinuities mostly boils down to properly tracking the discontinuity curve within a triangle and switching to different features when the query point crosses the curve (see Fig. 8).

Since the triangulation process (Section 4.2) ensures that each triangle contains at most one curve, we need to consider at most two regions per triangle. For each triangle T, if it contains a curve, we store a *curve feature* $\mathbf{F}_{i,T}^{curve}$ at triangle vertex i that is opposite to the curved edge, denoted as $\mathbf{F}_{0,T}^{curve}$ in Fig. 8. For the other two vertices, we do not store curve features unless they themselves are the vertex that is opposite to some other curve edge. The curve features are responsible for representing the region between the curve and the triangle edges (middle in Fig. 8).

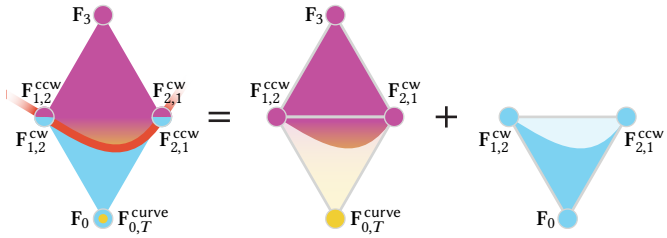


Fig. 8. Curve-feature interpolation. To achieve discontinuity-aware interpolation on a curved triangle (lower triangle here), we decompose the triangle into two regions. For simplicity, we do not show the four features associated with vertex 1 and 2 connecting out to other vertices ($\mathbf{F}_{1,j}^{cw}$, $\mathbf{F}_{1,j}^{ccw}$, $\mathbf{F}_{2,k}^{cw}$, $\mathbf{F}_{2,k}^{ccw}$ for some j, k that are not 1 or 2). We store a *curve* feature $\mathbf{F}_{0,T}^{curve}$ on triangle vertex 0 which is not on the discontinuous edge. Then, if the query point is between the curve and its associated (straight) triangle edge 1, 2 (middle figure), we use the curve feature for vertex 0 and radially interpolate features for vertices 1 and 2 as in Fig. 7 (e.g., for vertex 1, we radially interpolate between feature $\mathbf{F}_{1,2}^{ccw}$ and some other feature $\mathbf{F}_{1,j}^{cw}$). If the query is on the other side of the curve (right figure), we interpolate as in Fig. 7.

Algorithm 1. Query operation of our data structure.

```
1:  function Query(x, mesh)
2:    T, b1, b2 ← PointInTriangle(x, mesh)
3:    for each vertex v_i in triangle T do
4:      if IsContinuous(v_i) then
5:        F̂_i ← F_i
6:      else        ↶ Radially interpolate two closest discontinuous edges (Fig. 7)
7:        F̂_i ← F_i^cw θ_i^ccw/(θ_i^cw+θ_i^ccw) + F_i^ccw θ_i^cw/(θ_i^cw+θ_i^ccw)
8:      if ContainsCurve(T) and
9:          NotOnCurvedEdge(T, v_i) and G_T(x) < 0 then
10:       F̂_i ← F_{i,T}^curve      ← Use curve feature if x is between curve and
                                     triangle edge (Fig. 8)
11:   F = (1 − b1 − b2)F̂_0 + b1 F̂_1 + b2 F̂_2
12:   return MLP(F)
```

Given a query point inside a triangle T, if there is no curve inside the triangle, we proceed as in the linear-edge case (Fig. 7). Otherwise, we check which side of the curve the query point lands on. If it is inside the curve feature region (middle in Fig. 8) and the curve feature is stored in vertex i, then we assign $\hat{\mathbf{F}}_i = \mathbf{F}_{i,T}^{curve}$. For the other two vertices we use the radially-interpolated clockwise/counterclockwise features. Otherwise (right in Fig. 8), we interpolate as in the linear-edge case. To test which side of the curve the query point is on, we employ *implicitization* [Sederberg 1983; Loop and Blinn 2005]: any quadratic/cubic Bézier curve in a triangle T can be converted into an implicit function $G_T(\mathbf{x})$, where the sign of G_T determines which side $\mathbf{x}$ is on.

4.3.4 MLP feature decoding. We pass the query-point feature $\mathbf{F}$, interpolated per Eq. (1), to a multi-layer perceptron to obtain the final signal f:

$$f(\mathbf{x}) = \text{MLP}(\mathbf{F}). \tag{3}$$

We use a shallow MLP, with structure described in Section 5. Our complete inference pipeline is summarized in Algorithm 1.

4.4 Continuity analysis

We show that our interpolation scheme satisfies the continuity criteria in Section 4.1.

Theorem 4.3. *Given the curved triangular mesh that satisfies the criteria in Section 4.2, with a subset of its (curved) edges* $\Gamma = \{\gamma_0, ...\gamma_{n-1}\}$ *marked as discontinuous (with endpoints* $\partial\Gamma$ *as triangle vertices), the feature field reconstructed by our discontinuity-aware interpolation (Section 4.3) satisfies the continuity criteria in Section 4.1.*

Proof. We analyze the continuity at query point $\mathbf{x}$ in four cases:

(1) $\mathbf{x}$ is in the interior of a triangle: the barycentric interpolation ensures the continuity (Condition (1)). The nearest (counter-) clockwise feature does not change in the interior.
(2) $\mathbf{x} \in \Gamma \setminus \partial\Gamma$: Our radial interpolation would create two directional discontinuities at the two tangent directions of the curve as $\mathbf{x}$ will switch to a different region. Since we store different features at the two sides, point $\mathbf{x}$ satisfies Condition (2). For curved edges, the jump discontinuities are ensured by the switch to curve features as $\mathbf{x}$ crosses over the curve.
(3) $\mathbf{x} \in \partial\Gamma$: Suppose $\mathbf{x}$ is connected to n discontinuous edges (if $n = 1$, then $\mathbf{x}$ is at the endpoint of an open discontinuous edge). Consider a circle with sufficiently small radius around $\mathbf{x}$: the circle would intersect all the n edges. This leads to directional discontinuities towards the tangent directions of all the discontinuous edges.
(4) $\mathbf{x}$ is on a triangle edge that is not marked discontinuous: since the barycentric coordinate corresponding to the vertex opposite to the triangle edge is 0 and the radial interpolation does not produce a discontinuity at $\mathbf{x}$, the feature at $\mathbf{x}$ is continuous (Condition (1)).

□

5 IMPLEMENTATION

We implement our algorithm in PyTorch [Paszke et al. 2019]. We use the ADAM Optimizer [Kingma and Ba 2015] with a learning rate of 0.01 and $\beta = (0.9, 0.999)$. We train all models to convergence. This ranges from a couple of minutes for walk on spheres (Section 6.2) to a couple of hours for rendering (Section 6.1).

We use a shallow, two-layer perceptron with 64 neurons and ReLU activation to decode the interpolated features. For rendering (Section 6.1), we use 128 neurons for both layers and for physics-informed neural networks we use the tanh activation function. Additionally, for rendering, we concatenate the interpolated features with a one-blob encoding of $\mathbf{x}$ [Müller et al. 2018].

TriWild [Hu et al. 2019] generates curved triangulations in the gmsh format which we compress using Draco[2]. We use sparse data structures to setup pointers between the features and a global parameter bank. At run time, we populate per-triangle buffers using these data structures for faster lookups. In the supplementary document we provide additional details about the parameters for the curved triangulation and the exact data structures for feature setup.

We use the mean squared error loss function for all applications except rendering, for which we use the relative mean squared error $\|(f(\mathbf{x}) - g(\mathbf{x}))/(f(\mathbf{x}) + 0.01)\|^2$, where f is our network's prediction and g is the ground-truth signal. The gradients are not propagated through the denominator.

For sampling during training, we found standard stratified sampling, either according to a regular grid or the underlying triangulation to be sufficient, even near edges.

[2]https://google.github.io/draco/

Table 1. Triangulation and feature-field statistics. We report the number of triangles, vertices, discontinuous vertices, parameters, and inference time. The parameter count includes those of the feature field and the multi-layer perceptron. The inference time is measured for a $1k^2$-resolution image.

Scene	#Triangles	#Vertices	#Disc. vert.	#Params	Time
Hairball (Fig. 1a)	983,153	492,127	405,600	6,882,584	0.026s
Flowers (Fig. 9)	1,465,050	734,008	156,999	5,639,241	0.033s
Circles (Fig. 10)	11,425	13,481	846	87,504	0.021s
Loop (Fig. 12)	51,673	29,914	811	167,779	0.015s
Helmholtz (Fig. 14)	11,975	6,118	284	39,489	0.015s
Wave eq. (Fig. 13)	22,071	11,189	397	1,237,909	0.047s

6 APPLICATIONS

We showcase applications for fitting 2D signals, from rendering to solving differential equations with discontinuities specified by curves. We categorize them by how we produce the signals.

We compare our method mainly to two recent neural-feature-field approaches, InstantNGP [Müller et al. 2022] and ReLU fields [Karnewar et al. 2022]. InstantNGP excels at fitting high-resolution images and outperforms competing methods such as ACORN [Martel et al. 2021]. ReLU fields produce significantly sharper results than their linear counterparts and have been shown to be effective at fitting sharp edges in binary functions. We show the scene statistics of our method in Table 1.

6.1 Rendering

For rendering, our goal is to produce a resolution-independent representation that maps two-dimensional screen coordinates to RGB-color values [Pighin et al. 1997]. To obtain the location of geometric discontinuities, we project all silhouette edges and geometric-normal discontinuity edges onto the screen and clip them against occluding triangles. The resulting set of edges represents the locations of visibility and normal discontinuities in screen space.

For training data, we use a raster image rendered at $100k^2$ resolution with 512 samples per pixel to approximate the continuous signal. We also tried querying the continuous signal on the fly by retrieving RGB values for given screen coordinates, but opted for the raster image because it achieved similar fidelity with a simpler implementation. Rendering the raster image took 2 hours using Mitsuba 3 [Jakob et al. 2022]. In each epoch we retrieve $10k^2$ stratified pixels from the raster image and use a batch size of 2^{19}.

We demonstrate our approach on two scenes with fine geometric detail, *Hairball* in Fig. 1a and *Rose Bush* in Fig. 9. The $100k^2$ PIZ-compressed ground-truth EXR image in Fig. 9 is 33GB. In contrast, our method yields total file sizes (trainable parameters + mesh) of 25MB and 30MB for the *Rose Bush* and *Hairball* scenes respectively. We adjust the InstantNGP and ReLU-fields sizes to match our file size: hash-table size=2^{19} at 30MB for InstantNGP and grid

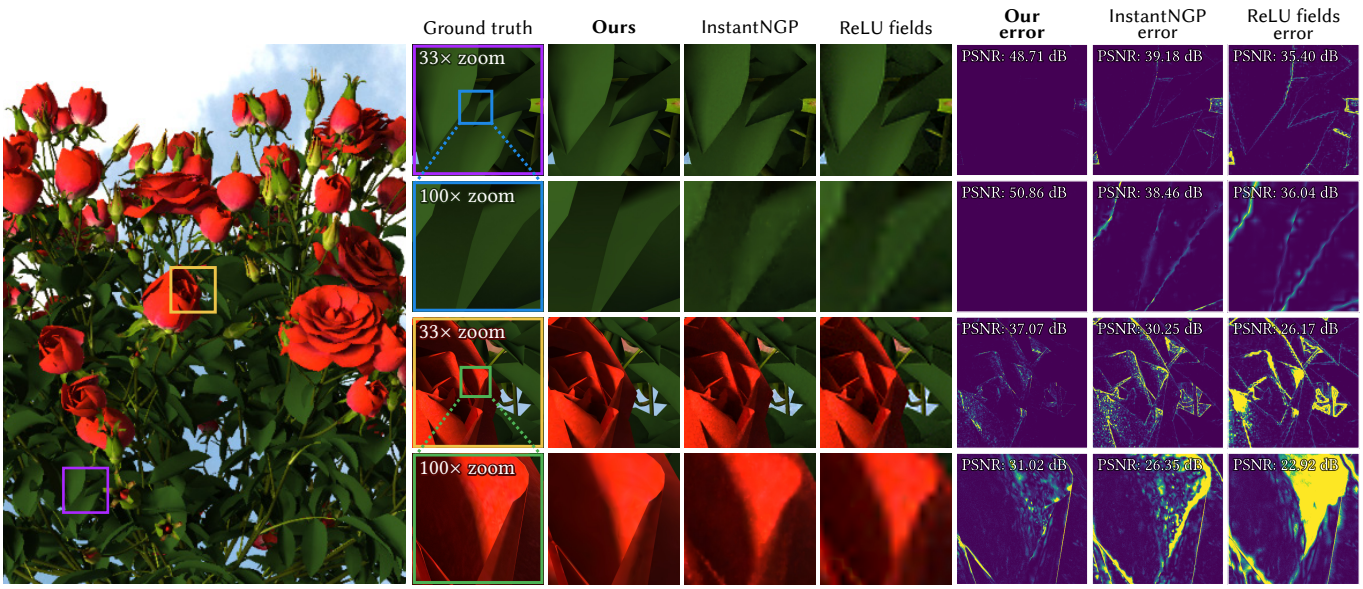


Fig. 9. Fitting rendering images. We fit our representation to a high-resolution ($100k^2$) image, where it is able to preserve sharp contrast across discontinuities at zoom levels of even up to 100×, with an improvement > 10dB in some regions compared to existing neural representations. InstantNGP and ReLU Fields (at equal total file size) provide a continuous approximation to a signal with jump discontinuities, which leads to blurring across edges and creates unpleasant artifacts at high zoom levels that dominate the error. Some delicate textures in the continuous part of the signal (the rose petal, last row) are blurred by all methods because the triangulation (for our method) and grid (for ReLU Fields) are not fine enough, and the hash-table size (for InstantNGP) is not large enough to capture them.

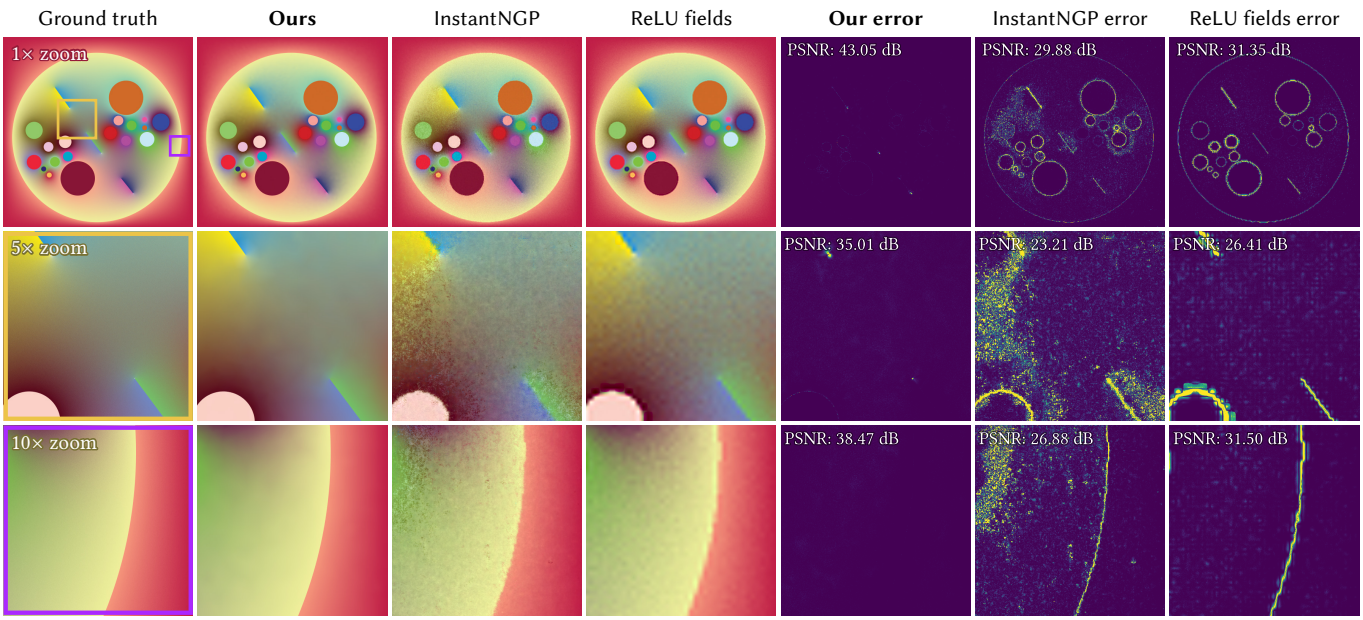


Fig. 10. Monte Carlo diffusion curves. We propose a novel way to solve partial differential equations by fitting our neural representation to the noisy data generated by a Monte Carlo walk-on-spheres solver [Sawhney and Crane 2020]. Here, we show results on solving a diffusion-curve image using only 500 × 500 × 11 noisy Monte Carlo samples. The capability to preserve discontinuities allows us to both maintain sharp edges and produce smoother results outside of the discontinuities, since our data structure requires much less network capacity to represent the edges. Our method automatically *denoises* the noisy input. We analyze the denoising property in Fig. 11. The ground-truth image has been rendered with 10,000 samples per pixel.

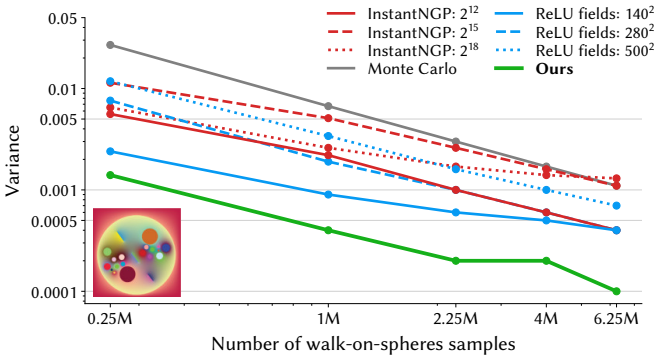


Fig. 11. Our method can denoise Monte Carlo samples generated by walk on spheres better than other (grid-based) neural methods at all sampling rates (for the scene in Fig. 10). We test InstantNGP with several hash-table sizes; it tends to overfit to the noise with an increasing number of entries in the hash table. ReLU fields is less prone to overfitting, but suffers from error due to blurring across edges and performs worse than our method.

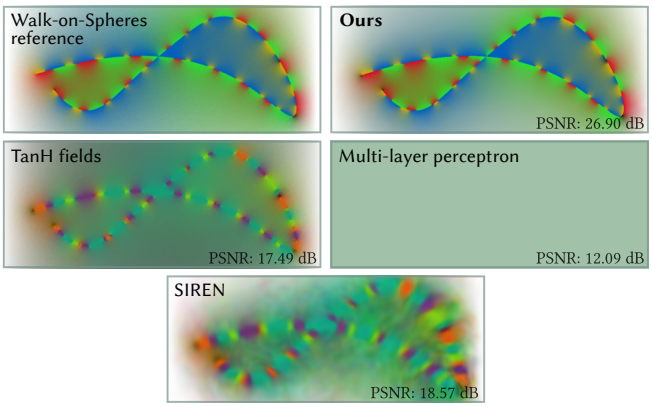


Fig. 12. "Physics-informed" diffusion curves. Our method can be used for directly solving partial differential equations without any training data. Here, we show results on solving a diffusion-curve image, and compare to an architecture similar to ReLU fields but using tanh activation due to ReLU having zero second derivatives everywhere. We also compare to a standard multi-layer perceptron. TanH fields and SIREN smooth over discontinuities, while the multi-layer perceptron completely failed to reconstruct the signal, despite our extensive attempts to tune the hyperparameters.

size=1500^2 at 26MB for ReLU fields. Both InstantNGP and ReLU fields smoothen discontinuities, while ours preserves sharp edges; this is most evident in the 33×- and 100×-zoom crops in which we show a 10dB improvement in Fig. 9. The supplementary video shows a zoom animation on that scene.

Both InstantNGP (0.0012s) and ReLU Fields (0.0009s) have a much faster inference time than our method (0.033s) due to highly optimized and simple implementations respectively, for a $1k^2$ image.

6.2 Walk-on-spheres diffusion curves

Monte Carlo geometry processing [Sawhney and Crane 2020] shows great promise in solving partial differential equations in a resolution-independent way using walk on spheres [Muller 1956]. However, storing the solution in a resolution-dependent raster image throws away a lot of the benefits that Monte Carlo brings in the first place. We fit our representation to the walk-on-spheres samples, enabling resolution-independent solution storage while preserving discontinuities (Fig. 10). Furthermore, since our representation is smooth outside discontinuities, our method automatically *denoises* the noisy Monte Carlo samples [Lehtinen et al. 2018], accelerating the convergence; see Fig. 11. To our knowledge, our method is the first to supervise neural representations with walk-on-spheres data. This enables a new regime for solving partial differential equations.

Specifically, we apply our method to learn a diffusion-curve image [Orzan et al. 2008] which is the solution to the Laplace equation with two-sided Dirichlet boundary conditions defined on a set of Bézier curves. Our training data is obtained from our GPU-based implementation of the walk-on-spheres algorithm.

In Fig. 10, we fit our method to Monte Carlo samples generated at a resolution of 500^2 with 11 samples per pixel. Our method achieves much higher quality (> 10dB) than other neural representations at equal file size since they overfit to noise and smoothen discontinuities. Our method cleanly denoises the Monte Carlo samples (reducing variance by 19.21×) while preserving discontinuities over a wide range of sample counts for the noisy input; see Fig. 11.

It is also possible to render diffusion curves by solving a linear system. However, the linear solve needs to globally discretize the entire canvas, even when zooming into a small region. This can potentially cause problems at extremely high resolutions.

6.3 Physics-informed neural networks

Like physics-informed neural networks [Raissi et al. 2019], our method can directly solve partial differential equations *without any training data*, by sampling points in the space and using a loss function directly from the differential equation. We demonstrate solutions to diffusion curves using this setup.

We minimize the Dirichlet energy $\mathcal{L}_\mathrm{d} = \frac{1}{2}\int_\Omega ||\nabla f||^2$, which is the weak form of the Laplace equation, where f is the output of the neural network. To enforce the boundary constraints, we add an extra loss term $\mathcal{L}_\mathrm{b} = \int_\Gamma ||f - b||^2$, where b is the given Dirichlet boundary condition. To minimize the Dirichlet energy everywhere in Ω, we stratify our samples in the triangles. For the two-sided boundary conditions, we sample two points offset by a distance of 0.001 on either side of the curve in the normal direction. The final loss is $\mathcal{L}_\mathrm{d} + 0.00001\,\mathcal{L}_\mathrm{b}$.

To compare against ReLU fields, which have zero second derivative everywhere, we replace the ReLU activation with TanH. We do not compare against InstantNGP as its implementation does not support second derivatives. We also compare to a SIREN [Sitzmann et al. 2020] network which has 5 layers with 256 neurons each. Finally, we compare to a standard 4-layer MLP, with 512 neurons per layer and TanH activation, which is similar to architectures often used in the physics-informed neural networks literature.

Fig. 12 shows the result. Our method is the only one able to compute a solution that looks close to the ground truth. The two-sided boundary condition we imposed leads to discontinuities at the curves. TanH fields fail to preserve the discontinuities. The MLP

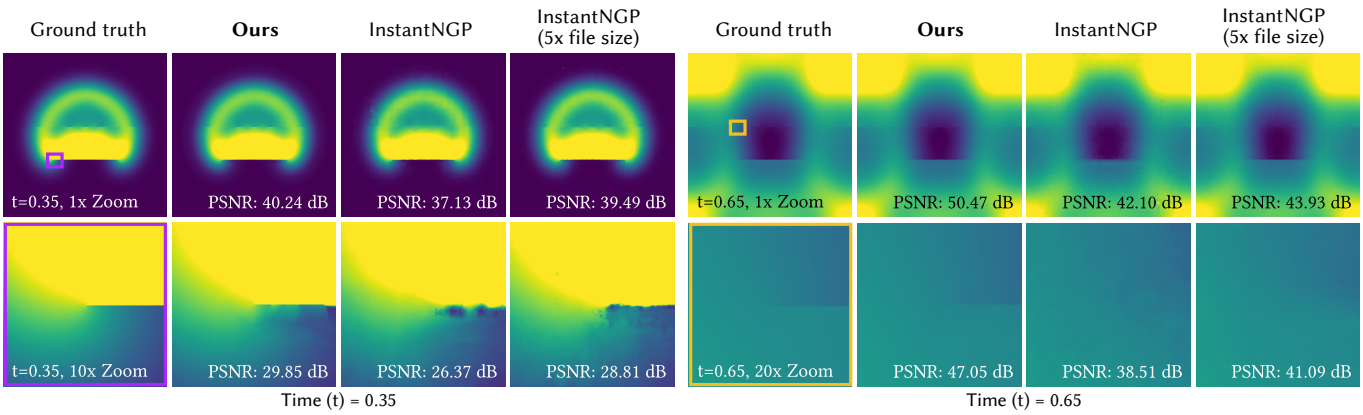


Fig. 13. Fitting to finite-difference solution of wave equation. Our method can compress video sequences such as the high-resolution solution to the wave equation without signal leakage across blockers. The domain has a large linear discontinuity (see $t = 0.35$, second row for zoom in) and a set of smaller slits that diffracts the wave (see t=0.65, zoom-in on second row). Our method preserves sharp contrast across the small slit, whereas InstantNGP with equal file size, or even 5× larger, cannot represent it.

completely fails to converge, despite us having tried a very broad set of hyperparameters. It is well known in the literature that the training of physics-informed neural networks can often be trapped in a local minimum [Krishnapriyan et al. 2021]. It is possible that our explicit representation of discontinuities helps with the training; we leave further investigations as future work.

6.4 Compressing finite-element/difference data

Our representation can also be used for fitting very high-resolution solutions obtained from finite-element or finite-difference methods, in a discontinuity preserving manner.

Discontinuities are very common in these settings, e.g., an absorbing or reflecting object in the center of a room could block a sound wave from freely propagating. Blurring across discontinuities in this setting can completely change the meaning of the signal, such as leaking sound past a blocker, allowing fluid to flow through a barrier, etc. This blurring makes current neural- and grid-based approaches unsuitable in these settings. Our method can compress the solution while also providing guarantees of not leaking the signal past barriers provided in the input discontinuity curves.

Helmholtz equation. The Helmholtz equation is the time-independent solution to the wave equation that is fundamental to acoustics or wave-optics simulation in computer graphics [James 2016]. It shows high-frequency ripples whose frequency increases with the wavenumber k. We compute the complex-valued target signal using the finite-element method with $590k$ triangles and a file size of 19MB. The medium inside the square domain with a linear barrier is air and the wavenumber is $k = 40$.

For training, we use nearest-neighbour interpolation (which preserves linear discontinuities) on the ground-truth mesh. We stratify our samples uniformly within all triangles in our triangulation and use grid-based stratified sampling for InstantNGP. Our method has a file size of 740KB, and we adjust InstantNGP's hash table to be of equal size in Fig. 14. We additionally train another version of InstantNGP with 4× the file size, which approximates edges better

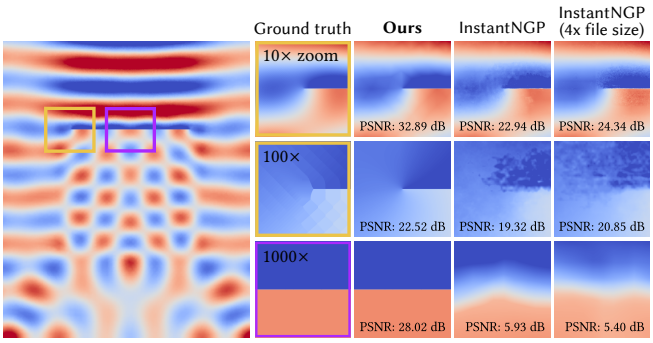


Fig. 14. Fitting to finite-element solution of Helmholtz equation. Given data computed using a finite-element method for solving the Helmholtz equation, our representation compresses the solution from 19MB to 740KB, while guaranteeing no signal leakage across the linear barrier. We show the real part of the solution; the complex part has similar characteristics. Equal-file-size InstantNGP blurs across the barrier and leaks signal across it. Increasing the hash-table size improves the results; however, it still blurs across edges as one zooms in further. Interestingly, the smoothness of our representation enables it to automatically remove the nearest-neighbor sampling artifacts in the ground truth (see second row).

at lower zoom levels. However, upon zooming in further, it shows blurring across edges.

Wave equation. Next, we apply our method to compress the time-varying solution to the wave equation, again defined on a square domain with several static blockers. There are several regularly spaced slits in the domain to emulate slit diffraction and a large linear barrier at the bottom; see Fig. 13. The sound wave originates at a point source. We again use nearest-neighbor interpolation to retrieve training data from the finite-difference method.

Our feature field has thus far been 2D. To extend it temporally, we replicate the same curved triangulation for different time instances, for each initializing the vertex features independently. To resolve a query, we linearly interpolate the features obtained through our

discontinuity-aware interpolation from the two closest time instances. Since the blockers are static, we do not smooth over any discontinuities by interpolating over time.

We find that our representation can correctly prevent sound from leaking across barriers by maintaining sharp discontinuities. It also maintains high fidelity while compressing the original signal from 303MB down to 6MB. We show significant improvement in the quality over equal and 5× file-size InstantNGP. The supplemental video shows an animation of the waves, also a zoomed-in version.

7 LIMITATIONS AND FUTURE WORK

Performance. Due to our use of an irregular mesh and a proof-of-concept implementation, our inference times are slower than the highly efficient and optimized modern feature fields. We expect that integrating our triangle-mesh data structure into a GPU rasterizer would significantly speed up both training and inference. Even with these improvements, feature fields stored on regular grids can still have faster inference times due to the regularity and locality.

Discontinuity locations. Our method assumes prior knowledge of the discontinuity locations. Closely integrating image-vectorization [Reddy et al. 2021] or discontinuity meshing with our representation, and updating edge locations using differentiable rendering [Li et al. 2020; Bangaru et al. 2021] could be an exciting avenue.

Level of detail. Our representation does not properly handle multi-scale data, since we do not model different scales; see Fig. 15. Future work involves investigating multi-resolution mesh structures [Hoppe 1996] for neural fields.

8 CONCLUSION

We present a hybrid neural-mesh representation for 2D signals with discontinuities. Unlike most existing neural fields which are by construction continuous, our representation can handle signals that contain both discontinuities and smooth (color) variations, allowing extreme levels of zooming. Our method opens up a wide variety of applications, including a new image representation for rendering, storing and solving diffusion-curve images, a new way to combine walk-on-spheres Monte Carlo partial-differential-equation solvers with neural representations, and high-fidelity compression of 2D physics-simulation data.

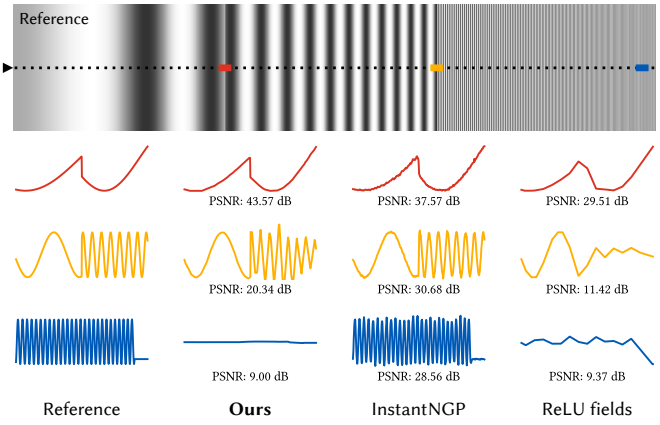


Fig. 15. An increasing-frequency signal given by $\sin((100 + f)x^2)$, with $f = \{50, 150, 1250\}$ in the three horizontal segments (top), and three 1D horizontal slices which are each 2% of the total width. Our method uses 11,274 triangles, and the 1D horizontal slice intersects about 150 edges. It handles discontinuities and low-to-medium frequency signal variation well (red and yellow). However, both our method and ReLU fields store features at a single scale, so they struggle to capture the high-frequency variation (blue). On the other hand, InstantNGP store features at multiple scales, so it can better resolve higher frequencies (blue). However, their hash collisions lead to spurious high-frequency variations in low-frequency regions (red).

ACKNOWLEDGEMENTS

This work was partially completed during the first author's internship at Adobe. It was supported in part by NSF grants 2105806 and 2238839, gifts from Adobe, the Ronald L. Graham Chair, and the UC San Diego Center for Visual Computing. We thank Wesley Chang, Sharv Laad and Mohammad Sina Nabizadeh for discussions and Sharon Lin for help with illustrations during this project.

REFERENCES

Marco Agus, Enrico Gobbetti, José Antonio Iglesias Guitián, and Fabio Marton. 2010. Split-Voxel: A Simple Discontinuity-Preserving Voxel Representation for Volume Rendering.. In *International Symposium on Volume Graphics*. 21–28.

Kavita Bala, Bruce Walter, and Donald P Greenberg. 2003. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 22, 3 (2003), 631–640.

Sai Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically Differentiating Parametric Discontinuities. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 107 (2021), 107:1–107:17.

Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew Duvall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. 2020. Immersive Light Field Video with a Layered Mesh Representation. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4, Article 86 (2020), 15 pages.

Jiawen Chen, Sylvain Paris, Jue Wang, Wojciech Matusik, Michael Cohen, and Fredo Durand. 2011. The video mesh: A data structure for image-based three-dimensional video editing. In *International Conference on Computational Photography*. 1–8.

Yinbo Chen, Sifei Liu, and Xiaolong Wang. 2021. Learning continuous image representation with local implicit image function. In *Computer Vision and Pattern Recognition*. 8628–8638.

Paul Heckbert. 1992. Discontinuity meshing for radiosity. In *Eurographics Workshop on Rendering*. 203–226.

Hugues Hoppe. 1996. Progressive meshes. In *SIGGRAPH*. 99–108.

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TriWild: Robust Triangulation with Curve Constraints. *ACM Trans. Graph. (Proc. SIGGRAPH)* 38, 4, Article 52 (2019), 15 pages.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022. *Mitsuba 3 renderer*. https://mitsuba-renderer.org.

Doug L. James. 2016. Physically Based Sound for Computer Animation and Virtual Environments. In *SIGGRAPH Courses*. Article 22, 8 pages.

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. ReLU fields: The little non-linearity that could. In *SIGGRAPH Conference Proceedings*. 1–9.

Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F. Cohen. 2007. Capturing and Viewing Gigapixel Images. *ACM Trans. Graph. (Proc. SIGGRAPH)* 26, 3 (2007), 93.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. 2021. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 26548–26560.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. 2018. Noise2Noise: Learning Image Restoration without Clean Data. In *International Conference on Machine Learning*.

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 193:1–193:15.

Dani Lischinski, Filippo Tampieri, and Donald P Greenberg. 1992. A discontinuity meshing algorithm for accurate radiosity. *IEEE Comput. Graph. Appl.* 12, 4 (1992), 10–1109.

Charles Loop and Jim Blinn. 2005. Resolution independent curve rendering using programmable graphics hardware. *ACM Trans. Graph. (Proc. SIGGRAPH)* 24, 3 (2005), 1000–1009.

Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. ACORN: Adaptive Coordinate Networks for Neural Scene Representation. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4, Article 58 (2021).

Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. 2021. Modulated periodic activations for generalizable local functional representations. In *International Conference on Computer Vision*. 14214–14223.

Nicolas Moës, John Dolbow, and Ted Belytschko. 1999. A finite element method for crack growth without remeshing. *International journal for numerical methods in engineering* 46, 1 (1999), 131–150.

Mervin E Muller. 1956. Some continuous Monte Carlo methods for the Dirichlet problem. *The Annals of Mathematical Statistics* (1956), 569–589.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph. (Proc. SIGGRAPH)* 41, 4, Article 102 (2022).

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2018. Neural Importance Sampling. *arXiv:1808.03856* (2018).

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-shaded Images. *ACM Trans. Graph. (Proc. SIGGRAPH)* 27, 3 (2008), 92:1–92:8.

Evgueni Parilov and Denis Zorin. 2008. Real-time rendering of textures with feature curves. *ACM Trans. Graph.* 27, 1 (2008), 1–15.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. 8024–8035.

Darko Pavić and Leif Kobbelt. 2010. Two-Colored Pixels. 29, 2 (2010), 743–752.

Frederic H Pighin, Dani Lischinski, and David Salesin. 1997. Progressive Previewing of Ray-Traced Images Using Image Plane Disconinuity Meshing. *Rendering Techniques (Proc. EGWR)* 97 (1997), 115–125.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707.

Ganesh Ramanarayanan, Kavita Bala, and Bruce Walter. 2004. Feature-Based Textures. In *Eurographics Workshop on Rendering*.

Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. 2021. Im2Vec: Synthesizing vector graphics without vector supervision. In *Computer Vision and Pattern Recognition*. 7342–7351.

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 4 (2013), 130.

Alexander Reshetov and David Luebke. 2016. Infinite resolution textures. In *High Performance Graphics*. 139–150.

Mike Salisbury, Corin Anderson, Dani Lischinski, and David H Salesin. 1996. Scale-dependent reproduction of pen-and-ink illustrations. In *SIGGRAPH*. 461–468.

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo geometry processing: A grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4 (2020).

Thomas Warren Sederberg. 1983. *Implicit and parametric curves and surfaces for computer aided geometric design*. Ph. D. Dissertation. Purdue University.

Peter Selinger. 2003. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf

Pradeep Sen. 2004. Silhouette maps for improved texture magnification. In *Graphics Hardware*. 65–73.

Pradeep Sen, Mike Cammarano, and Pat Hanrahan. 2003. Shadow Silhouette Maps. *ACM Trans. Graph. (Proc. SIGGRAPH)* 22, 3 (2003), 521–526.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. *Advances in Neural Information Processing Systems* 34, 6087–6101.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, Vol. 33. 7462–7473.

Ying Song, Jiaping Wang, Li-Yi Wei, and Wencheng Wang. 2016. Vector Regression Functions for Texture Compression. *ACM Trans. Graph.* 35, 1, Article 5 (2016).

Marco Tarini and Paolo Cignoni. 2005. Pinchmaps: textures with customizable discontinuities. *Comput. Graph. Forum (Proc. Eurographics)* (2005).

Jack Tumblin and Prasun Choudhury. 2004. Bixels: Picture Samples with Sharp Embedded Boundaries. *Rendering Techniques (Proc. EGWR)* (2004).

Edgar Velazquez-Armendariz, Eugene Lee, Kavita Bala, and Bruce Walter. 2006. Implementing the render cache and the edge-and-point image on graphics hardware. In *Graphics Interface*. 211–217.

Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. 2014. Hierarchical Diffusion Curves for Accurate Automatic Image Vectorization. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 33, 6 (2014), 230:1–230:11.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. *Comput. Graph. Forum (Proc. Eurographics STAR)* 41, 2 (2022), 641–676.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for real-time rendering of neural radiance fields. In *International Conference on Computer Vision*. 5752–5761.

Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. 2021. NTopo: Mesh-free Topology Optimization using Implicit Neural Representations. In *Advances in Neural Information Processing Systems*, Vol. 34. 10368–10381.