TetraSDF: Analytic Isosurface Extraction with Multi-resolution Tetrahedral Grid

Seonghun Oh[1*], Youngjung Uh[1], and Jin-Hwa Kim[2,3†]

[1] Yonsei University, Republic of Korea
[2] NAVER AI Lab, Republic of Korea
[3] Seoul National University, Republic of Korea
{rendell,yj.uh}@yonsei.ac.kr, j1nhwa.kim@navercorp.com

Abstract. Extracting an explicit surface that exactly matches the zero-level set of a neural signed distance function (SDF) remains challenging. Sampling-based isosurfacing methods such as Marching Cubes introduce discretization error. In contrast, continuous piecewise affine (CPWA) analytic approaches typically require plain ReLU MLPs, which limits the ability to learn high-frequency SDFs in practice. We present TetraSDF, an analytic isosurface extraction framework for SDFs that retains the expressiveness of grid-based encoders while enabling exact zero-level set extraction, by representing the SDF with a ReLU MLP composed with a multi-resolution tetrahedral positional encoder. Our positional encoder's barycentric interpolation preserves a global CPWA structure, allowing us to track ReLU linear regions within an encoder-induced polyhedral complex. We further introduce a fixed analytic input preconditioner derived from the encoder's metric to reduce directional bias, thereby stabilizing training. Across multiple benchmarks, TetraSDF matches or surpasses existing grid-based encoders in SDF reconstruction accuracy, while faithfully recovering the network's zero-level set as a triangle mesh.

1 Introduction

Extracting explicit isosurfaces (*e.g.*, triangle meshes) from learned SDFs is a standard step in neural implicit surface pipelines. Learned SDFs are widely used for distance- and collision-based queries, while many downstream tools and pipelines consume explicit surface representations, making isosurface extraction a common interface step. Ensuring that the extracted surface matches the SDF's zero-level set helps preserve geometric consistency. As a result, downstream objectives defined on the surface, such as normal alignment to field-derived normals or collision checking against the extracted surface, remain faithful to the network's isosurface representation.

Classical methods such as Marching Cubes [23], Marching Tetrahedra [43], and Dual Contouring [14] discretize the implicit field defining the isosurface, inevitably losing geometric fidelity due to their resolution-dependent sampling.

[*] Work done during an internship at NAVER AI Lab. [†] Corresponding author.
Project page: https://seonghunn.github.io/tetrasdf/

The resulting mismatch between the extracted mesh and the network's isosurface cannot be eliminated at finite sampling resolution, and reducing the mismatch inflates triangle budgets and may introduce staircase artifacts. Analytic Marching [21] later addressed this issue by interpreting ReLU-based MLPs as continuous piecewise affine (CPWA) functions [13, 25, 29, 35], allowing exact isosurface extraction from the network's zero-level set. However, its complexity scales poorly with network expressiveness, making it impractical. Subsequent work refines this with an edge subdivision [1] that leverages the sign vectors to identify the parent cells of boundary vertices without iterating all cells, improving both precision and efficiency.

Although these analytic methods yield explicit isosurfaces that exactly match the networks' zero-level sets, they are limited to ReLU MLPs where CPWA analysis applies. Such networks are prone to spectral bias [30, 40], making it difficult to learn complex SDFs. TropicalNeRF [16] attempts to retain the expressiveness of grid-based positional encoders while enabling analytic extraction via a piecewise trilinear interpretation. However, for grid-based encoders using trilinear interpolation [3, 26], its analytic guarantee relies on enforcing an eikonal constraint within each trilinear cell, which is difficult to satisfy uniformly, and thus uses a diagonal-plane intersection heuristic; consequently, the extracted isosurface is not guaranteed to exactly match the network's zero-level set in general.

To overcome the precision loss caused by non-affine interpolation while still avoiding spectral bias, we introduce a network architecture that combines a multi-resolution tetrahedral positional encoder with a ReLU MLP, exploiting the affine nature of barycentric interpolation. This design preserves the benefits of grid-based positional encoders for learning high-frequency SDFs, while ensuring that the overall mapping remains CPWA and thus amenable to analytic isosurface extraction. Notice that, unlike axis-aligned cuboids, the tetrahedral encoder partitions space into *polyhedral cells*, which are then further subdivided by the folded hyperplanes of the ReLU MLP.

Building on this observation, we propose a novel analytic isosurface extraction framework for networks employing our multi-resolution tetrahedral positional encoder with a ReLU MLP. While the positional encoder enhances the shape expressiveness, the extracted isosurface still exactly matches the network's zero-level set due to the CPWA nature of the overall mapping. The encoder is designed to support efficient cell indexing and neighbor identification, which are essential for analytic traversal of the polyhedral structure.

Furthermore, we extend the edge subdivision algorithm to operate jointly on encoder-induced polyhedral cells and ReLU MLP linear regions, enabling exact CPWA analysis under positional encoding. We also introduce an analytic input preconditioner for the multi-resolution tetrahedral encoder that mitigates directional bias induced by the encoder's grid geometry, improving optimization conditioning during training. In summary, our contributions are as follows:

- We propose TetraSDF, a framework that bridges expressive SDF learning and exact analytic isosurface extraction by preserving a global CPWA structure via barycentric interpolation in a tetrahedral positional encoder.

- We propose a tensorized scheme for constructing and handling the resulting polyhedral structures, enabling an efficient GPU implementation.
- We characterize directional bias induced by the encoder's geometry via its metric and derive an input preconditioner to mitigate it, improving SDF learning accuracy.

2 Related Work

2.1 Isosurface Extraction from Neural Implicit Fields

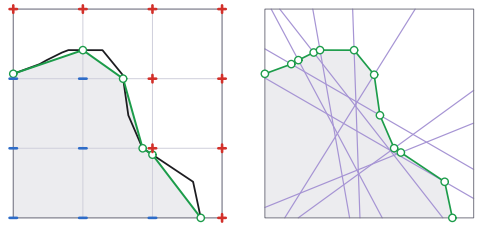
In neural implicit representations, shapes are modeled as continuous scalar fields parameterized by neural networks [24, 27]. Classical isosurface extraction methods such as Marching Cubes and its variants [14, 23, 43] convert these fields into meshes via sampling. However, the resulting surfaces suffer from discretization error, and reducing this error requires dense sampling, which rapidly increases mesh complexity. Recent finite-sample surface reconstruction and adaptive isosurfacing methods further improve explicit surface recovery from sampled or queried fields [19,31,34]. While effective, these methods still recover surfaces from finite or adaptive field evaluations, rather than directly from the continuous neural field represented by the trained network itself. To improve fidelity or enable learning-based extraction, several works integrate neural networks with isosurface extraction or differentiable mesh parameterizations [4, 5, 9, 36, 37]. These approaches can produce higher-quality meshes and support end-to-end training, but still rely on discrete sampling or surrogate volumetric parameterizations as the basis for mesh extraction. Analytic meshing methods instead exploit the CPWA structure of ReLU MLPs [1,21,39] or piecewise trilinear formulations [16] to more directly characterize the network's zero-level sets, avoiding reliance on sampling. However, in practice, the former are restricted to plain MLPs with limited ability to represent complex SDFs, while the latter relies on geometric heuristics that can degrade precision.

2.2 Positional Encoding Methods

Learning high-frequency geometric details in neural implicit functions is often hindered by spectral bias [30, 40]. To mitigate this effect, positional encoding schemes have been proposed to map input coordinates into higher-dimensional feature spaces with rich frequency content [38, 40] to overcome spectral bias. Beyond frequency-based methods, grid-based positional encoders have become dominant for large-scale 3D scenes, where learnable features are stored in spatial grids and interpolated at query points [3, 8, 22, 26]. Among grid-based approaches, Tetra-NeRF [20] employs an adaptive tetrahedral representation and PermutoSDF [32] leverages a permutohedral lattice; both use barycentric-style interpolation of features within simplex cells. However, the data-dependent tetrahedralization in Tetra-NeRF and the higher-dimensional embedding in PermutoSDF make explicit cell and neighbor indexing less straightforward for analytic extraction.

2.3 Preconditioning

Preconditioning is a classical technique from numerical linear algebra [10, 33] that improves the convergence of iterative solvers by transforming ill-conditioned systems into better-conditioned forms. This concept has been widely adopted in deep learning to stabilize training and accelerate convergence through gradient or parameter preconditioning via adaptive per-parameter scaling [12,17,42]. ZCA whitening [44] similarly normalizes the covariance of input features to promote more isotropic learning behavior in the data space. Recent works further explore preconditioning in structured domains such as implicit neural fields and camera parameter spaces [6, 28]. In the same spirit, we derive a fixed analytic input preconditioner that whitens the encoder-induced metric, thereby removing the directional bias introduced by the encoder's geometric structure.



(a) Sampling-based extraction (b) Analytic extraction

Fig. 1: A schematic 2D illustration of sampling-based and analytic isosurface extraction for CPWA functions, such as ReLU MLPs. Sampling-based extraction approximates the network's zero-level set from finitely sampled field values, whereas analytic extraction recovers the zero-level set by exploiting the CPWA structure. Black denotes the network's zero-level set, and green denotes the extracted surface.

3 Preliminary

3.1 Definitions

Definition 1 (Tetrahedral networks). *A* tetrahedral network $f = \nu^{(M)} \circ \tau$ *is the composition of a multi-resolution tetrahedral positional encoder* τ *and a* M*-layer ReLU network* $\nu^{(M)}$*.* $\nu^{(M)}$ *is defined as follows:*

$$\nu^{(M)} = \rho^{(M)} \circ \sigma^{(M-1)} \circ \rho^{(M-1)} \circ \cdots \circ \sigma^{(1)} \circ \rho^{(1)}. \tag{1}$$

Here, $\sigma(\mathbf{x}) = \max(\mathbf{x}, 0)$ *and* $\rho^{(i)}(\mathbf{x}) = W^{(i)}\mathbf{x} + b^{(i)}$*.*

Remark 1 (Piecewise affine property of tetrahedral networks). The encoder τ maps an input $\mathbf{x} \in \mathbb{R}^3$ to a feature vector by performing barycentric interpolation at each level of the multi-resolution tetrahedral positional encoder and

concatenating the resulting feature vectors. Since barycentric interpolation is affine, the interpolated feature vector at each level is affine in $\mathbf{x}$ within the containing tetrahedron. Moreover, because the concatenation of affine functions is affine, the output of τ remains affine on such regions (for detailed proofs, see Appendix A.1). Consequently, τ defines a piecewise affine mapping over the input domain. The ReLU network ν is also piecewise affine in its input, so its composition, the tetrahedral network f is piecewise affine w.r.t. $\mathbf{x}$.

Definition 2 (Sign vectors). *Let $\nu_k^{(m)}$ denote the k-th neuron in the m-th layer of an M-layer ReLU network $\nu^{(M)}$, and let $\nu_k^{(m)}(\mathbf{x})$ denote its pre-activation value at input $\mathbf{x}$. The* sign vector *associated with $\mathbf{x}$ at that neuron is defined as:*

$$\mathbf{s}_k^{(m)}(\mathbf{x}) = [\,\gamma_1^{(1)}(\mathbf{x}),\, \gamma_2^{(1)}(\mathbf{x}),\, \ldots,\, \gamma_k^{(m)}(\mathbf{x})\,], \tag{2}$$

where each scalar entry $\gamma_j^{(i)}(\mathbf{x})$ is defined using a small positive constant ϵ_s as:

$$\gamma_j^{(i)}(\mathbf{x}) = \begin{cases} +1, & \text{if } \nu_j^{(i)}(\mathbf{x}) > \epsilon_s, \\ 0, & \text{if } |\nu_j^{(i)}(\mathbf{x})| \le \epsilon_s, \\ -1, & \text{if } \nu_j^{(i)}(\mathbf{x}) < -\epsilon_s. \end{cases} \tag{3}$$

Here, $\gamma_j^{(i)}(\mathbf{x}) = 0$ indicates that $\mathbf{x}$ lies on the decision boundary $\mathcal{H}_j^{(i)} = \{\mathbf{x} \mid \nu_j^{(i)}(\mathbf{x}) = 0\}$. The values $+1$ and -1 correspond to the two half-spaces, i.e.*, the two linear regions on either side of this boundary.*

3.2 Edge Subdivision

Through successive ReLUs, each decision boundary $\mathcal{H}_k^{(m)}$ becomes a *folded hyperplane* [1,11] in the input space. Starting from an initial skeleton with vertices $\mathcal{V}$ and edges $\mathcal{E}$ (*e.g.*, a cube bounding the scene), we sequentially process all boundaries $\mathcal{H}_k^{(m)}$ in increasing order of (m, k), updating $\mathcal{V}$ and $\mathcal{E}$ at each step. For each boundary $\mathcal{H}_k^{(m)}$, every intersected edge is subdivided. A sign change of $\gamma_k^{(m)}$ at the two endpoints indicates that the edge crosses $\mathcal{H}_k^{(m)}$ and that its vertices lie in different linear regions. For an edge $(\mathbf{x}_0, \mathbf{x}_1)$ satisfying $\gamma_k^{(m)}(\mathbf{x}_0)\,\gamma_k^{(m)}(\mathbf{x}_1) < 0$, let $\nu_k^{(m)}(\mathbf{x}_0) = d_0$ and $\nu_k^{(m)}(\mathbf{x}_1) = d_1$. Then, the intersection is computed as:

$$\hat{\mathbf{x}}_{0,1} = (1 - w)\mathbf{x}_0 + w\mathbf{x}_1, \quad w = |d_0|/|d_0 - d_1|. \tag{4}$$

The intersection point $\hat{\mathbf{x}}_{0,1}$, satisfying $\gamma_k^{(m)}(\hat{\mathbf{x}}_{0,1}) = 0$, is added to $\mathcal{V}$, and the corresponding edge in $\mathcal{E}$ is split into two edges, $(\mathbf{x}_0, \hat{\mathbf{x}}_{0,1})$ and $(\hat{\mathbf{x}}_{0,1}, \mathbf{x}_1)$.

The tricky part, however, is that, when a folded hyperplane subdivides linear regions (convex polyhedra), the induced intersection polygon determines which new edges must be added to $\mathcal{E}$ (see Fig. 2c). To handle this consistently without explicitly enumerating all linear regions, we employ the *sign vectors* [1] defined in Definition 2. A candidate edge formed by two previously computed intersection

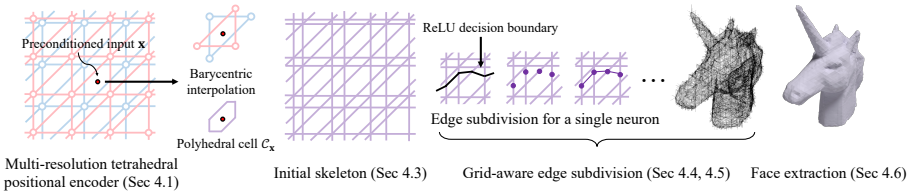


Fig. 2: Overview of *TetraSDF*. A preconditioned input $\mathbf{x}$ is mapped by the multi-resolution tetrahedral positional encoder to interpolated features within its containing polyhedral cell $\mathcal{C}_\mathbf{x}$ (Secs. 4.1 and 4.2). These cells form the encoder-induced polyhedral complex (the initial skeleton), from which we start edge subdivision (Sec. 4.3). We then perform grid-aware edge subdivision that jointly tracks polyhedral cells and ReLU MLP linear regions to obtain the candidate vertex and edge sets $(\mathcal{V}, \mathcal{E})$ (Secs. 4.4 and 4.5), and finally extract mesh faces from $(\mathcal{V}, \mathcal{E})$ to obtain the final mesh (Sec. 4.6). In (a)–(c), we zoom into a single edge subdivision iteration for a neuron.

points is considered valid only when the following conditions are satisfied: First, their sign vectors must match on all nonzero entries, ensuring that both points belong to the same linear region. Notice that a zero entry is treated as a wildcard that can agree with either -1 or $+1$ (for the details, see Sec. 3.3). Second, if a vertex pair shares at least two zero entries in their sign vectors, the pair is connected, since both vertices lie on the same pair of hyperplanes. For a high-level intuition and a step-by-step summary, see Appendix A.7.

Notice that, when integrated with our multi-resolution tetrahedral positional encoder (Sec. 4.1), we augment the sign vector by appending the grid-based region indicator (Sec. 4.4) to maintain consistency, while the rest of the procedure remains unchanged.

3.3 Perturbation with Sign Vectors

Perturbing each zero entry in a sign vector to both $+1$ and -1 generates the sign vectors of all neighboring parent linear regions. This provides a systematic way to explore local adjacency relationships between regions. We later combine this perturbation scheme with the grid-based region indicators introduced in Sec. 4.5, enabling efficient adjacency tracking across the tetrahedral network structure.

4 Method

We introduce the design of the multi-resolution tetrahedral positional encoder τ and its input preconditioner. The subsequent sections describe our mesh extraction pipeline, which consists of 1) constructing the initial skeleton, 2) performing grid-aware edge subdivision, and 3) reconstructing the surface.

4.1 Multi-resolution Tetrahedral Positional Encoder

Grid-based positional encoders are a common approach to mitigate spectral bias for ReLU MLPs, but a widely used design choice based on trilinear interpolation [3, 26] breaks the CPWA structure and hinders analytic extraction. Our

multi-resolution tetrahedral encoder replaces it with barycentric interpolation, preserving high-frequency modeling while keeping the overall mapping CPWA.

Definition 3 (Multi-resolution tetrahedral grid). *Let $L \in \mathbb{N}$ be the number of resolution levels, and $N_{\min}$ and $N_{\max}$ be the minimum and maximum grid resolutions, respectively, where $N_{\min} < N_{\max}$. Then, the geometric progression ratio γ is defined as follows [26]:*

$$\gamma := \exp\left(\frac{\ln N_{\max} - \ln N_{\min}}{L-1}\right), \tag{5}$$

while the ℓ-level resolution N_ℓ is defined as follows:

$$N_\ell := \left\lfloor N_{\min}\,\gamma^\ell \right\rfloor, \quad \ell \in \{0, 1, \ldots, L-1\}. \tag{6}$$

At each level ℓ, the unit cube $\mathcal{S} \in [0,1]^3$ is uniformly subdivided into N_ℓ^3 cube cells, each of which is further decomposed into six congruent tetrahedra following the tetrahedral convention [2] in Fig. 3. We further discuss the design rationale in Appendix A.6.1.

Definition 4 (Tetrahedral positional encoder). *For a query point $\mathbf{x} \in \mathcal{S}$, let $\mathcal{T}^{(\ell)}(\mathbf{x})$ be any[4] tetrahedron at level ℓ that contains $\mathbf{x}$, and $V(\mathcal{T}^{(\ell)}(\mathbf{x}))$ be its set of four vertices. Each vertex $v \in V(\mathcal{T}^{(\ell)}(\mathbf{x}))$ is mapped to the index $i = h(v)$ for the hash table via the spatial hash function h [41]. The entry of index i corresponds to the learnable feature vector $H_i^{(\ell)} \in \mathbb{R}^d$. The barycentric weight[5] of $\mathbf{x}$ with respect to v is denoted by $w_v^{(\ell)}(\mathbf{x})$. Then, the tetrahedral positional encoder $\tau : \mathbb{R}^3 \to \mathbb{R}^{dL}$ is defined as:*

$$\tau(\mathbf{x}) := \bigoplus_{\ell=0}^{L-1} \left(\sum_{v \in V(\mathcal{T}^{(\ell)}(\mathbf{x}))} w_v^{(\ell)}(\mathbf{x})\, H_i^{(\ell)} \right), \tag{7}$$

where $\oplus$ denotes concatenation over all resolution levels.

Definition 5 (Polyhedral cells). *For $\mathbf{x} \in \mathcal{S}$ and each level ℓ, the choice of $\mathcal{T}^{(\ell)}(\mathbf{x})$ determines which feature vectors are selected at that level. We define the* polyhedral cell *of $\mathbf{x}$ as the maximal region on which this combination of selected feature vectors across all levels remains constant:*

$$\mathcal{C}_{\mathbf{x}} := \bigcap_{\ell=0}^{L-1} \mathcal{T}^{(\ell)}(\mathbf{x}). \tag{8}$$

[4] Singular cases where $\mathbf{x}$ lies on the boundaries are negligible, as they are handled implicitly by our interpolation scheme.

[5] Barycentric coordinates specify a point with respect to the vertices of a simplex by expressing the point as a convex combination of the vertices, $\mathbf{x} = \sum_i w_i v_i$ with $\sum_i w_i = 1$ and $w_i \geq 0$. The tuple $(w_0, \ldots, w_n)$ is called the barycentric coordinates of $\mathbf{x}$, and each coefficient w_i is the barycentric weight associated with the vertex v_i.

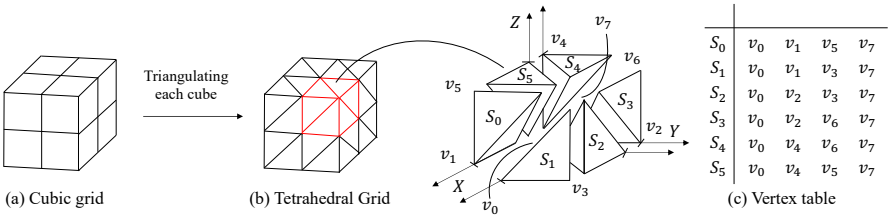


Fig. 3: At each level, each cube cell is decomposed into six congruent tetrahedra.

where $\bigcap_{\ell=0}^{L-1}$ *denotes the geometric intersection over all resolution levels* ℓ *of* $\mathbf{x}$*-containing tetrahedra. The collection of all cells over* $\mathbf{x} \in \mathcal{S}$ *can be written as* $\mathcal{C} := \bigcup_{\mathbf{x}\in\mathcal{S}} \mathcal{C}_{\mathbf{x}}$*, which forms a polyhedral complex that partitions* $\mathcal{S}$*. On each cell* $\mathcal{C}_{\mathbf{x}}$*, the encoder* τ *is affine. Polyhedral cell boundaries are where* τ *switches affine regions, forming fixed partitions that further refine the regions by the subsequent ReLU network. Unlike the adaptive partitions learned by the ReLU layers, those of the tetrahedral positional encoder are determined by hyperparameters and remain fixed during training.*

4.2 Input Preconditioning

In our positional encoder, the barycentric coordinates $\mathbf{w}(\mathbf{x}) = [w_1(\mathbf{x}), w_2(\mathbf{x}), w_3(\mathbf{x})]^\top$ with $w_0(\mathbf{x}) = 1 - \sum_{i=1}^{3} w_i(\mathbf{x})$ are an affine mapping of the input $\mathbf{x}$ inside each tetrahedron $\mathcal{T}$ (see Appendix A.1). Hence, there exist a constant matrix $\mathbf{J}_\mathcal{T} \in \mathbb{R}^{3\times3}$ and vector $\mathbf{b}_\mathcal{T}$ such that

$$\mathbf{w}(\mathbf{x}) = \mathbf{J}_\mathcal{T}\mathbf{x} + \mathbf{b}_\mathcal{T}, \quad \text{and} \quad d\mathbf{w} = \mathbf{J}_\mathcal{T}\, d\mathbf{x}. \tag{9}$$

The Jacobian $\mathbf{J}_\mathcal{T}$ describes how barycentric weights vary w.r.t. spatial displacements, and therefore controls the direction-dependent sensitivity of feature updates during backpropagation. Anisotropy in $\mathbf{J}_\mathcal{T}$ leads to uneven effective learning rates along different directions. For the six-tetrahedra subdivision used in our encoder (Fig. 3), the average local metric $\mathbf{M} = \frac{1}{6}\sum_\mathcal{T} \mathbf{J}_\mathcal{T}^\top \mathbf{J}_\mathcal{T}$ is inherently anisotropic: directions orthogonal to the cube body diagonal are disproportionately amplified. To mitigate this geometric bias, we introduce a global linear preconditioner $\mathbf{A}^*$ and feed $\mathbf{x}' = \mathbf{A}^*\mathbf{x}$ into the encoder. This transformation makes it isotropic and reduces the spectral condition number from 16.39 to 5.05. For simplicity, we reuse $\mathbf{x}$ to denote the preconditioned coordinates hereafter. The explicit form and derivation of $\mathbf{A}^*$ are given in Appendix A.2.

4.3 Initial Skeleton Extraction from the Encoder

Naively enumerating all $\mathcal{C}_\mathbf{x}$ is prohibitive at multi-resolution scales, so we propose a tensorized algorithm that exploits shared plane normals with level-dependent offsets to extract the vertices and edges of $\mathcal{C}$ as an initial skeleton. Since cell

vertices are the intersections of tetrahedra across resolutions, we exploit the grid's regularity by representing $\mathcal{C}$ as a union of parallel-plane sets. The slopes of the tetrahedral subdivisions are constant across resolution levels, their normals are shared, while their offsets vary. Thus, multiple planes can be compactly represented as sharing a single normal but differing in offset. We denote the set of unique normals as $\mathcal{N} = \{\mathbf{n}_0, \mathbf{n}_1, \dots, \mathbf{n}_i, \dots\}$, and, for each normal $\mathbf{n}_i$, define the associated offsets across all resolution levels as $\mathbf{d}^{(i)} = [\, d_0^{(i)}, d_1^{(i)}, \dots]^\top$.

Vertices are extracted as intersections of three or more planes with linearly independent normals, and edges as intersections of two or more non-parallel planes. Note that the tetrahedral subdivision in Fig. 3 involves only six fixed plane normals, while their offsets vary across resolution levels. By grouping a small set of shared normals with their corresponding offsets, the extraction process can be efficiently implemented using parallel tensor operations. For the detailed algorithm, refer to Appendix A.4.

4.4 Encoding Regions and Boundaries

Sign vectors allow us to identify the linear region of a preconditioned input $\mathbf{x}$ within the ReLU network $\nu^{(M)}$ and to determine whether $\mathbf{x}$ lies on a decision boundary. However, the encoder τ further refines these regions according to the boundary of $\mathcal{C}_\mathbf{x}$. To correctly represent regions under the full tetrahedral network $f = \nu^{(M)} \circ \tau$, we therefore require a more specialized indicator that also encodes the position of $\mathbf{x}$ within the grid structure induced by τ.

Barycentric masks The barycentric coordinates of $\mathbf{x}$ in the tetrahedron $\mathcal{T}^{(\ell)}(\mathbf{x})$ at level ℓ are defined as:

$$\mathbf{w}^{(\ell)}(\mathbf{x}) = [w_0^{(\ell)}(\mathbf{x}), w_1^{(\ell)}(\mathbf{x}), w_2^{(\ell)}(\mathbf{x}), w_3^{(\ell)}(\mathbf{x})].$$

Then, we define the barycentric mask at level ℓ as:

$$\mathbf{m}^{(\ell)}(\mathbf{x}) = [m_0^{(\ell)}(\mathbf{x}),\, m_1^{(\ell)}(\mathbf{x}),\, m_2^{(\ell)}(\mathbf{x}),\, m_3^{(\ell)}(\mathbf{x})],$$

where

$$m_i^{(\ell)}(\mathbf{x}) = \begin{cases} 1, & w_i^{(\ell)}(\mathbf{x}) > \epsilon_b, \\ 0, & w_i^{(\ell)}(\mathbf{x}) \le \epsilon_b, \end{cases} \quad i \in \{0,1,2,3\}, \tag{10}$$

and $\epsilon_b > 0$ is a small threshold. A zero entry indicates that $\mathbf{x}$ lies on the boundary of $\mathcal{T}^{(\ell)}(\mathbf{x})$. Accordingly, a single zero indicates that $\mathbf{x}$ lies on a face of the tetrahedron, two zeros indicate that it lies on an edge, and three zeros indicate that it lies on a vertex. The *barycentric masks* of $\mathbf{x}$ across all levels are concatenated to form:

$$\mathbf{m}(\mathbf{x}) = \bigoplus_{\ell=0}^{L-1} \mathbf{m}^{(\ell)}(\mathbf{x}) \in \{0,1\}^{4L}. \tag{11}$$

Recall that $\mathcal{C}_\mathbf{x}$ is defined by the geometric intersection of the $\mathbf{x}$-containing tetrahedra $\mathcal{T}^{(\ell)}(\mathbf{x})$; thus the barycentric mask indicates on which boundary of the $\mathcal{C}_\mathbf{x}$ the point lies.

Region indicators To uniquely locate $\mathbf{x}$ within the $\mathcal{C}$, we introduce an additional spatial encoding, referred to as the *region indicator*. For a given level $\ell \in \{0, \dots, L-1\}$ with resolution N_ℓ, the *grid anchor* of a point $\mathbf{x}$ is defined as: $\mathbf{a}^{(\ell)}(\mathbf{x}) := \lfloor N_\ell \mathbf{x} \rfloor$, where $\lfloor \cdot \rfloor$ denotes the componentwise floor operator. This anchor corresponds to the cube's lexicographically smallest vertex (the "v_0" corner in Fig. 3). Each cube at level ℓ is subdivided into six congruent tetrahedra. The *tetra offset* $t^{(\ell)}(\mathbf{x}) \in \{0, 1, 2, 3, 4, 5\}$ indicates which of these tetrahedra anchored at $\mathbf{a}^{(\ell)}(\mathbf{x})$ contains $\mathbf{x}$, according to the subdivision scheme shown in Fig. 3c. We then define the *tetra index* at level ℓ as: $\mathbf{r}^{(\ell)}(\mathbf{x}) := [\, \mathbf{a}^{(\ell)}(\mathbf{x}),\ t^{(\ell)}(\mathbf{x}) \,] \in \mathbb{Z}^4$. The *region indicator* is obtained by concatenating all levelwise tetra indices:

$$\mathbf{r}(\mathbf{x}) := \bigoplus_{\ell=0}^{L-1} \mathbf{r}^{(\ell)}(\mathbf{x}) \in \mathbb{Z}^{4L}. \tag{12}$$

This region indicator encodes the intersection of the tetrahedra containing $\mathbf{x}$ across all levels, serving as a unique spatial index of the cell $\mathcal{C}_{\mathbf{x}}$ within the $\mathcal{C}$.

4.5 Identifying Neighboring Cells on the Grid

First, for the grid part, we use both the region indicator $\mathbf{r}(\mathbf{x}) \in \mathbb{Z}^{4L}$ and the barycentric mask $\mathbf{m}(\mathbf{x}) \in \{0, 1\}^{4L}$ to identify neighboring cells when the point $\mathbf{x}$ lies on the boundary of its current cell $\mathcal{C}_{\mathbf{x}}$. For each resolution level ℓ, we consider the corresponding $\mathbf{r}^{(\ell)}(\mathbf{x})$ and $\mathbf{m}^{(\ell)}(\mathbf{x})$.

If one or more entries of $\mathbf{m}^{(\ell)}(\mathbf{x})$ are zero, then $\mathbf{x}$ lies on the corresponding boundary of $\mathcal{T}^{(\ell)}(\mathbf{x})$. To enumerate all neighboring cells, we perturb the tetra index $\mathbf{r}^{(\ell)}(\mathbf{x})$ according to the zero pattern in $\mathbf{m}^{(\ell)}(\mathbf{x})$. For each level ℓ, we precompute a lookup table of neighbor configurations (see Appendix A.8). Each entry in this table consists of an anchor displacement $\Delta_i \in \mathbb{Z}^3$ and a tetra offset $t_i \in \{0, \dots, 5\}$, corresponding to face, edge, or vertex adjacency. The pair $(\Delta_0, t_0) = ([0, 0, 0], t^{(\ell)}(\mathbf{x}))$ represents $\mathcal{T}^{(\ell)}(\mathbf{x})$ itself, while the remaining entries specify neighboring tetrahedra. The tetra index of the i-th neighbor at level ℓ is then given by $\mathbf{r}_i^{(\ell)}(\mathbf{x}) = \big[\mathbf{a}^{(\ell)}(\mathbf{x}) + \Delta_i,\ t_i\big]$.

To obtain the complete set of neighboring polyhedral cells, the levelwise tetra indices $\mathbf{r}_i^{(\ell)}(\mathbf{x})$ are concatenated across all levels for every possible combination of neighbors. This expansion yields $N_{\text{grid}} = K_0 K_1 \dots K_{L-1}$ region indicators where K_ℓ denotes the number of neighboring tetrahedra including itself. The region indicator corresponding to the n-th neighboring cell for $\mathcal{C}_{\mathbf{x}}$ is:

$$\mathbf{r}_n(\mathbf{x}) = \mathbf{r}_p^{(0)}(\mathbf{x}) \oplus \mathbf{r}_q^{(1)}(\mathbf{x}) \oplus \cdots \oplus \mathbf{r}_r^{(L-1)}(\mathbf{x}) \tag{13}$$

where each $\mathbf{r}_j^{(\ell)}(\mathbf{x})$ denotes the tetra index of the j-th neighboring tetrahedron at level ℓ. The collection of all such indices defines the set of grid-based neighboring regions:

$$\mathcal{A}_{\text{grid}}(\mathbf{x}) = \{\, \mathbf{r}_0(\mathbf{x}),\, \mathbf{r}_1(\mathbf{x}),\, \cdots,\, \mathbf{r}_{N_{\text{grid}}-1}(\mathbf{x}) \,\}. \tag{14}$$

Second, for the ReLU MLP part, we follow the same procedure described for $\nu_k^{(m)}$ in Sec. 3.2. The sign-vector $\mathbf{s}_k^{(m)}(\mathbf{x})$ is perturbed at its zero entries,

producing all adjacent regions of ReLU MLP around the boundary on which $\mathbf{x}$ lies. The set of perturbed sign-vectors is denoted by:

$$\mathcal{A}_{\text{relu}}(\mathbf{x}) = \{\, \mathbf{s}^{(m)}_{(k,0)}(\mathbf{x}), \cdots, \mathbf{s}^{(m)}_{(k,N_{\text{relu}}-1)}(\mathbf{x}) \,\}, \tag{15}$$

where $\mathbf{s}^{(m)}_{(k,i)}(\mathbf{x})$ denotes the sign-vector of the i-th adjacent region, including the one containing $\mathbf{x}$ itself. Here, $N_{\text{relu}} = 2^{n_0}$, with n_0 denoting the number of zero entries in $\mathbf{s}^{(m)}_k(\mathbf{x})$.

Finally, we combine the grid-based and ReLU-based neighbor sets to obtain the complete set of parent regions adjacent to the query point $\mathbf{x}$: $\mathcal{A}(\mathbf{x}) = \{\, \mathbf{r}_i(\mathbf{x}) \oplus \mathbf{s}^{(m)}_{(k,j)}(\mathbf{x}) \,\}$ where $\mathbf{r}_i(\mathbf{x}) \in \mathcal{A}_{\text{grid}}(\mathbf{x})$ and $\mathbf{s}^{(m)}_{(k,j)}(\mathbf{x}) \in \mathcal{A}_{\text{relu}}(\mathbf{x})$. For two query points $\mathbf{x}_a$ and $\mathbf{x}_b$, comparing $\mathcal{A}(\mathbf{x}_a)$ and $\mathcal{A}(\mathbf{x}_b)$ provides a test for region equivalence: if they share a common element, two points lie within or on the boundary of the same region.

4.6 Face Extraction

After the edge subdivision (Sec. 3.2) using the sign vectors (Definition 2) and the region indicator (Sec. 4.4), we obtain the candidate vertex set $\mathcal{V}$ and the edge set $\mathcal{E}$. We then select the zero-level subsets from these as follows: $\mathcal{V}^* = \{\mathbf{x} \in \mathcal{V} \mid |f(\mathbf{x})| \leq \epsilon_f\}$, $\mathcal{E}^* = \{(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{E} \mid \mathbf{x}_a, \mathbf{x}_b \in \mathcal{V}^*\}$, where ϵ_f is a small threshold used to extract the zero-level set of f. The final triangle mesh is obtained through a straightforward procedure that connects adjacent vertices according to their local connectivity [16].

5 Experiments

Datasets We evaluate on three datasets: the Stanford 3D Scanning Repository with 5 meshes [7], ABC with 100 randomly selected meshes [18], and Thingi10K with 500 randomly selected closed meshes [45]. We follow the preprocessing of DeepSDF normalization [27]. For each ground-truth mesh, we sample 500K SDF query points, drawing the majority from a narrow band around the surface and the remainder uniformly within the bounding volume.

Settings We use a multi-resolution tetrahedral positional encoder with $L = 4$ levels and feature dimension $d = 2$ per level, followed by a ReLU MLP with three hidden layers of width 12 each. We consider three resolution settings: *Small* $(N_{\min}, N_{\max}) = (2, 32)$, *Medium* $(4, 64)$, and *Large* $(8, 128)$. We train networks for 10 epochs using an ℓ_1 SDF loss and an eikonal regularizer with weight $\lambda_{\text{eik}} = 5 \times 10^{-3}$. All experiments use a single NVIDIA V100 GPU; training a network for each SDF takes about 1 minute.

Baselines We consider both network architectures for SDF learning and meshing algorithms for isosurface extraction. For network architectures, we use grid-based positional encoders—HashGrid [26] and PermutoGrid [32]—each followed by a ReLU MLP with the same depth, width, and resolution settings. We also include a plain ReLU MLP configured as in Analytic Marching (AM) [21], with ten

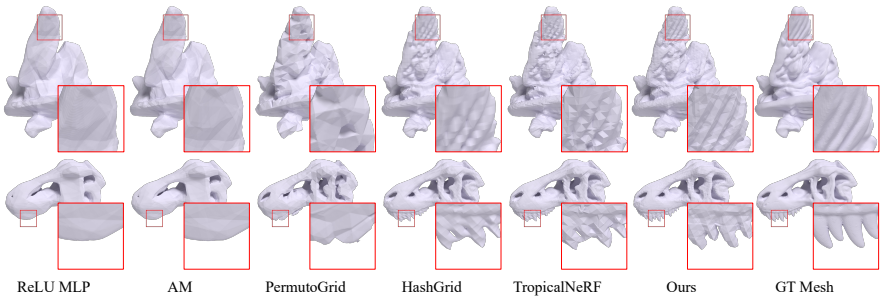


Fig. 4: Qualitative comparison on Thingi10K corresponding to Tab. 1.

hidden layers of width 90. Unless otherwise noted, meshes for these networks are extracted using Marching Cubes at 512^3 resolution. We apply the corresponding analytic methods: the AM extractor to the plain ReLU MLP and TropicalNeRF [16] to the HashGrid. In both cases, meshes are extracted from the same corresponding trained networks used for SDF evaluation. For sampling-based methods, we compare against the Marching Cubes (MC), Marching Tetrahedra (MT), and Dual Contouring (DC). We use publicly released implementations for HashGrid, PermutoGrid, AM, and TropicalNeRF.

Evaluation metrics We measure geometric accuracy with Chamfer Distance (CD), computed from 10^6 surface samples between the extracted mesh and the ground-truth (GT) mesh. We also assess self-consistency—agreement between the extracted mesh and the network's isosurface—using three additional metrics. For Surface-sampled SDF (SSDF) and Angular Difference (AD), we uniformly sample 10^6 points on the surface of the extracted mesh and query the network: SSDF is the mean absolute SDF value predicted by the network, while AD is the mean absolute angle between the mesh normals and the network-derived normals. Vertex-sampled SDF (VSDF) is SSDF computed at the extracted mesh's vertices. In all three metrics, zero indicates perfect self-consistency, *i.e.*, the extracted mesh exactly matches the network's zero-level set.

5.1 Ground-truth Accuracy

We report CD to ground-truth meshes across network baselines in Tab. 1 under the *Large* resolution setting. Our method attains the lowest CD on Thingi10K and ABC among all baselines and surpasses all analytic-extraction methods across datasets. Fig. 4 provides qualitative comparisons on Thingi10K for Tab. 1. A plain ReLU MLP shows limited ability to represent complex SDFs. Moreover, HashGrid may smooth high-curvature regions because of the implicit smoothing of trilinear interpolation, whereas our method better preserves sharp features. We also compare analytic isosurface extraction baselines under different resolution settings on the Stanford dataset in Tab. 2 and Fig. A1, reporting CD ($\times 10^{-6}$) against ground-truth meshes. As the resolution decreases, TropicalNeRF's accuracy degrades markedly, suggesting that its geometric heuristics

Table 1: CD ($\times 10^{-6}$) to ground-truth meshes on Thingi10K, ABC, Stanford at (*Large*) resolution setting.

Method	Stanford	ABC	Thingi10K
ReLU MLP	5480	4584	3779
AM	5475	4570	3775
PermutoGrid	3644	3104	2897
HashGrid	**1659**	1854	1763
TropicalNeRF	1737	1866	1809
Ours	1718	**1758**	**1722**

Table 2: CD ($\times 10^{-6}$) to ground-truth meshes on Stanford with analytic extraction baselines across resolutions.

R	Method	Bunny	Dragon	Happy	Arma.	Lucy	Avg.
–	AM	3628	6021	6839	4577	6309	5475
S	Tropical	4093	5674	7187	**3671**	6208	5367
	Ours	**2810**	**4160**	**4889**	3794	**5398**	**4210**
M	Tropical	2201	2606	2984	2506	3555	2770
	Ours	**2024**	**2483**	**2362**	**2300**	**3141**	**2462**
L	Tropical	1842	**1726**	1792	1611	1714	1737
	Ours	**1817**	1732	**1779**	**1581**	**1681**	**1718**

Table 3: GT mesh-sampled SDF MAE ($\times 10^{-3}$, $\downarrow$). We sample 10^6 points uniformly on the GT mesh surface, query the network, and report the mean absolute SDF value.

Method	Small			Medium			Large		
	Stanford	ABC	Thingi10K	Stanford	ABC	Thingi10K	Stanford	ABC	Thingi10K
PermutoGrid	7.62	4.54	5.36	4.34	2.66	3.05	2.24	1.53	1.50
HashGrid	3.24	2.19	2.67	**1.36**	1.21	1.12	**0.53**	**0.61**	0.50
Ours	**2.82**	**1.74**	**1.79**	1.39	**1.12**	**1.06**	0.62	**0.61**	**0.49**

struggle with the inherent nonlinearity of trilinear interpolation, whereas our method avoids these issues due to our CPWA structure and analytic extraction. The qualitative results in Fig. A1 show that—even under the *Small* setting—our method remains visually reasonable.

To decouple representation fidelity from extraction faithfulness, we report GT-mesh-sampled SDF mean absolute error (MAE) in Tab. 3, which evaluates the learned field independently of the extractor. Our GT-mesh SDF MAE is comparable to or better than grid-based encoder baselines across datasets and settings, indicating that our encoder design does not sacrifice SDF fitting fidelity while enabling exact isosurface recovery. Tab. 6 reports the quantitative effect of the input preconditioner $\mathbf{A}^*$ on Thingi10K, with CD ($\times 10^{-6}$) measured against the ground-truth mesh under the *Large* resolution setting, and Fig. 6 shows qualitative results on the Stanford Bunny. Both indicate improved SDF accuracy for our network with $\mathbf{A}^*$. Applying our encoder-derived $\mathbf{A}^*$ reduces CD for our method but worsens it for HashGrid, suggesting that $\mathbf{A}^*$ is tailored to our encoder rather than a generic preconditioner. Under matched capacity—same levels, per-level table size, and resolution—the total number of learnable features is identical to HashGrid. See Appendix A.6.2 for further analysis.

5.2 Self-consistency with the Network's Isosurface

We evaluate self-consistency in Tab. 4 using SSDF ($\times 10^{-6}$), VSDF ($\times 10^{-6}$), and AD (°). Analytic baselines are measured on their own trained networks with their corresponding extractors, whereas MC256, MC512, and MC1024 are applied to our trained network. Both AM and our method achieve both SSDF and VSDF on the order of 10^{-8}, and AD rounds to 0.00. These values are near machine precision, yielding $\sim 10^3\times$ stronger self-consistency than MC1024. By contrast,

Table 4: Self-consistency of different meshing methods on Thingi10K, ABC, and Stanford: SSDF and VSDF ($\times 10^{-6}$) and AD (°) between each extracted mesh and its SDF network. Marching Cubes at each resolution is applied to our trained network.

Method	Thingi10K			ABC			Stanford		
	SSDF ↓	VSDF ↓	AD ↓	SSDF ↓	VSDF ↓	AD ↓	SSDF ↓	VSDF ↓	AD ↓
MC256	217.0	98.0	5.44	218.1	101.8	5.70	289.3	128.3	8.91
MC512	65.2	27.0	3.10	66.7	29.2	3.23	100.1	40.2	5.81
MC1024	17.6	1.79	1.68	18.9	8.26	1.72	20.5	6.00	2.19
AM	0.020	0.020	0.00	0.021	0.022	0.00	0.025	0.024	0.00
TropicalNeRF	144.1	8.7	3.45	136.1	12.3	3.64	189.0	13.4	4.80
Ours	0.078	0.082	0.00	0.077	0.080	0.00	0.076	0.078	0.00

TropicalNeRF shows lower self-consistency, consistent with the limits of its geometric heuristics for handling trilinear interpolation. The small discrepancy between AM and our method stems from running our encoder in single precision and reflects machine-precision effects, including tie-breaking at tetrahedral grid boundaries and barycentric weight evaluation.

A broader comparison with widely used sampling-based extractors is given in Tab. 5 and Fig. 5. Tab. 5 reports CD ($\times 10^{-6}$) for MC, MT, and DC across multiple grid resolutions and across our network's different resolution settings with all meshes extracted from our trained network. For each method, we compare against a high-resolution pseudo ground-truth—MC1024 for MC, MT512 for MT, and DC1024 for DC—chosen to closely approximate the network's isosurface. In Tab. 5, our method achieves stronger consistency with fewer vertices across extractor resolutions and network settings, except for DC512 in the *Small* setting. This exception is consistent with DC's use of network normals, whereas MC and MT use only SDF values. Still, DC's slight edge comes at the cost of roughly 30× more vertices than ours. As shown in Fig. 5, sampling-based methods often need a large number of triangles to improve self-consistency, while discrete sampling itself introduces staircase artifacts. In contrast, ours achieves better self-consistency at a fixed, low triangle budget with 10×–20× fewer vertices, and avoids over-fragmentation and staircase artifacts. For runtime and memory, and for additional comparisons with adaptive sampling-based extractors and AM applied to larger ReLU MLPs, see Appendix A.5.

6 Conclusion

We introduced *TetraSDF*, an analytic isosurface extraction framework that preserves the CPWA structure of learned SDFs through a multi-resolution tetrahedral positional encoder with barycentric interpolation. This encoder generates an explicitly indexed polyhedral complex, enabling analytic extraction via region indicators, barycentric masks, and a grid-aware edge subdivision scheme that jointly tracks polyhedral cells and ReLU linear regions. We further introduced a closed-form input preconditioner derived from the encoder-induced

Table 5: CD ($\times 10^{-6}$) on the Stanford dataset for sampling-based extractors applied to our trained network under different resolution schedules, compared against pseudo ground-truth per method.

Method	Small		Medium		Large	
	$\lvert\mathcal{V}\rvert\downarrow$	CD↓	$\lvert\mathcal{V}\rvert\downarrow$	CD↓	$\lvert\mathcal{V}\rvert\downarrow$	CD↓
MC128	24,768	1734	24,735	1821	25,646	1871
MC256	101,337	1355	103,444	1380	105,227	1418
MC512	409,321	1290	418,464	1308	426,394	1323
Ours	29,557	**1286**	73,851	**1301**	280,929	**1316**
MT128	96,890	1677	88,928	1740	90,116	1806
MT256	396,516	1397	364,464	1363	370,935	1387
Ours	29,557	**1368**	73,851	**1314**	280,929	**1332**
DC128	41,461	2099	41,794	1524	42,048	1735
DC256	167,873	1351	169,164	1352	170,670	1437
DC512	676,567	**1329**	682,080	1334	688,562	1341
Ours	29,557	1348	73,851	**1331**	280,929	**1336**

Table 6: CD ($\times 10^{-6}$) with/without $\mathbf{A}^*$ on Thingi10K; $\Delta = \mathrm{CD}_{\mathrm{w/o}} - \mathrm{CD}_{\mathrm{w/}}$ (positive is better).

Method	w/o $\mathbf{A}^*$	w/ $\mathbf{A}^*$	$\Delta\uparrow$
HashGrid	1763	1993	−230
Ours	1856	1722	**134**

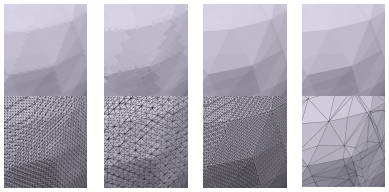


Fig. 5: Qualitative comparison under the *Small* setting. Sampling-based methods often over-fragment triangles with surface artifacts, while our method avoids both by construction.

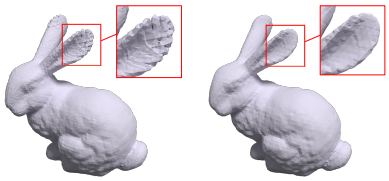


Fig. 6: Qualitative effect of the input preconditioner $\mathbf{A}^*$ on the Stanford Bunny. The preconditioner improves the network's SDF accuracy, especially in high-curvature regions (*e.g.*, the ears).

metric to reduce directional bias and improve training stability. Across multiple benchmarks, we match or surpass grid-based encoder baselines in SDF accuracy while producing highly self-consistent meshes faithful to the learned isosurfaces, enabled by an efficient, GPU-parallel tensorized formulation.

Future work includes integrating TetraSDF into end-to-end neural reconstruction pipelines. In such settings, our field-faithful isosurface extraction could provide a consistent mesh-based supervision signal via field–mesh agreement under compact triangle budgets, without resolution-driven over-fragmentation.

7 Acknowledgements

The NAVER Smart Machine Learning (NSML) platform [15] was used for the experiments. This work was supported by NAVER (Neural 3D Representation and Asset Generation for Web Rendering) and the National Research Foundation of Korea (RS-2026-25497603: Fundamentals of Online 4D Reconstruction with Semantic Viewpoint Control and Generative Priors for Proactive Content Consumption).

References

1. Berzins, A.: Polyhedral complex extraction from ReLU networks using edge subdivision. In: International Conference on Machine Learning (ICML). pp. 2234–2244. PMLR (2023)
2. Burstedde, C., Holke, J.: A tetrahedral space-filling curve for nonconforming adaptive meshes. SIAM Journal on Scientific Computing **38**(5), C471–C503 (2016)
3. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: TensoRF: Tensorial radiance fields. In: European Conference on Computer Vision (ECCV). pp. 333–350. Springer (2022)
4. Chen, Z., Tagliasacchi, A., Funkhouser, T., Zhang, H.: Neural dual contouring. ACM Trans. Graph. **41**(4), 104:1–104:13 (2022). https://doi.org/10.1145/3528223.3530108
5. Chen, Z., Zhang, H.: Neural marching cubes. ACM Trans. Graph. **40**(6), 1–15 (2021). https://doi.org/10.1145/3478513.3480518
6. Chng, S.F., Saratchandran, H., Lucey, S.: Preconditioners for the stochastic training of neural fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 27222–27232 (2025)
7. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH). pp. 303–312 (1996). https://doi.org/10.1145/237170.237269
8. Fridovich-Keil, S., Meanti, G., Warburg, F.R., Recht, B., Kanazawa, A.: K-Planes: Explicit radiance fields in space, time, and appearance. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 12479–12488 (2023)
9. Gao, J., Chen, W., Xiang, T., Tsang, C.F., Jacobson, A., McGuire, M., Fidler, S.: Learning deformable tetrahedral meshes for 3D reconstruction. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
10. Greenbaum, A.: Iterative Methods for Solving Linear Systems. SIAM (1997)
11. Grigsby, J.E., Lindsey, K.: On transversality of bent hyperplane arrangements and the topological expressiveness of ReLU neural networks. SIAM Journal on Applied Algebra and Geometry **6**(2), 216–242 (2022)
12. Gupta, V., Koren, T., Singer, Y.: Shampoo: Preconditioned stochastic tensor optimization. In: International Conference on Machine Learning (ICML) (2018)
13. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: Proceedings of the 36th International Conference on Machine Learning (ICML) (2019)
14. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: Proceedings of SIGGRAPH (2002)
15. Kim, H., Kim, M., Seo, D., Kim, J., Park, H., Park, S., Jo, H., Kim, K., Yang, Y., Kim, Y., et al.: NSML: Meet the MLaaS platform with a real-world case study. arXiv preprint arXiv:1810.09957 (2018)
16. Kim, J.H.: Polyhedral complex derivation from piecewise trilinear networks. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 37 (2024)
17. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
18. Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., Panozzo, D.: ABC: A big CAD model dataset for geometric deep learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

19. Kohlbrenner, M., Alexa, M.: Isosurface extraction for signed distance functions using power diagrams. Computer Graphics Forum **44**(2), e70037 (2025). https://doi.org/10.1111/cgf.70037
20. Kulhanek, J., Sattler, T.: Tetra-NeRF: Representing neural radiance fields using tetrahedra. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 18458–18469 (2023)
21. Lei, J., Jia, K.: Analytic marching: An analytic meshing solution from deep implicit surface networks. In: International Conference on Machine Learning (ICML) (2020)
22. Liu, L., Gu, J., Zaw Lin, K., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. Advances in Neural Information Processing Systems (NeurIPS) **33**, 15651–15663 (2020)
23. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics **21**(4), 163–169 (1987)
24. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3D reconstruction in function space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4460–4470 (2019)
25. Montúfar, G.F., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: Advances in Neural Information Processing Systems (NeurIPS) (2014)
26. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. **41**(4), 102:1–102:15 (2022). https://doi.org/10.1145/3528223.3530127
27. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 165–174 (2019)
28. Park, K., Henzler, P., Mildenhall, B., Barron, J.T., Martin-Brualla, R.: CamP: Camera preconditioning for neural radiance fields. ACM Trans. Graph. **42**(6) (2023). https://doi.org/10.1145/3618321
29. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Sohl-Dickstein, J.: On the expressive power of deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML) (2017)
30. Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., Courville, A.: On the spectral bias of neural networks. In: International Conference on Machine Learning (ICML). pp. 5301–5310. PMLR (2019)
31. Ren, D., Shi, H., Zheng, J., Cai, J.: McGrids: Monte Carlo-driven adaptive grids for iso-surface extraction. In: European Conference on Computer Vision (ECCV). pp. 127–144. Springer (2024). https://doi.org/10.1007/978-3-031-72998-0_8
32. Rosu, R.A., Behnke, S.: PermutoSDF: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2023)
33. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM (2003)
34. Sellán, S., Batty, C., Stein, O.: Reach for the spheres: Tangency-aware surface reconstruction of SDFs. In: SIGGRAPH Asia 2023 Conference Papers (2023). https://doi.org/10.1145/3610548.3618196
35. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: Proceedings of the 35th International Conference on Machine Learning (ICML) (2018)

36. Shen, T., Gao, J., Yin, K., Liu, M.Y., Fidler, S.: Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. In: Advances in Neural Information Processing Systems (NeurIPS) (2021)
37. Shen, T., Munkberg, J., Hasselgren, J., Yin, K., Wang, Z., Chen, W., Gojcic, Z., Fidler, S., Sharp, N., Gao, J.: Flexible isosurface extraction for gradient-based mesh optimization. ACM Trans. Graph. **42**(4) (2023). https://doi.org/10.1145/3592430
38. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in Neural Information Processing Systems (NeurIPS) **33**, 7462–7473 (2020)
39. Stippel, C., Mujkanovic, F., Leimkühler, T., Hermosilla, P.: Marching neurons: Accurate surface extraction for neural implicit shapes. ACM Trans. Graph. **44**(6), 222:1–222:12 (2025). https://doi.org/10.1145/3763328
40. Tancik, M., Srinivasan, P.P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J.T., Ng, R.: Fourier features let networks learn high frequency functions in low dimensional domains. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
41. Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., Gross, M.H.: Optimized spatial hashing for collision detection of deformable objects. In: Vision, Modeling, and Visualization (VMV). vol. 3, pp. 47–54 (2003)
42. Tieleman, T., Hinton, G.: Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude (2012)
43. Treece, G.M., Prager, R.W., Gee, A.H.: Regularised marching tetrahedra: Improved iso-surface extraction. Computers & Graphics **23**(4), 583–598 (1999)
44. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision (ECCV) (2014)
45. Zhou, Q., Jacobson, A.: Thingi10K: A dataset of 10,000 3D-printing models. arXiv preprint arXiv:1605.04797 (2016)