TetraSDF: Precise Mesh Extraction with Multi-resolution Tetrahedral Grid

Seonghun Oh*
Yonsei University
rendell@yonsei.ac.kr

Youngjung Uh
Yonsei University
yj.uh@yonsei.ac.kr

Jin-Hwa Kim†
NAVER AI Lab & SNU AIIS
j1nhwa.kim@navercorp.com

Abstract

Extracting meshes that exactly match the zero-level set of neural signed distance functions (SDFs) remains challenging. Sampling-based methods introduce discretization error, while continuous piecewise affine (CPWA) analytic approaches apply only to plain ReLU MLPs. We present TetraSDF, a precise analytic meshing framework for SDFs represented by a ReLU MLP composed with a multi-resolution tetrahedral positional encoder. The encoder's barycentric interpolation preserves global CPWA structure, enabling us to track ReLU linear regions within an encoder-induced polyhedral complex. A fixed analytic input preconditioner derived from the encoder's metric further reduces directional bias and stabilizes training. Across multiple benchmarks, TetraSDF matches or surpasses existing grid-based encoders in SDF reconstruction accuracy, and its analytic extractor produces highly self-consistent meshes that remain faithful to the learned isosurfaces, all with practical runtime and memory efficiency.

1. Introduction

Extracting surface meshes from learned SDFs has long been a fundamental problem in neural implicit surface representation. Classical methods such as Marching Cubes [23], Marching Tetrahedra [39], and Dual Contouring [15] discretize the implicit field, inevitably losing geometric fidelity due to their resolution-dependent sampling. Analytic Marching [21] later addressed this issue by interpreting ReLU-based MLPs as CPWA functions [14, 25, 29, 33], allowing exact mesh extraction from the true 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 meshes 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, 36], making it difficult to learn complex SDFs. TropicalNeRF [17] extends this by showing that a ReLU MLP combined with a grid-based positional encoder can be interpreted within a piecewise trilinear framework, providing a theoretical foundation for analytic extraction. For grid-based encoders using trilinear interpolation [4, 26], the analysis assumes that the eikonal constraint holds within each trilinear cell; Under this, the zero-level set forms a plane within each cell, enabling analytic extraction despite the non-affine interpolation. In practice, however, enforcing this condition uniformly is difficult, which is only approximately corrected by the diagonal-plane intersection heuristic.

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 surface 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 mesh-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 mesh 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

*Work done during an internship at NAVER AI Lab.
†Corresponding author.

encoder's grid geometry, improving optimization conditioning during training. The overall procedure is formulated in terms of parallel tensor operations, effectively leveraging the parallelism of modern GPUs.

In summary, our contributions are as follows:

- We introduce an analytic mesh extraction that exactly recovers the zero-level set of an SDF using a multi-resolution tetrahedral encoder, yielding a global CPWA.
- We employ a fixed global input preconditioner for the multi-resolution tetrahedral encoder to further improve the accuracy of the SDFs learned by our model.
- Extensive experiments demonstrate the superior precision of our method compared with widely used meshing algorithms across multiple datasets and positional encoders.

2. Related work

2.1. Positional encoding methods

Learning high-frequency geometric details in neural implicit functions is often hindered by spectral bias [30, 36]. To mitigate this effect, positional encoding schemes have been proposed to map input coordinates into higher-dimensional feature spaces with rich frequency content [35, 36] effectively 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 [4, 9, 22, 26]. Among grid-based approaches, Tetra-NeRF [20] employs an adaptive tetrahedral representation and PermutoSDF [31] 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.

2.2. 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 [15, 23, 39] convert these fields into meshes by sampling on a finite-resolution grids. However, the resulting surfaces suffer from discretization error, and reducing this error requires dense sampling, which rapidly increases mesh complexity. To improve fidelity or enable learning-based extraction, several works integrate neural networks with isosurface extraction or differentiable mesh parameterizations [5, 6, 10, 34]. 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] or piecewise trilinear formulations [17] to more directly characterize the network's zero-level sets, avoiding reliance on sampling. However, 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 in practice.

2.3. Preconditioning

Preconditioning is a classical technique from numerical linear algebra [11, 32] 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 [13, 18, 38]. ZCA whitening [40] 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 [7, 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.

3. Preliminary

3.1. Definitions

Definition (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:

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

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

Remark (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 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 (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:

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

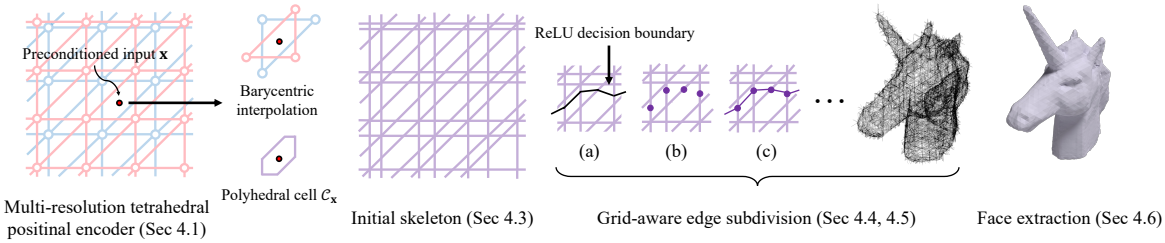


Figure 1. Overview of *TetraSDF*. A preconditioned input $\mathbf{x}$ is mapped by the multi-resolution tetrahedral positional encoder to barycentrically interpolated features within its containing polyhedral cell $\mathcal{C}_\mathbf{x}$ (Secs. 4.1 and 4.2). These cells form the encoder-induced polyhedral 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.

where each 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 $\mathcal{H}_k^{(m)} = \{\mathbf{x} \mid \nu_k^{(m)}(\mathbf{x}) = 0\}$. 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, 12] in the input space. Starting from an initial grid 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. 1c). To handle this consistently without explicitly enumerating all linear regions, we employ the *sign vectors* [1] defined in Sec. 3.1. A candidate edge formed by two previously computed intersection 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.

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 to generate and track adjacency across the tetrahedral network structure efficiently.

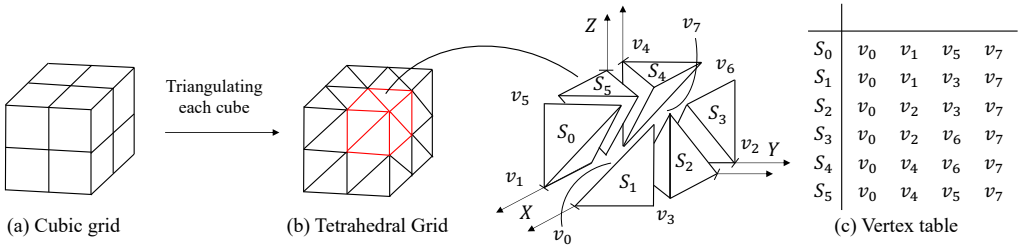
4. Method

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 edge subdivision, and 3) reconstructing the surface.

4.1. Multi-resolution tetrahedral positional encoder

Grid-based positional encoders mitigate spectral bias for ReLU MLPs, but their trilinear interpolation breaks the CPWA structure and hinders analytic extraction. Our multi-resolution tetrahedral encoder replaces trilinear with barycentric interpolation, preserving high-frequency modeling while keeping the CPWA structure.

Definition (multi-resolution tetrahedral grid)**.** Let $L \in \mathbb{N}$ be the number of resolution levels, and $N_{\min}$ and $N_{\max}$ be



S_0	v_0	v_1	v_5	v_7
S_1	v_0	v_1	v_3	v_7
S_2	v_0	v_2	v_3	v_7
S_3	v_0	v_2	v_6	v_7
S_4	v_0	v_4	v_6	v_7
S_5	v_0	v_4	v_5	v_7

(a) Cubic grid (b) Tetrahedral Grid (c) Vertex table

Figure 2. Subdivision of a cube cell into six congruent tetrahedra.

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 [3] in Fig. 2.

Definition (Tetrahedral positional encoder)**.** For a query point $\mathbf{x} \in \mathcal{S}$, let $\mathcal{T}^{(\ell)}(\mathbf{x})$ be any[1] 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 [37]. The entry of index i corresponds to the learnable feature vector $H_i^{(\ell)} \in \mathbb{R}^d$. The barycentric weight[2] 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 (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:

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

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

[2]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.

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 $\mathcal{C} := \bigcup_{\mathbf{x}\in\mathcal{S}} \mathcal{C}_\mathbf{x}$. 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 Sec. 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. 2), 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 Sec. A.2.

4.3. Initial skeleton extraction from the encoder

We begin by extracting the vertices and edges of $\mathcal{C}$ as our 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 com-

pactly represented as sharing a single normal but differing in offset. We denote the set of unique normals as $\mathcal{N} = \{\mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_i, \ldots\}$, and, for each normal $\mathbf{n}_i$, define the associated offsets across all resolution levels as $\mathbf{d}^{(i)} = [\, d_0^{(i)},\, d_1^{(i)},\, \ldots\,]^\top$.

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. 2 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 is 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 $\mathbf{x}$ is lying 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 $\ell \in \{0, \ldots, L-1\}$ 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. 2). 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. 2c. 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-offset $\mathbf{r}^{(\ell)}(\mathbf{x})$ 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.5). Each entry in this table consists of an anchor displacement $\Delta_i \in \mathbb{Z}^3$ and a tetra index $t_i \in \{0, \ldots, 5\}$, 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-offset 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 offsets $\mathbf{r}_i^{(\ell)}(\mathbf{x})$ are concatenated across all levels for every possible combination of neighbors. This expansion yields $N_{\text{grid}} = K_0 K_1 \ldots K_{L-1}$ region indicators where K_ℓ denotes the number of neighboring tetrahedra including itself. The region indicator corresponding to the k-th neighboring cell for $\mathcal{C}_\mathbf{x}$ is expressed as:

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

where each $\mathbf{r}_j^{(\ell)}(\mathbf{x})$ denotes the tetra-offset of the j-th neighboring tetrahedron at level ℓ. The collection of all such offsets 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 $\nu_m^{(k)}$ in Sec. 3.2. The sign-vector

$\mathbf{s}_m^{(k)}(\mathbf{x})$ 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:

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

where $\mathbf{s}^{(k)}_{(m,i)}(\mathbf{x})$ 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 $\mathbf{s}^{(k)}_m(\mathbf{x})$.

Finally, we combine the grid-based and ReLU-based neighbor sets to obtain the complete set of parent regions adjacent to the query point $\mathbf{x}$: $\mathcal{A}(\mathbf{x}) = \{\, \mathbf{r}_i(\mathbf{x}) \oplus \mathbf{s}^{(k)}_{(m,j)}(\mathbf{x}) \,\}$ where $\mathbf{r}_i(\mathbf{x}) \in \mathcal{A}_{\text{grid}}(\mathbf{x})$ and $\mathbf{s}^{(k)}_{(m,j)}(\mathbf{x}) \in \mathcal{A}_{\text{relu}}(\mathbf{x})$. 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

Collecting After the edge subdivision (Sec. 3.2) using the sign vectors (Sec. 3.1) 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 [17].

5. Experiments

Datasets We evaluate on three datasets: the Stanford 3D Scanning Repository [8], ABC [19], and Thingi10K [41]. From ABC we randomly select 100 meshes, and from Thingi10K, 500 closed meshes. We follow the preprocessing of DeepSDF normalization [27]. For each ground-truth mesh, we sample 500 K 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 $\lambda_{\text{eik}} = 5\times10^{-3}$. 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 [31]—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 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 [17] 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).

Evaluation metrics We measure geometric accuracy with Chamfer Distance (CD), computed from 10^6 surface samples between the extracted mesh and the ground-truth 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. Comprehensive results

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.

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

Table 1. CD ($\times10^{-6}$) to ground-truth meshes on Thingi10K, ABC, and Stanford under the *Large* setting.

Fig. 3 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 low-pass behavior of trilinear interpolation, whereas our method better preserves sharp features.

Tab. 2 reports the quantitative effect of the input preconditioner $\mathbf{A}^*$ on Thingi10K, with CD ($\times10^{-6}$) measured against the ground-truth mesh under the *Large* resolution

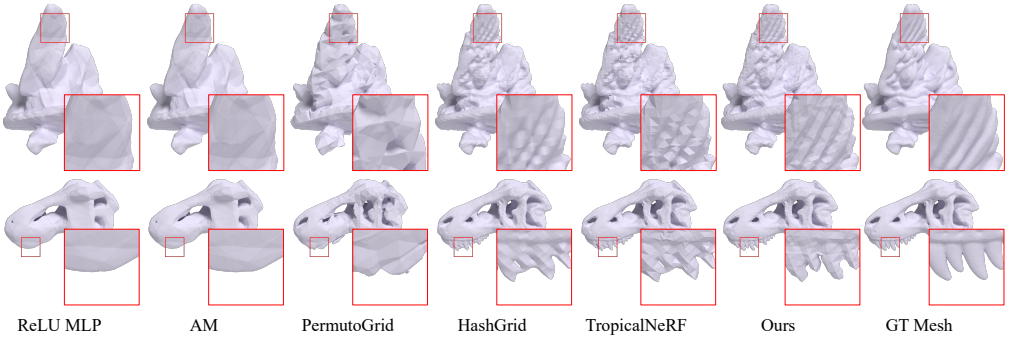
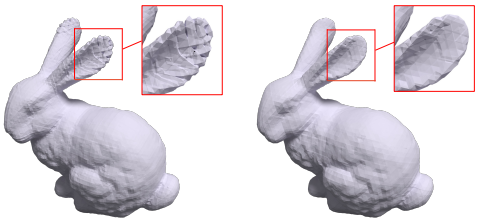


Figure 3. Qualitative comparison on Thingi10K corresponding to Tab. 1.

setting, and Fig. 4 shows qualitative results on the Stanford Bunny. Both indicate improved SDF accuracy for our network with $\mathbf{A}^*$. Applying $\mathbf{A}^*$ reduces CD for our method but worsens it for HashGrid, suggesting that $\mathbf{A}^*$ is tailored to our encoder. Under matched capacity—same levels, per-level table size, and resolution—the total number of learnable features is identical to HashGrid.

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

Table 2. CD ($\times 10^{-6}$) with and without the input preconditioner $\mathbf{A}^*$ on Thingi10K; $\Delta = \mathrm{CD}_{\mathrm{w/o}} - \mathrm{CD}_{\mathrm{w/}}$, and positive Δ is better.



(a) Without preconditioner (b) With preconditioner

Figure 4. 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).

We also compare analytic mesh-extraction baselines under different resolution settings on the Stanford dataset in Tab. 3 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 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.

We report runtime on the Stanford dataset across resolution settings. Tab. 5 reports the runtime for initial skeleton extraction and the end-to-end total runtime, which includes skeleton extraction. It also lists the initial-skeleton sizes $|\mathcal{V}|$ and $|\mathcal{E}|$, and the peak GPU memory. The results show that our method extracts the large vertex and edge sets of the initial skeleton—encoder-induced polyhedral complex—efficiently while keeping the total runtime reasonable. When the encoder's levels and resolutions are fixed, the initial skeleton can be cached and reused.

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. CD ($\times 10^{-6}$) to ground-truth meshes on the Stanford dataset with analytic extraction baselines.

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, 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. 6 and Fig. 5. Tab. 6 reports CD ($\times 10^{-6}$) for MC, MT, and DC across multiple grid resolutions and across our network's different resolution set-

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

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

R	Initial skeleton extraction			Total (s)	Memory (GB)
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	Time (s)		
Small	4.4×10^5	1.6×10^6	0.24	1.83	0.99
Medium	4.4×10^6	1.6×10^7	0.33	1.42	1.53
Large	3.5×10^7	1.3×10^8	1.99	4.56	8.01

Table 5. Runtime decomposed into initial skeleton extraction and total runtime, with peak GPU memory, for each resolution setting R on the Stanford dataset. When the encoder settings are fixed, the initial skeleton can be cached and reused.

Method	Small		Medium		Large	
	$\lvert\mathcal{V}\rvert$ ↓	CD↓	$\lvert\mathcal{V}\rvert$ ↓	CD↓	$\lvert\mathcal{V}\rvert$ ↓	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
MC256	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	82,050	**1331**	280,929	**1336**

Table 6. 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.

tings 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. 6, 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 $30\times$ 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. However, our meshes—while using $10\times$–$20\times$ fewer vertices—adhere more closely to the isosurface and are free from those artifacts.

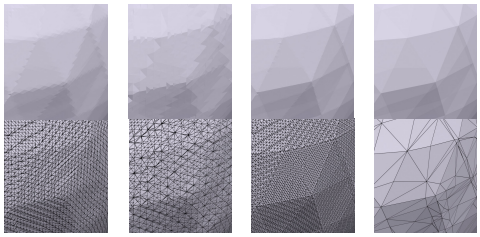


Figure 5. Qualitative comparison under the *Small* setting. Sampling-based methods often over-fragment triangles with surface artifacts, while our method avoids both by construction.

6. Conclusion

We introduced *TetraSDF*, an analytic mesh 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 metric to reduce directional bias and improve training stability. In multiple benchmarks, we surpass other grid-based encoders, while producing self-consistent meshes that remain faithful to the learned isosurfaces, all with practical runtime and memory efficiency.

Future work includes extending the polyhedral complex and region indexing to more general simplex or adaptive grids, incorporating improved SDF training objectives to further enhance mesh quality.

7. Acknowledgements

The NAVER Smart Machine Learning (NSML) platform [16] was used for the experiments. This work was supported by NAVER (No. 2025-11-1700, Neural 3D Representation and Asset Generation for Web Rendering), and partly supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00072, Development of Audio/Video Coding and Light Field Media Fundamental Technologies for Ultra Realistic Tera-media).

References

[1] Arturs Berzins. Polyhedral complex extraction from relu networks using edge subdivision. In *International Conference on Machine Learning*, pages 2234–2244. PMLR, 2023. 1, 2, 3

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. 14

[3] Carsten Burstedde and Johannes Holke. A tetrahedral space-filling curve for nonconforming adaptive meshes. *SIAM Journal on Scientific Computing*, 38(5):C471–C503, 2016. 4

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *European conference on computer vision*, pages 333–350. Springer, 2022. 1, 2

[5] Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 40(6), 2021. 2

[6] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Transactions on Graphics*, 41(4), 2022. 2

[7] Shin-Fang Chng, Hemanth Saratchandran, and Simon Lucey. Preconditioners for the stochastic training of neural fields. *arXiv preprint arXiv:2402.08784*, 2024. 2

[8] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312, 1996. 6

[9] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-Planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023. 2

[10] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

[11] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, 1997. 2

[12] J Elisenda Grigsby and Kathryn Lindsey. 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. 3

[13] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning (ICML)*, 2018. 2

[14] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019. 1

[15] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of SIGGRAPH*, 2002. 1, 2

[16] Hanjoo Kim, Minkyu Kim, Dongjoo Seo, Jinwoong Kim, Heungseok Park, Soeun Park, Hyunwoo Jo, KyungHyun Kim, Youngil Yang, Youngkwan Kim, Nako Sung, and Jung-Woo Ha. NSML: Meet the MLAAS platform with a real-world case study. *arXiv preprint arXiv:1810.09957*, 2018. 9

[17] Jin-Hwa Kim. Polyhedral complex derivation from piecewise trilinear networks. In *Advances in Neural Information Processing Systems 37*, 2024. 1, 2, 6

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 2

[19] Sebastian Koch, Zhongshi Jiang, Daniele Panozzo, Marc Alexa, Alexey Artemov, Evgeny Burnaev, Albert Matveev, Francis Williams, and Denis Zorin. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 6

[20] Jonas Kulhanek and Torsten Sattler. Tetra-NeRF: Representing neural radiance fields using tetrahedra. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 18458–18469, 2023. 2

[21] Jiabao Lei and Kui Jia. Analytic marching: An analytic meshing solution from deep implicit surface networks. arXiv preprint arXiv:2002.06597, 2020. 1, 2, 6

[22] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 2

[23] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987. 1, 2

[24] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 2

[25] Guido F. Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 1

[26] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Habermann. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41(4), 2022. 1, 2, 4, 6

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. 2, 6

[28] Keunhong Park, Philipp Henzler, Ben Mildenhall, Jonathan T. Barron, and Ricardo Martin-Brualla. Camp: Camera preconditioning for neural radiance fields. In *SIGGRAPH Asia*, 2023. Project page: https://camp-nerf.github.io/. 2

[29] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017. 1

[30] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 1, 2

[31] Radu Alexandru Rosu and Sven Behnke. Permutosdf: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 6

[32] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003. 2

[33] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018. 1

[34] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2

[35] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 2

[36] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2

[37] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, pages 47–54, 2003. 4

[38] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude, 2012. 2

[39] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: Improved iso-surface extraction. *Computers & Graphics*, 23(4):583–598, 1999. 1, 2

[40] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014. Includes discussion of ZCA whitening for input normalization. 2

[41] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016. 6

TetraSDF: Precise Mesh Extraction with Multi-resolution Tetrahedral Grid

Supplementary Material

A.1. Theoretical proofs

In this section, we prove that barycentric interpolation within a tetrahedron is an affine transformation of the input, which we use to show that tetrahedral networks are piecewise affine in Sec. 3.1.

Theorem 1 (Affine property of barycentric interpolation within a tetrahedron). Let $\mathbf{g} : \mathbb{R}^3 \to \mathbb{R}^d$ be a function that encodes a point $\mathbf{x}$ into a feature vector $\mathbf{g}(\mathbf{x})$. For any point $\mathbf{x} \in \mathbb{R}^3$, $\mathbf{g}(\mathbf{x})$ is obtained via barycentric interpolation using the feature vectors that represent the four vertices of a tetrahedron:

$$\mathbf{g}(\mathbf{v}_0),\ \mathbf{g}(\mathbf{v}_1),\ \mathbf{g}(\mathbf{v}_2),\ \text{and}\ \mathbf{g}(\mathbf{v}_3).$$

Then, $\mathbf{g}(\mathbf{x})$ is an affine transformation of the input $\mathbf{x}$, which can be expressed as:

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}, \quad \mathbf{A} \in \mathbb{R}^{d\times 3}, \quad \mathbf{b} \in \mathbb{R}^d, \tag{16}$$

where $\mathbf{A}$ is a linear transformation matrix, and $\mathbf{b}$ is a translation vector.

Proof. Let $\boldsymbol{w} = (w_0, w_1, w_2, w_3)$ be the barycentric coordinates of a point $\mathbf{x}$ for the tetrahedron defined by its coordinates of vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$. Each barycentric coefficient $w_i \in \mathbb{R}$ satisfies the constraint: $w_0 + w_1 + w_2 + w_3 = 1$. First, we express the function $\mathbf{g}$ in terms of the barycentric interpolation:

$$\mathbf{g}(\mathbf{x}) = w_0\mathbf{g}(\mathbf{v}_0) + w_1\mathbf{g}(\mathbf{v}_1) + w_2\mathbf{g}(\mathbf{v}_2) + w_3\mathbf{g}(\mathbf{v}_3). \tag{17}$$

Using the barycentric constraint:

$$w_0 = 1 - (w_1 + w_2 + w_3), \tag{18}$$

we substitute this into the equation:

$$\mathbf{g}(\mathbf{x}) = (1 - w_1 - w_2 - w_3)\mathbf{g}(\mathbf{v}_0) + w_1\mathbf{g}(\mathbf{v}_1) + w_2\mathbf{g}(\mathbf{v}_2) + w_3\mathbf{g}(\mathbf{v}_3). \tag{19}$$

Rearranging the equation, we obtain:

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{v}_0) + (\mathbf{g}(\mathbf{v}_1) - \mathbf{g}(\mathbf{v}_0))w_1 + (\mathbf{g}(\mathbf{v}_2) - \mathbf{g}(\mathbf{v}_0))w_2 + (\mathbf{g}(\mathbf{v}_3) - \mathbf{g}(\mathbf{v}_0))w_3. \tag{20}$$

Defining the matrix $\mathbf{G}$ as:

$$\mathbf{G} = \left[\mathbf{g}(\mathbf{v}_1) - \mathbf{g}(\mathbf{v}_0),\ \mathbf{g}(\mathbf{v}_2) - \mathbf{g}(\mathbf{v}_0),\ \mathbf{g}(\mathbf{v}_3) - \mathbf{g}(\mathbf{v}_0)\right] \quad \in \mathbb{R}^{d\times 3}, \tag{21}$$

we can write:

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{v}_0) + \mathbf{G}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}. \tag{22}$$

Similarly, the point $\mathbf{x}$ itself can be expressed in barycentric form as

$$\mathbf{x} = w_0\mathbf{v}_0 + w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + w_3\mathbf{v}_3, \tag{23}$$

which, using the same elimination of w_0 and grouping terms relative to $\mathbf{v}_0$, yields

$$\mathbf{x} = \mathbf{v}_0 + \mathbf{C}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}, \tag{24}$$

where $\mathbf{C} = [\mathbf{v}_1 - \mathbf{v}_0,\ \mathbf{v}_2 - \mathbf{v}_0,\ \mathbf{v}_3 - \mathbf{v}_0] \in \mathbb{R}^{3\times 3}$.

If the vectors $\{\mathbf{v}_1 - \mathbf{v}_0, \mathbf{v}_2 - \mathbf{v}_0, \mathbf{v}_3 - \mathbf{v}_0\}$ is linearly independent, *i.e.*, the four vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, and $\mathbf{v}_3$ do not lie on the same plane and no three vertices are collinear, then the matrix $\mathbf{C}$ is invertible.
Thus, the barycentric coordinates of $\mathbf{x}$ can be computed as:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \mathbf{C}^{-1}(\mathbf{x} - \mathbf{v}_0). \tag{25}$$

Substituting Eq. (25) into Eq. (22):

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{v}_0) + \mathbf{G}\mathbf{C}^{-1}(\mathbf{x} - \mathbf{v}_0). \tag{26}$$

Rewriting this:

$$\mathbf{g}(\mathbf{x}) = \mathbf{G}\mathbf{C}^{-1}\mathbf{x} + (\mathbf{g}(\mathbf{v}_0) - \mathbf{G}\mathbf{C}^{-1}\mathbf{v}_0). \tag{27}$$

Thus, $\mathbf{g}(\mathbf{x})$ is an affine transformation with:

$$\mathbf{A} = \mathbf{G}\mathbf{C}^{-1}, \quad \mathbf{b} = \mathbf{g}(\mathbf{v}_0) - \mathbf{G}\mathbf{C}^{-1}\mathbf{v}_0. \tag{28}$$

with corresponding $\mathbf{A}$ and $\mathbf{b}$ in Eq. (16), which completes the proof. □

Lemma 1 (Affine property of levelwise feature concatenation). In a region where the feature vector selection is fixed, the levelwise concatenation of the barycentrically interpolated feature vector is also an affine transformation of the input $\mathbf{x}$.

Proof. For each resolution level $\ell \in \{0, \ldots, L-1\}$, let

$$\mathbf{g}^{(\ell)}(\mathbf{x}) = \mathbf{A}^{(\ell)}\mathbf{x} + \mathbf{b}^{(\ell)} \in \mathbb{R}^d \tag{29}$$

denote the barycentric feature interpolation within the tetrahedron containing $\mathbf{x}$ at level ℓ by Eq. (27). Define the concatenated feature vector $\mathbf{z}(\mathbf{x})$ as:

$$\mathbf{z}(\mathbf{x}) = \bigoplus_{\ell=0}^{L-1} \mathbf{g}^{(\ell)}(\mathbf{x}) = \left[\mathbf{g}^{(0)}(\mathbf{x})^\top, \ \ldots, \ \mathbf{g}^{(L-1)}(\mathbf{x})^\top\right]^\top \in \mathbb{R}^{dL}, \tag{30}$$

where $\oplus$ denotes concatenation over all resolution levels. In a region where the feature selection is fixed across all levels, we have

$$\mathbf{z}(\mathbf{x}) = \begin{bmatrix} \mathbf{A}^{(0)} \\ \vdots \\ \mathbf{A}^{(L-1)} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{b}^{(0)} \\ \vdots \\ \mathbf{b}^{(L-1)} \end{bmatrix} = \mathbf{A}'\mathbf{x} + \mathbf{b}' \tag{31}$$

for suitable $\mathbf{A}'$ and $\mathbf{b}'$. Thus, $\mathbf{z}(\mathbf{x})$ is an affine transformation of the input $\mathbf{x}$, which completes the proof. □

A.2. Input preconditioning

Our global input preconditioner mitigates the geometric anisotropy introduced by the tetrahedral encoder by mapping $\mathbf{x}$ to $\mathbf{A}^*\mathbf{x}$. Since the learnable features are interpolated using barycentric coefficients that are derived from an affine function of the input $\mathbf{x}$ inside each tetrahedron from Eq. (25), the mapping from input to barycentric coordinates should not favor specific directions. Although the encoder is multi-resolution, each resolution level is a scaled and translated copy of the same tetrahedral tiling, and the resulting features from each level are only concatenated (*ref.* Lemma 1). Therefore, it suffices to analyze the encoder-induced metric on a single canonical unit cube and its six tetrahedra; the same preconditioner applies globally to all cells and levels.

Encoder-induced local metric Consider a tetrahedron $\mathcal{T}$ from the subdivision of the cube shown in Figure 2; for convenience, we treat it as a unit cube with ordered vertices $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$. Define

$$\mathbf{C}_\mathcal{T} = \begin{bmatrix} \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 & \mathbf{v}_3 - \mathbf{v}_0 \end{bmatrix} \in \mathbb{R}^{3\times 3}, \quad \mathbf{w} = [w_1, w_2, w_3]^\top. \tag{32}$$

Inside $\mathcal{T}$, Eq. (25) yields $\mathbf{x} = \mathbf{v}_0 + \mathbf{C}_\mathcal{T}\mathbf{w}$, and differentiation gives $d\mathbf{w} = \mathbf{C}_\mathcal{T}^{-1}d\mathbf{x}$. Measuring the squared norm of $d\mathbf{w}$ gives:

$$\|d\mathbf{w}\|^2 = \left\|\mathbf{C}_\mathcal{T}^{-1}d\mathbf{x}\right\|^2 = d\mathbf{x}^\top \mathbf{C}_\mathcal{T}^{-\top}\mathbf{C}_\mathcal{T}^{-1}d\mathbf{x}, \quad \mathbf{M}_\mathcal{T} := \mathbf{C}_\mathcal{T}^{-\top}\mathbf{C}_\mathcal{T}^{-1}. \tag{33}$$

This encoder-induced local metric quantifies how sensitively barycentric coordinates respond to infinitesimal displacements in $\mathbf{x}$. Our tetrahedral encoder is obtained by regularly tiling space with scaled and translated copies of a canonical unit cube subdivided into six congruent tetrahedra (see Fig. 2). Let $\mathcal{T} \in \{S_0, \dots, S_5\}$ denote these tetrahedra and let $\mathbf{M}_\mathcal{T}$ be their local metrics from Eq. (33). Their simple average represents the average encoder-induced anisotropy within one canonical cube cell:

$$\mathbf{M} = \frac{1}{6} \sum_{\mathcal{T} \in \{S_0,\dots,S_5\}} \mathbf{M}_\mathcal{T} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}. \tag{34}$$

However, the particular choice of cube diagonal and axis ordering in Figure 2 is arbitrary: permuting (x, y, z) yields an equivalent tetrahedral tiling with the same encoder structure. To avoid baking this convention into our preconditioner, we construct an axis-permutation invariant version of $\mathbf{M}$ by averaging over all permutations of the coordinate axes. Let $\mathbf{P}^{(0)}, \dots, \mathbf{P}^{(5)}$ be the six 3×3 permutation matrices

$$\mathbf{P}^{(0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}^{(1)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\mathbf{P}^{(3)} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{P}^{(4)} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{P}^{(5)} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

We then define:

$$\mathbf{M}_{\text{sym}} = \frac{1}{6} \sum_{k=0}^{5} \mathbf{P}^{(k)} \mathbf{M} \mathbf{P}^{(k)\top} = \frac{1}{3} \begin{bmatrix} 5 & -2 & -2 \\ -2 & 5 & -2 \\ -2 & -2 & 5 \end{bmatrix}. \tag{35}$$

This $\mathbf{M}_{\text{sym}}$ is invariant under axis relabeling and serves as a convention-free average metric for the canonical tetrahedral tiling. It has eigenvalues $\left(\frac{1}{3}, \frac{7}{3}, \frac{7}{3}\right)$, with $\mathbf{u}_\parallel = \frac{1}{\sqrt{3}}(1,1,1)^\top$ as eigenvector for $\frac{1}{3}$, and the eigenspace for $\frac{7}{3}$ given by $\{\mathbf{u} \in \mathbb{R}^3 \mid \mathbf{u}^\top \mathbf{u}_\parallel = 0\}$. Hence the spectral condition number, defined as the ratio of the largest to the smallest eigenvalue, is

$$\kappa(\mathbf{M}_{\text{sym}}) = \frac{\lambda_{\max}}{\lambda_{\min}} = 7. \tag{36}$$

Thus, with respect to this average metric, the encoder-induced geometry attenuates motion along the body diagonal and amplifies any transverse direction by a factor $\sqrt{7}$ in magnitude, on average over canonical cube cells.

Global linear preconditioner We start by expressing the global preconditioner as an affine map in homogeneous coordinates:

$$\tilde{\mathbf{y}} = \mathbf{T}\tilde{\mathbf{x}}, \qquad \mathbf{T} = \begin{bmatrix} \mathbf{A}^* & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \qquad \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \quad \tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix}. \tag{37}$$

where $\mathbf{A}^* \in \mathbb{R}^{3\times3}$ is invertible. Then, inside a tetrahedron $\mathcal{T}$, we have $\mathbf{w}(\mathbf{y}) = \mathbf{C}_\mathcal{T}^{-1}(\mathbf{y} - \mathbf{v}_0)$ with $\mathbf{y} = \mathbf{A}^*\mathbf{x} + \mathbf{t}$, and hence, as a function of $\mathbf{x}$,

$$\mathbf{w}(\mathbf{x}) = \mathbf{C}_\mathcal{T}^{-1}(\mathbf{A}^*\mathbf{x} + \mathbf{t} - \mathbf{v}_0). \tag{38}$$

Differentiating with respect to $\mathbf{x}$ gives:

$$d\mathbf{w} = \mathbf{C}_\mathcal{T}^{-1}\mathbf{A}^*\, d\mathbf{x}, \quad \|d\mathbf{w}\|^2 = d\mathbf{x}^\top \mathbf{A}^{*\top} \mathbf{M}_\mathcal{T} \mathbf{A}^*\, d\mathbf{x}, \tag{39}$$

where $\mathbf{M}_\mathcal{T}$ is the original local metric from Eq. (33). Here the translation $\mathbf{t}$ cancels in the differential and does not affect the metric. Thus the linear map $\mathbf{A}^*$ genuinely changes the encoder-induced geometry in the input space. We now average these preconditioned local metrics over the canonical cube and over all axis permutations. Using the permutation matrices $\mathbf{P}^{(k)}$ and the definition of $\mathbf{M}_{\text{sym}}$ from Eq. (34) and Eq. (35), we obtain

$$\frac{1}{36} \sum_{k=0}^{5} \sum_{\mathcal{T} \in \{S_0,\dots,S_5\}} \mathbf{A}^{*\top} \mathbf{P}^{(k)} \mathbf{M}_\mathcal{T} \mathbf{P}^{(k)\top} \mathbf{A}^* = \mathbf{A}^{*\top} \mathbf{M}_{\text{sym}} \mathbf{A}^*. \tag{40}$$

Thus, the effective average metric seen from the input space is $\mathbf{A}^{*\top}\mathbf{M}_{\mathrm{sym}}\mathbf{A}^{*}$. We choose $\mathbf{A}^{*}$ to *isotropize* this average and enforce volume preservation to avoid global rescaling:

$$\mathbf{A}^{*\top}\mathbf{M}_{\mathrm{sym}}\mathbf{A}^{*} = c\,\mathbf{I}, \quad \det(\mathbf{A}^{*}) = 1. \tag{41}$$

Under these conditions, the preconditioned average encoder-induced metric is isotropic and has unit condition number $\kappa(\mathbf{A}^{*\top}\mathbf{M}_{\mathrm{sym}}\mathbf{A}^{*}) = 1$:

$$d\mathbf{x}^{\top}\mathbf{A}^{*\top}\mathbf{M}_{\mathrm{sym}}\mathbf{A}^{*}d\mathbf{x} = c\,\|d\mathbf{x}\|^{2}. \tag{42}$$

Thus, with respect to the symmetrized cube-average metric, all directions in the preconditioned input space are treated equally; on average over canonical cells, the encoder no longer imposes a preferred direction on infinitesimal displacements. Solving Eq. (41) with $\mathbf{M}_{\mathrm{sym}}$ yields the closed-form preconditioner:

$$\mathbf{A}^{*} = \frac{7^{1/3}}{3}\begin{bmatrix} 1+\frac{2}{\sqrt{7}} & 1-\frac{1}{\sqrt{7}} & 1-\frac{1}{\sqrt{7}} \\ 1-\frac{1}{\sqrt{7}} & 1+\frac{2}{\sqrt{7}} & 1-\frac{1}{\sqrt{7}} \\ 1-\frac{1}{\sqrt{7}} & 1-\frac{1}{\sqrt{7}} & 1+\frac{2}{\sqrt{7}} \end{bmatrix}. \tag{43}$$

which satisfies $\det(\mathbf{A}^{*}) = 1$ and $\mathbf{A}^{*\top}\mathbf{M}_{\mathrm{sym}}\mathbf{A}^{*} = c\,\mathbf{I}$ with $c = 7^{2/3}/3$. Numerically, we use

$$\mathbf{A}^{*} \approx \begin{bmatrix} 1.11965708 & 0.39663705 & 0.39663705 \\ 0.39663705 & 1.11965708 & 0.39663705 \\ 0.39663705 & 0.39663705 & 1.11965708 \end{bmatrix}. \tag{44}$$

To quantify the effect of this preconditioner, Tab. A1 reports spectral condition numbers $\kappa(\cdot)$ of the encoder-induced metrics on the canonical cube before and after applying $\mathbf{A}^{*}$.

Case	$\kappa(\mathbf{M})$	$\kappa(\mathbf{A}^{*\top}\mathbf{M}\mathbf{A}^{*})$
$\mathbf{M}_{\mathrm{sym}}$	7.00	**1.00**
$\mathbf{M}_{S_0}$	16.39	**5.05**
$\mathbf{M}_{S_1}$	16.39	**5.05**
$\mathbf{M}_{S_2}$	16.39	**5.05**
$\mathbf{M}_{S_3}$	16.39	**5.05**
$\mathbf{M}_{S_4}$	16.39	**5.05**
$\mathbf{M}_{S_5}$	16.39	**5.05**

Table A1. Condition numbers of encoder-induced metrics on the canonical unit cube with the tetrahedral subdivision shown in Fig. 2, before and after applying the global preconditioner $\mathbf{A}^{*}$.

A.3. Polyhedral cells

Recall from Sec. 4.1 that the multi-resolution tetrahedral encoder partitions the bounded domain $\mathcal{S} \subset \mathbb{R}^3$ into $\mathbf{x}$-containing tetrahedra $\mathcal{T}^{(\ell)}(\mathbf{x})$ at each resolution level $\ell \in \{0, 1, \ldots, L-1\}$. For $\mathbf{x} \in \mathcal{S}$, we define the encoder-induced polyhedral cell of $\mathbf{x}$ as:

$$\mathcal{C}_{\mathbf{x}} := \bigcap_{\ell=0}^{L-1} \mathcal{T}^{(\ell)}(\mathbf{x}). \tag{45}$$

Lemma 2 (Convexity of polyhedral cells). For any $\mathbf{x} \in \mathcal{S}$, the polyhedral cell $\mathcal{C}_{\mathbf{x}}$ is a convex polyhedron.

Proof. Each tetrahedron $\mathcal{T}^{(\ell)}(\mathbf{x})$ can be written as the intersection of four half-spaces of the form

$$\mathcal{T} = \left\{\mathbf{y} \in \mathbb{R}^3 \,\middle|\, \mathbf{a}_j^{\top}\mathbf{y} \le b_j,\ j \in \{0,1,2,3\},\ b_j \in \mathbb{R}\right\}, \tag{46}$$

and half-spaces are convex by definition. Hence each $\mathcal{T}^{(\ell)}(\mathbf{x})$ is convex. Because the intersection of convex sets remains convex [2], the cell $\mathcal{C}_{\mathbf{x}}$ is also convex. □

Remark (restated). Notice that the encoder τ is piecewise affine and continuous over $\mathcal{S}$. When $L = 1$, τ is affine with respect to $\mathbf{x}$ within each tetrahedron, while for $L > 1$, τ is affine with respect to $\mathbf{x}$ within each cell $\mathcal{C}_{\mathbf{x}}$. In the special case where the geometric progression ratio

$$\gamma := \exp\left(\frac{\ln N_{\max} - \ln N_{\min}}{L-1}\right) \tag{47}$$

satisfies $\gamma \in \mathbb{N}$ and $\gamma \geq 2$, every cell $\mathcal{C}_{\mathbf{x}}$ is a tetrahedron at the finest level.

A.4. Implementation details of initial skeleton extraction

In the initial skeleton extraction step, our goal is to extract all vertices and edges of $\mathcal{C}$ from the multi-resolution tetrahedral grid. This section provides the detailed algorithmic construction described in Sec. 4.3.

A.4.1. Grid representation as a union of parallel-plane sets

We exploit the inherent regularity of the multi-resolution tetrahedral grid to represent $\mathcal{C}$ using a union of parallel-plane sets. As the slopes of tetrahedral subdivision are constant across levels, their normal set is identical while their offsets differ at levels. Thus, multiple planes compactly represent sharing a single normal varying their offsets. We denote these unique normals as:

$$\mathcal{N} = \{\mathbf{n}_0, \mathbf{n}_1, \dots, \mathbf{n}_i, \dots\}. \tag{48}$$

For each $\mathbf{n}_i$, the associated offsets across all resolution levels are defined as a vector as follows:

$$\mathbf{d}^{(i)} = \left[d_0^{(i)}, d_1^{(i)}, \dots\right]^\top. \tag{49}$$

This representation makes it easier to extract the vertices and edges of $\mathcal{C}$. Moreover, the formulation scales efficiently to large numbers of vertices and edges.

A.4.2. Vertex extraction

A vertex of $\mathcal{C}$ is the intersection of at least three planes whose normals are linearly independent. We define the set of such normal triplets by:

$$\mathcal{T}^{(ijk)} := \left\{\{\mathbf{n}_i, \mathbf{n}_j, \mathbf{n}_k\} \in [\mathcal{N}]^3 \;\middle|\; \det[\mathbf{n}_i\ \mathbf{n}_j\ \mathbf{n}_k] \neq 0\right\}, \tag{50}$$

where $[\mathcal{N}]^k$ denotes the set of all k-element subsets of $\mathcal{N}$. For each triplet, we define the normal matrix as:

$$\mathbf{N}^{(ijk)} := \begin{bmatrix}\mathbf{n}_i & \mathbf{n}_j & \mathbf{n}_k\end{bmatrix} \in \mathbb{R}^{3\times 3}, \tag{51}$$

and the corresponding offset matrix is given by:

$$\mathbf{D}^{(ijk)} = \begin{bmatrix} \left(\mathbf{d}^{(i)} \otimes \mathbf{1}_{|\mathbf{d}^{(j)}|} \otimes \mathbf{1}_{|\mathbf{d}^{(k)}|}\right)^\top \\ \left(\mathbf{1}_{|\mathbf{d}^{(i)}|} \otimes \mathbf{d}^{(j)} \otimes \mathbf{1}_{|\mathbf{d}^{(k)}|}\right)^\top \\ \left(\mathbf{1}_{|\mathbf{d}^{(i)}|} \otimes \mathbf{1}_{|\mathbf{d}^{(j)}|} \otimes \mathbf{d}^{(k)}\right)^\top \end{bmatrix} \in \mathbb{R}^{3\times(|\mathbf{d}^{(i)}||\mathbf{d}^{(j)}||\mathbf{d}^{(k)}|)}, \tag{52}$$

where $\mathbf{1}_{|\mathbf{d}|}$ denotes the column vector of ones of length $|\mathbf{d}|$ and $\otimes$ denotes the Kronecker product, which can be viewed as the tensor product collapsed into a vector in our case.
Thus, the vertices for the chosen normal triplet $\mathcal{T}^{(ijk)}$ are the solution of linear equation:

$$\mathbf{N}^{(ijk)\top}\mathbf{X}^{(ijk)} = \mathbf{D}^{(ijk)} \tag{53}$$

which can be obtained by:

$$\mathbf{X}^{(ijk)} = \mathbf{N}^{(ijk)-\top}\mathbf{D}^{(ijk)}. \tag{54}$$

Notice that each column of $\mathbf{X}^{(ijk)}$ corresponds to a vertex. By collecting these columns over all triplets of $\mathcal{T}$ yields the candidate vertex set $\mathcal{X}_{\text{all}}$. Since planes are unbounded, some candidate vertices may fall outside the bounded domain $\mathcal{S}$. We therefore define the final extracted vertex set as:

$$\mathcal{V} := \mathcal{X}_{\text{all}} \cap \mathcal{S}. \tag{55}$$

Note that the computation can be efficiently parallelized over triplets via tensor broadcasting.

A.4.3. Edge extraction

An edge of $\mathcal{C}$ is the intersection of at least two non-parallel planes. Similarly, we define the set of normal pairs describing the two planes as follows:

$$\mathcal{P} := \{\{\mathbf{n}_i, \mathbf{n}_j\} \in [\mathcal{N}]^2 \mid \mathbf{n}_i \times \mathbf{n}_j \neq \mathbf{0}\}. \tag{56}$$

For each $\{\mathbf{n}_i, \mathbf{n}_j\} \in \mathcal{P}$, we define $\mathbf{u}_{ij} := \frac{\mathbf{n}_i \times \mathbf{n}_j}{||\mathbf{n}_i \times \mathbf{n}_j||}$ as the unit direction vector of the line of intersection between the two planes. Note that the offsets of the planes are not required for this.
Given the above normals $\mathbf{n}_i$ and $\mathbf{n}_j$ together with their offsets $d_n^{(i)}$ and $d_m^{(j)}$ and the previously extracted vertex set $\mathcal{V}$, the endpoints of the edges are required to satisfy:

$$\mathcal{V}_{nm}^{(ij)} := \{\mathbf{v} \in \mathcal{V} \mid \mathbf{n}_i^\top \mathbf{v} = d_n^{(i)},\ \mathbf{n}_j^\top \mathbf{v} = d_m^{(j)}\}. \tag{57}$$

However, generating edges directly from $\mathcal{V}_{mn}^{(ij)}$ may results in overlaps. To avoid this, we order them by the projection of each vertex onto $\mathbf{u}_{ij}$:

$$t_{mn}^{(ij)}(\mathbf{v}) := \mathbf{u}_{ij}^\top \mathbf{v}, \quad \mathbf{v} \in \mathcal{V}_{mn}^{(ij)}. \tag{58}$$

This scalar projection represents the distance from the origin. The sequence obtained by sorting $\mathcal{V}_{mn}^{(ij)}$ in ascending t_{ij} order is as follows:

$$\widetilde{\mathcal{V}}_{mn}^{(ij)} = \big[\mathbf{v}_0, \ldots, \mathbf{v}_\ell, \mathbf{v}_{\ell+1}, \ldots\big]. \tag{59}$$

The non-overlapping local edge set is then obtained by connecting consecutive vertices in $\widetilde{\mathcal{V}}_{mn}^{(i,j)}$:

$$\mathcal{E}_{mn}^{(ij)} := \big\{\{\mathbf{v}_\ell,\ \mathbf{v}_{\ell+1}\} \mid \mathbf{v}_\ell, \mathbf{v}_{\ell+1} \in \widetilde{\mathcal{V}}_{mn}^{(i,j)}, 0 \leq \ell < |\widetilde{\mathcal{V}}_{mn}^{(i,j)}| - 1\big\}. \tag{60}$$

The whole edge set $\mathcal{E}$ is obtained by taking the union of $\mathcal{E}_{mn}^{(ij)}$ over all normal pair and offset pair combinations. Since the vertex set $\mathcal{V}$ is already defined within the bounded domain, no additional post-processing is required to be within $\mathcal{S}$. Once again, this computation can be parallelized efficiently using tensor broadcasting.

A.5. Neighbor configuration lookup tables for the tetrahedral grid

We report the lookup tables of neighbor configurations used in Sec. 4.5 for the tetrahedral grid in Fig. 2, divided into face, edge, and vertex neighbors. Each tetrahedron $\mathcal{T}$ has ordered vertices (v_0, v_1, v_2, v_3). We fix the local face and edge indices as follows: face index $f \in \{0, 1, 2, 3\}$ refers to the face opposite vertex v_f (*e.g.*, $f = 0$ corresponds to the face $\{v_1, v_2, v_3\}$), and edge index $e \in \{0, \ldots, 5\}$ refers to

$$e_0 = \{v_0, v_1\},\ e_1 = \{v_0, v_2\},\ e_2 = \{v_0, v_3\},\ e_3 = \{v_1, v_2\},\ e_4 = \{v_1, v_3\},\ e_5 = \{v_2, v_3\}. \tag{61}$$

For each configuration, the neighbor index i runs over $\{0, 1, \ldots\}$, where $i = 0$ denotes the self configuration (Δ_0=$(0, 0, 0)$, t_0=t). In the face and edge neighbor tables, we list only the non-trivial neighbors with indices $i \geq 1$ and do not explicitly include the $i = 0$ row. For the vertex neighbor table, the lookup is defined per cube corner; for a given cube corner c and tetra index t, one of these rows may coincide with the self configuration.

Table A2. Face neighbor lookup table. For each tetra index $t \in \{0, \ldots, 5\}$ and local face index $f \in \{0, \ldots, 3\}$, we list the neighbor index $i \geq 1$, the anchor displacement $\Delta_i = (\Delta_{x,i}, \Delta_{y,i}, \Delta_{z,i}) \in \mathbb{Z}^3$, and the neighbor tetra index t_i. In the table, the columns $\Delta_x, \Delta_y, \Delta_z$ give the components of Δ_i for each row i. These entries correspond to the configurations (Δ_i, t_i) used in Sec. 4.5. The self configuration $(i{=}0, \Delta_0{=}(0,0,0), t_0{=}t)$ is omitted.

Input	f	i	Δ_x	Δ_y	Δ_z	t_i	Input	f	i	Δ_x	Δ_y	Δ_z	t_i
$t{=}0$	0	1	1	0	0	4	$t{=}3$	0	1	0	1	0	5
	1	1	0	0	0	5		1	1	0	0	0	4
	2	1	0	0	0	1		2	1	0	0	0	2
	3	1	0	-1	0	2		3	1	-1	0	0	1
$t{=}1$	0	1	1	0	0	3	$t{=}4$	0	1	0	0	1	2
	1	1	0	0	0	2		1	1	0	0	0	3
	2	1	0	0	0	0		2	1	0	0	0	5
	3	1	0	0	-1	5		3	1	-1	0	0	0
$t{=}2$	0	1	0	1	0	0	$t{=}5$	0	1	0	0	1	1
	1	1	0	0	0	1		1	1	0	0	0	0
	2	1	0	0	0	3		2	1	0	0	0	4
	3	1	0	0	-1	4		3	1	0	-1	0	3

Table A3. Edge neighbor lookup tables for tetra indices $t = 0$–5. For each input edge (t, e) and neighbor index $i \geq 1$, we list the anchor displacement $\Delta_i = (\Delta_{x,i}, \Delta_{y,i}, \Delta_{z,i})$ and the neighbor tetra index t_i. In the table, the columns $\Delta_x, \Delta_y, \Delta_z$ give the components of Δ_i for each row i. These entries correspond to the configurations (Δ_i, t_i) used in Sec. 4.5. The self configuration $(i{=}0, \Delta_0{=}(0,0,0), t_0{=}t)$ is omitted.

Input	i	Δ_x	Δ_y	Δ_z	t_i
t=0, e=0	1	0	−1	−1	3
	2	0	−1	−1	4
	3	0	0	−1	5
	4	0	−1	0	2
	5	0	0	0	1
t=0, e=1	1	0	−1	0	2
	2	0	−1	0	3
	3	0	0	0	5
t=0, e=2	1	0	0	0	1
	2	0	0	0	2
	3	0	0	0	3
	4	0	0	0	4
	5	0	0	0	5
t=0, e=3	1	0	−1	0	1
	2	0	−1	0	2
	3	1	−1	0	3
	4	1	0	0	4
	5	1	0	0	5
t=0, e=4	1	0	0	0	1
	2	1	0	0	3
	3	1	0	0	4
t=0, e=5	1	0	0	0	5
	2	1	0	0	4
	3	0	0	1	1
	4	1	0	1	2
	5	1	0	1	3

Input	i	Δ_x	Δ_y	Δ_z	t_i
t=1, e=0	1	0	−1	−1	3
	2	0	−1	−1	4
	3	0	0	−1	5
	4	0	−1	0	2
	5	0	0	0	0
t=1, e=1	1	0	0	−1	4
	2	0	0	−1	5
	3	0	0	0	2
t=1, e=2	1	0	0	0	0
	2	0	0	0	2
	3	0	0	0	3
	4	0	0	0	4
	5	0	0	0	5
t=1, e=3	1	0	0	−1	0
	2	0	0	−1	5
	3	1	0	−1	4
	4	1	0	0	2
	5	1	0	0	3
t=1, e=4	1	0	0	0	0
	2	1	0	0	3
	3	1	0	0	4
t=1, e=5	1	0	0	0	2
	2	1	0	0	3
	3	0	1	0	0
	4	1	1	0	4
	5	1	1	0	5

Input	i	Δ_x	Δ_y	Δ_z	t_i
t=2, e=0	1	−1	0	−1	0
	2	−1	0	−1	5
	3	0	0	−1	4
	4	−1	0	0	1
	5	0	0	0	3
t=2, e=1	1	0	0	−1	4
	2	0	0	−1	5
	3	0	0	0	1
t=2, e=2	1	0	0	0	0
	2	0	0	0	1
	3	0	0	0	3
	4	0	0	0	4
	5	0	0	0	5
t=2, e=3	1	0	0	−1	3
	2	0	0	−1	4
	3	0	1	−1	5
	4	0	1	0	0
	5	0	1	0	1
t=2, e=4	1	0	0	0	3
	2	0	1	0	0
	3	0	1	0	5
t=2, e=5	1	0	0	0	1
	2	1	0	0	3
	3	0	1	0	0
	4	1	1	0	4
	5	1	1	0	5

Input	i	Δ_x	Δ_y	Δ_z	t_i
t=3, e=0	1	−1	0	−1	0
	2	−1	0	−1	5
	3	0	0	−1	4
	4	−1	0	0	1
	5	0	0	0	2
t=3, e=1	1	−1	0	0	0
	2	−1	0	0	1
	3	0	0	0	4
t=3, e=2	1	0	0	0	0
	2	0	0	0	1
	3	0	0	0	2
	4	0	0	0	4
	5	0	0	0	5
t=3, e=3	1	−1	0	0	1
	2	−1	0	0	2
	3	−1	1	0	0
	4	0	1	0	4
	5	0	1	0	5
t=3, e=4	1	0	0	0	2
	2	0	1	0	0
	3	0	1	0	5
t=3, e=5	1	0	0	0	4
	2	0	1	0	5
	3	0	0	1	2
	4	0	1	1	0
	5	0	1	1	1

Input	i	Δ_x	Δ_y	Δ_z	t_i
t=4, e=0	1	−1	−1	0	1
	2	−1	−1	0	2
	3	0	−1	0	3
	4	−1	0	0	0
	5	0	0	0	5
t=4, e=1	1	−1	0	0	0
	2	−1	0	0	1
	3	0	0	0	3
t=4, e=2	1	0	0	0	0
	2	0	0	0	1
	3	0	0	0	2
	4	0	0	0	3
	5	0	0	0	5
t=4, e=3	1	−1	0	0	0
	2	−1	0	0	5
	3	−1	0	1	1
	4	0	0	1	2
	5	0	0	1	3
t=4, e=4	1	0	0	0	5
	2	0	0	1	1
	3	0	0	1	2
t=4, e=5	1	0	0	0	3
	2	0	1	0	5
	3	0	0	1	2
	4	0	1	1	0
	5	0	1	1	1

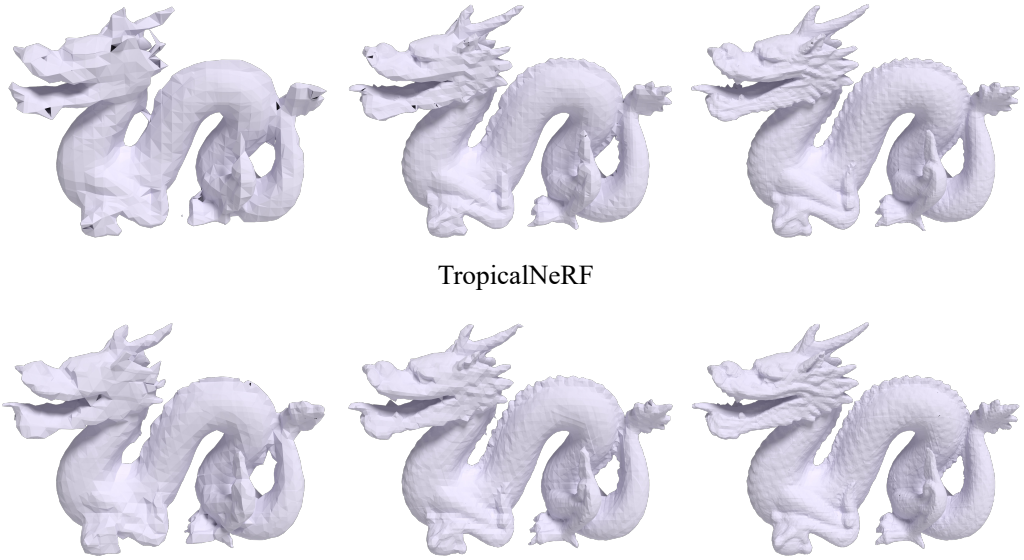
Input	i	Δ_x	Δ_y	Δ_z	t_i
t=5, e=0	1	−1	−1	0	1
	2	−1	−1	0	2
	3	0	−1	0	3
	4	−1	0	0	0
	5	0	0	0	4
t=5, e=1	1	0	−1	0	2
	2	0	−1	0	3
	3	0	0	0	0
t=5, e=2	1	0	0	0	0
	2	0	0	0	1
	3	0	0	0	2
	4	0	0	0	3
	5	0	0	0	4
t=5, e=3	1	0	−1	0	3
	2	0	−1	0	4
	3	0	−1	1	2
	4	0	0	1	0
	5	0	0	1	1
t=5, e=4	1	0	0	0	4
	2	0	0	1	1
	3	0	0	1	2
t=5, e=5	1	0	0	0	0
	2	1	0	0	4
	3	0	0	1	1
	4	1	0	1	2
	5	1	0	1	3

Table A4. Vertex neighbor lookup table around cube corner $(0, 0, 0)$. For each neighbor index i, we list the anchor displacement $\Delta_i = (\Delta_{x,i}, \Delta_{y,i}, \Delta_{z,i})$ and the neighbor tetra index t_i. These entries correspond to the configurations (Δ_i, t_i) used in Sec. 4.5. For a general cube corner $c = (c_x, c_y, c_z) \in \{0, 1\}^3$, we obtain the corresponding offsets by replacing $\Delta_i \leftarrow \Delta_i + c$. Row index ranges (*e.g.*, 1–6) indicate consecutive neighbor indices that share the same (Δ_i, t_i) configuration. Note that, unlike the face and edge tables, this vertex table is defined per cube corner for convenience, since the vertices of each tetrahedron can be mapped to a cube corner. For a specific cube corner c and tetra index t, one of these rows may coincide with the self configuration $(\Delta_0{=}(0, 0, 0), t_0{=}t)$.

i	Δ_x	Δ_y	Δ_z	t_i
1–6	−1	−1	−1	0–5
7–8	−1	−1	0	1, 2
9–10	−1	0	−1	0, 5
11–12	−1	0	0	0, 1
13–14	0	−1	−1	3, 4
15–16	0	−1	0	2, 3
17–18	0	0	−1	4, 5
19–24	0	0	0	0–5

A.6. Qualitative comparison across resolution settings

Fig. A1 shows the qualitative effect of the three resolution settings (*Small*, *Medium*, *Large*) on the Stanford Dragon, complementing the quantitative results in Tab. 3. As the resolution decreases, TropicalNeRF loses fine details much more rapidly than our method. Our method still produces visually plausible meshes even under the *Small* setting.



TropicalNeRF

Ours

Figure A1. Qualitative results on the Stanford Dragon across resolution settings: *Small* (left), *Medium* (middle), and *Large* (right).