TetWeave: Isosurface Extraction using On-The-Fly Delaunay Tetrahedral Grids for Gradient-Based Mesh Optimization

ALEXANDRE BINNINGER, ETH Zurich, Switzerland
RUBEN WIERSMA, ETH Zurich, Switzerland
PHILIPP HERHOLZ, Zurich, Switzerland
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland

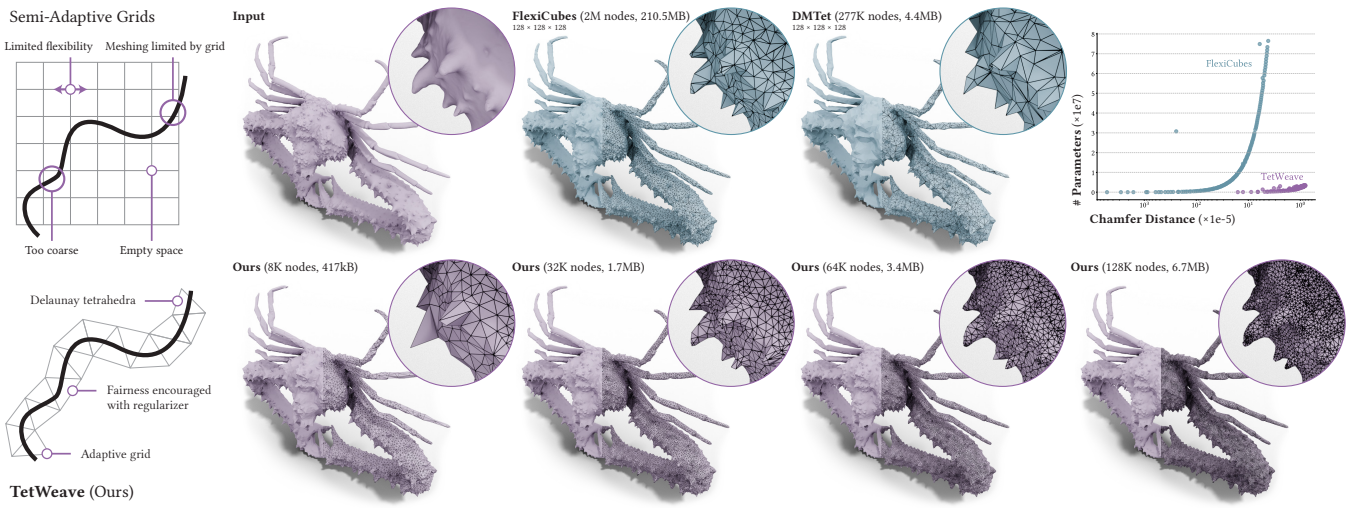


Fig. 1. **TetWeave** jointly optimizes a tetrahedral grid and a directional signed distance function used for Marching Tetrahedra. Our method *weaves* a background grid around the surface, which is regularized to give fair output meshes. The results are compared with semi-adaptive grid methods (top row), which start from a predefined grid. These methods have limited flexibility and their meshing is influenced by the configuration of the predefined grid. They require many more nodes in the background grid because of unused empty space to get a similar level of detail in the output mesh (graph in top right).

We introduce TetWeave, a novel isosurface representation for gradient-based mesh optimization that jointly optimizes the placement of a tetrahedral grid used for Marching Tetrahedra and a novel directional signed distance at each point. TetWeave constructs tetrahedral grids on-the-fly via Delaunay triangulation, enabling increased flexibility compared to predefined grids. The extracted meshes are guaranteed to be watertight, two-manifold and intersection-free. The flexibility of TetWeave enables a resampling strategy that places new points where reconstruction error is high and allows to encourage mesh fairness without compromising on reconstruction error. This leads to high-quality, adaptive meshes that require minimal memory usage and few parameters to optimize. Consequently, TetWeave exhibits near-linear memory scaling relative to the vertex count of the output mesh — a substantial improvement over predefined grids. We demonstrate the applicability of TetWeave to a broad range of challenging tasks in computer graphics and vision, such as multi-view 3D reconstruction, mesh compression and geometric texture generation. Our code is available at https://github.com/AlexandreBinninger/TetWeave.

CCS Concepts: • **Computing methodologies** → **Mesh geometry models**; *Shape representations*; Reconstruction.

Additional Key Words and Phrases: isosurface mesh extraction, gradient-based mesh optimization, photogrammetry

ACM Reference Format:
Alexandre Binninger, Ruben Wiersma, Philipp Herholz, and Olga Sorkine-Hornung. 2025. TetWeave: Isosurface Extraction using On-The-Fly Delaunay Tetrahedral Grids for Gradient-Based Mesh Optimization. *ACM Trans. Graph.* 44, 4 (August 2025), 19 pages. https://doi.org/10.1145/3730851

1 Introduction

Many recent 3D applications require shape representations that are both expressive for artists and differentiable for optimization. Differentiable shape representations enable appealing applications, such as shape generation [Gao et al. 2022], text-to-3D synthesis [Poole et al. 2022], inverse rendering [Munkberg et al. 2022] and geometric texture synthesis [Hertz et al. 2020; Liu et al. 2018]. An illustrative task using differentiable shape representations is surface reconstruction from multi-view images. In this task, the objective is to recover the 3D geometry of an object from a set of input views. A common approach is to optimize a shape representation to match

Authors' Contact Information: Alexandre Binninger, ETH Zurich, Zurich, Switzerland, alexandre.binninger@inf.ethz.ch; Ruben Wiersma, ETH Zurich, Zurich, Switzerland, ruben.wiersma@inf.ethz.ch; Philipp Herholz, Zurich, Switzerland, ph.herholz@gmail.com; Olga Sorkine-Hornung, ETH Zurich, Zurich, Switzerland, sorkine@inf.ethz.ch.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3730851

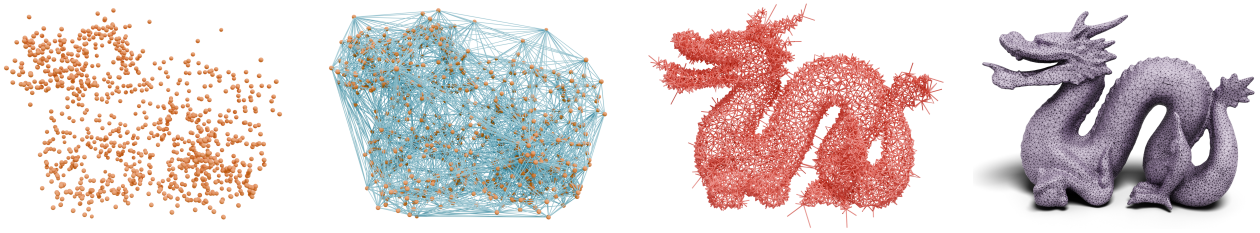


Fig. 2. Illustration of our mesh extraction pipeline, which begins with a point cloud where each point is associated with a signed distance value. The process starts by generating a tetrahedral grid through Delaunay triangulation. Next, active edges are identified, and a directional signed distance is computed for each active point using spherical harmonics (Sec. 3.1). The final mesh is extracted using the Marching Tetrahedra algorithm (Sec. 3.2). Our method iteratively refines the randomly initialized point cloud, distributing points to closely align with the target shape (Sec. 4.3), which ensures scalability and adaptability. In this figure, only a portion of the point cloud and Delaunay Triangulation is displayed to enhance clarity.

the input views using gradient descent. This requires a differentiable renderer and, of interest to us, a differentiable shape representation. While triangle meshes are widely used in computer graphics as an efficient and practical shape representation and thus highly desirable, they are difficult to use in a differentiable setting, especially when the topology is not known a priori, as is the case with surface reconstruction. To address these limitations, considerable effort has been invested in developing representations capable of generating meshes for gradient-based optimization, rather than relying on direct mesh optimization.

A typical approach in shape optimization is to optimize a signed distance function and then generate a mesh using an isosurface extraction method, like Marching Cubes [Lorensen and Cline 1987]. This approach inherits limitations from the isosurface extraction technique. For instance, Marching Cubes is prone to staircasing artifacts. To overcome these challenges, works like DMTet [Shen et al. 2021] and FlexiCubes [Shen et al. 2023] jointly optimize the implicit representation and the spatial grid structure from which the final mesh is extracted. We refer to these approaches as *grid-adaptive isosurface representations*. Still, these approaches face issues, such as high memory consumption due to poor scaling properties, susceptibility to self-intersections and challenges with adaptive meshing across complex multi-scale grids.

To address these issues, we propose TetWeave, a novel differentiable isosurface representation that allows joint optimization of the placement of an unstructured tetrahedral grid and the signed distance at each node of the grid. Contrary to previous methods like DMTet or FlexiCubes that rely on deforming a precomputed grid structure, we use Delaunay triangulation [Delaunay 1934] on an arbitrary point cloud to generate a support structure (which we refer to as a *grid*). This grid is used for isosurface extraction with Marching Tetrahedra [Doi and Koide 1991], which guarantees meshes to be watertight, 2-manifold and intersection-free. These are crucial properties for many downstream applications.

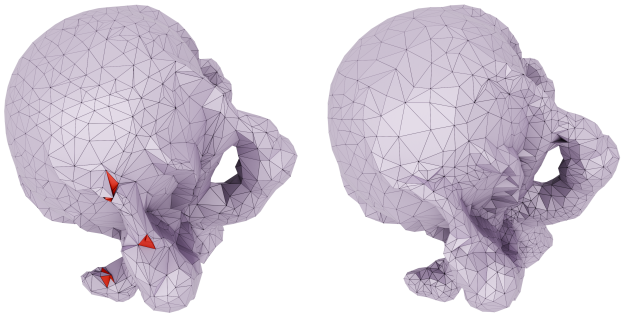
The flexibility of TetWeave comes at a much lower cost than prior work using a predefined grid. For example, FlexiCubes [Shen et al. 2023] starts from a voxel grid, where each grid cell is equipped with parameters that adjust the position of the grid points and the placement of the resulting mesh's vertices. This flexibility requires a considerable number of parameters per grid cell: 21 parameters per grid cell and 3 per grid point. Moreover, because FlexiCubes starts from a voxel grid, the surface resolution scales poorly as the grid resolution increases. This results in a memory-intensive representation that inevitably fails to reconstruct high-frequency details (see Figure 1). TetWeave does not rely on such a predefined grid structure. The grid points are free to move anywhere in ambient space and only require storing a position and the value of the signed distance function at that location.

To allow additional flexibility for the placement of mesh vertices, we introduce the notion of *directional signed distance*, encoded with spherical harmonics coefficients at each point. This allows distinct implicit surface positions along different grid edges (unlike a single SDF per point), providing finer control over vertex placement during extraction. To ensure that running Marching Cubes on the resulting background grid produces high-quality triangulations with minimal sliver triangles, we propose a simple loss function measuring fairness. Finally, our approach incorporates a resampling technique to refine details during optimization, enabling adaptive meshing tailored to customizable objectives, such as reducing reconstruction error. Thus, we achieve linear memory scaling relative to the resolution of the resulting meshes, while producing high-quality, highly detailed meshes with adaptive resolution, as shown in Fig. 1. We demonstrate that this is useful in applications of gradient-based mesh optimization, such as multi-view 3D reconstruction, mesh compression and geometric texture generation.

Summarizing our main contributions:

- We use Marching Tetrahedra on Delaunay triangulations of arbitrary point clouds in gradient-based mesh optimization pipelines.
- A directional signed distance function to more accurately capture the distance to the surface along tetrahedral edges.
- A method to adapt the tetrahedral grid to an unknown surface.
- Two regularization terms to improve the quality of our meshes.

We accentuate that TetWeave is not designed as a general-purpose adaptive meshing technique or for isosurface extraction from fixed scalar fields. Rather, we propose a specialized representation optimized for gradient-based mesh processing, particularly suited for applications like multi-view 3D reconstruction.



FlexiCubes - 128^3 grid resolution TetWeave - 32K points

Fig. 3. Contrary to FlexiCubes [Shen et al. 2023], our method is guaranteed free from self-intersections.

2 Related work

In this section, we briefly review related work on differentiable surface representations, with a particular emphasis on isosurface extraction methods. We refer the reader to the survey by de Araújo et al. [2015] for a more extensive presentation. Table 1 provides a compact comparison of closely related isosurfacing methods. The approaches are compared according to the following criteria, inspired by [Shen et al. 2023]:

- *Grad*: differentiation with respect to the generated mesh is possible and enables robust gradient-based pipelines.
- *Sharp*: sharp features are correctly reconstructed.
- *Fair*: the tessellation is fair, with few sliver triangles.
- *Intersection-free*: the mesh is guaranteed not to self-intersect.
- *2-manifold*: the topology is guaranteed to be two-manifold.
- *Scalable resolution*: the number of vertices scales well with the number of parameters of the underlying representation.

Differentiable shape representations. Some methods differentiate directly on the positions of a (triangular) mesh. Such approaches require careful regularization to avoid degeneracy [Nicolet et al. 2021] and are bound by the topology of the initialization [Hanocka et al. 2020; Wang et al. 2018]. Mesh R-CNN [Gkioxari et al. 2019] uses a two-stage approach to support arbitrary topologies, first predicting the topology before optimizing the vertex positions of a template mesh. AtlasNet [Groueix et al. 2018] predicts parametric patches to reconstruct a mesh, but the patches are not guaranteed to be connected. MeshSDF [Remelli et al. 2020] allows for the computation of gradients from the mesh extraction process.

Several learning methods reconstruct a mesh from a point cloud. PointTriNet [Sharp and Ovsjanikov 2020] iteratively predicts triangle meshes, IERMeshing [Liu et al. 2020] constructs the mesh based on the ratio between the geodesic and Euclidean distance, and DSE [Rakotosaona et al. 2021] builds a triangle mesh via 2D Delaunay triangulation of local projections. DeepDT [Luo et al. 2021] and DMNet [Zhang et al. 2023b] utilize Delaunay triangulation for point cloud meshing, employing neural networks to classify tetrahedra for surface reconstruction. These methods have difficulties ensuring that the resulting meshes are watertight or manifold. By introducing a continuous latent connectivity space at each vertex, SpaceMesh [Shen et al. 2024] trains a neural network that generates a watertight 2-manifold mesh from a point cloud, but cannot guarantee intersection-free outputs.

Neural implicit representations emerged as a powerful representation for continuously generating or interpolating shapes with various topologies. Early work represents shapes with a unique latent vector, and a neural network maps this latent code to a signed distance function (SDF) [Mescheder et al. 2019; Park et al. 2019]. Many improvements over these methods involve latent code decomposition, being in a regular grid structure [Peng et al. 2020] or a multiresolution grid [Müller et al. 2022; Takikawa et al. 2021], as a point cloud [Petrov et al. 2024; Zhang et al. 2022], as a set of local grids [Yang et al. 2024; Yariv et al. 2024], as a set of 3D Gaussians [Hertz et al. 2022], or as a general set [Zhang et al. 2023a] where the weight of the latent code is given via an attention mechanism, rather than spatial proximity. Other methods focus on learning parts of the shape with different frequencies via positional encoding schemes [Hertz et al. 2021; Sitzmann et al. 2020; Tancik et al. 2020] or by learning a high-frequency displacement map on top of a base implicit shape [Yifan et al. 2022]. Despite these improvements, the resulting representations still need to be converted to meshes via isosurface mesh extraction techniques for use in downstream applications or in optimization pipelines operating directly on meshes.

Isosurface mesh extraction is the process of generating a surface mesh from an implicit function, typically by evaluating an SDF on a regular grid [Lorensen and Cline 1987] or a tetrahedral grid [Doi and Koide 1991]. Vertices of the mesh are placed on the edges of the background grid and their connectivity is based on a lookup table. While widely used, this approach fails to reconstruct sharp features and produces unfair tessellations. Dual Contouring [Ju et al. 2002] leverages the dual of an octree grid and optimizes vertices inside cubes based on normals to recover sharp features better, but produces non-manifold, self-intersecting meshes. Schaefer et al. [2007] also use an optimization setting within a Dual Marching Cube formulation by optimizing quadratic error functions (QEF). Placing vertices at the face centroids ensures the differentiability of the isosurface extraction but comes at the expense of flexibility, which is

Table 1. Comparison of isosurface mesh extraction methods in three categories: classic isosurfacing methods are typically used on top of a sign distance field, neural methods uses a neural network to estimate the parameters of an isosurface extraction technique, while grid adaptive methods allow for joint optimization of the grid structure and the mesh extraction's parameters. Criteria are explained in Sec. 2.

		Grad	Sharp	Fair	Intersection Free	2-manifold	Scalable Resolution
classic	MC [Lorensen and Cline 1987]	✓	✗	✗	✓	✗	✗
	Dual Contouring [Ju et al. 2002]	✗	✓	✗	✗	✗	✗
	DMC—centroid [Nielson 2004]	✓	✗	✓	✓	✓	✗
	DMC—QEF [Schaefer et al. 2007]	✗	✓	✓	✓	✓	✗
neural	NMC [Chen and Zhang 2021]	✓	✓	✗	✗	✓	✗
	NDC [Chen et al. 2022]	✗	✓	✓	✓	✗	✗
	Voromesh [Maruani et al. 2023]	✓	✓	✗	✓	✓	✓
grid adaptive	DMTet [Shen et al. 2021]	✓	✓	✗	✓	✓	✗
	FlexiCubes [Shen et al. 2023]	✓	✓	✓	✗	✓	✗
	TetWeave **(Ours)**	✓	✓	✓	✓	✓	✓

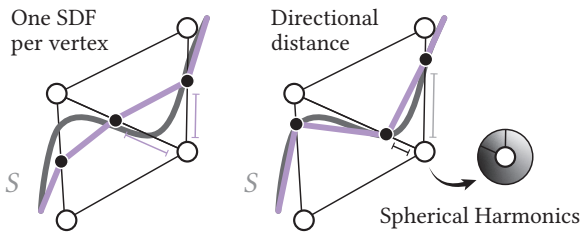


Fig. 4. Storing a single SDF value at each point can lead to inaccurate surface reconstruction (left), where a directional distance encoded as spherical harmonics allows our method to place mesh vertices differently on each tetrahedral edge, resulting in a more accurate shape reconstruction (right).

essential for accurately reconstructing sharp features [Nielson 2004]. Marching Triangles [Hilton et al. 1996] iteratively expands a surface mesh from a starting vertex by enforcing a Delaunay property at each newly added triangle. DelIso [Dey and Levine 2007] proposes a two-stage strategy that extracts a coarse surface from an initial 3D Delaunay triangulation and refines it with a surface-restricted Delaunay triangulation. However, the method is prone to artifacts when the isosurface exhibits sharp edges. More recently, Sellán et al. [2023, 2024] introduce techniques that improve reconstruction by leveraging tangency information derived from signed distance values to extract additional insights from point clouds.

Some methods leverage the use of a neural network to predict the parameters of a regular structure from which they can generate a mesh. DefTet [Gao et al. 2020] predicts the deformation and the occupancy value of a regular tetrahedral grid, Deep Marching Cubes [Liao et al. 2018] and Neural Marching Cubes [Chen and Zhang 2021] learn the vertex positions and mesh topologies in a regular grid, Neural Dual Contouring (NDC) [Chen et al. 2022] predicts the edge crossing and the vertex locations, then extracts a mesh via dual contouring, but can produce non-manifold surfaces. Voromesh [Maruani et al. 2023] learns the position of Voronoi cell generators and then extracts the mesh from the boundary of the occupied cells, but produces many small facets.

Methods for adaptive grid generation in isosurface extraction are well-established. While most approaches combine grid subdivision with local refinement to preserve grid quality [Bey 1995; Hui and Jiang 1999; Ju et al. 2024; Kim et al. 2000; Liu and Joe 1995], their process follows purely geometric criteria. Because our method focuses on mesh optimization, we refine the grid using optimization-derived error signals, typically from 2D inputs, and use an energy-based approach to produce a fair tessellation. Delaunay triangulation has been used in the context of isosurface extraction techniques with adaptive grids. Zhao et al. [2021] progressively refine a 3D Delaunay triangulation to enable coarse-to-fine isosurface extraction. Their method aims at reconstructing a mesh from a point cloud, while TetWeave is designed for gradient-based mesh optimization and reconstruction pipelines. As such, their refinement is driven by a pure geometric analysis based on local curvature estimation, surface smoothness and grid fairness, which can cause their reconstruction to struggle on fine details. In contrast, our error-based refinement strategy is more robust to this limitation and can generalize to different meshing targets. McGrids [Ren et al. 2025] (Monte-Carlo grids) generates a mesh by extracting it from the Delaunay triangulation of an iteratively grown point cloud. Although their approach starts with a similar concept to ours, their objectives and respective contributions differ significantly: McGrids start from a given SDF and develop contributions for sampling the SDF using a Monte-Carlo method. We jointly optimize the SDF and the background grid for an unknown shape. This presents unique challenges in optimization, as this setting is less constrained, but offers opportunities to influence the mesh quality and adaptivity through application-specific metrics.

To overcome the limitations of a fixed background grid for gradient-based mesh optimization, DMTet [Shen et al. 2021] and FlexiCubes [Shen et al. 2023] jointly optimize the implicit representation and the spatial grid structure from which they extract the final mesh. While FlexiCubes can achieve adaptive meshing by leveraging an octree structure instead of a uniform grid, this approach requires specific constraints to reduce the occurrence of non-manifold edges–without guaranteeing their complete removal. Furthermore, FlexiCubes does not provide a principled method to determine where the voxel grid should be refined. These *grid-adaptive isosurface representations* are closest to our work. We focus on removing some of the limitations of these methods, such as high memory consumption due to poor scaling properties and over-parameterized flexibility, susceptibility to self-intersections (see Fig. 3) and challenges with adaptive meshing.

3 Shape representation

We start with a description of the differentiable shape representation of TetWeave. In section 4, we show how this representation is used in an optimization pipeline. TetWeave's shape representation is strikingly simple: it consists of a point cloud $P = \{p_1, p_2, \ldots, p_n\}$, with $p_i \in \mathbb{R}^3$, and each point is associated with a base signed distance value $s_i \in \mathbb{R}$ and a feature vector $\boldsymbol{c}_i \in \mathbb{R}^q$. From this representation, our algorithm constructs a surface mesh in three steps, illustrated in Figure 2:

(1) From P, compute a tetrahedral grid (P, T) using Delaunay triangulation.
(2) For the endpoints of edges whose base signed distances have opposite signs, compute the corresponding directional signed distance $\hat{s}_i(e)$ from s_i and $\boldsymbol{c}_i$. We call such edges *active edges*.
(3) Use Marching Tetrahedra to extract the surface mesh (V, F).

3.1 Directional signed distance

While TetWeave can generate high-quality meshes with only one signed distance per point, this representation can be suboptimal (see Figure 4). During the Marching Tetrahedra step, we require the distance along the edges of the tetrahedral grid to place the mesh vertices. These distances differ per edge and storing one distance results in a compromise between all active edges connected to a point. Therefore, we define a *directional* signed distance function $\hat{s}$ to gain more flexibility in positioning mesh vertices. This approach enables our method to position vertices uniquely based on the specific edge (Fig. 4), better aligning the local normal with the target surface and

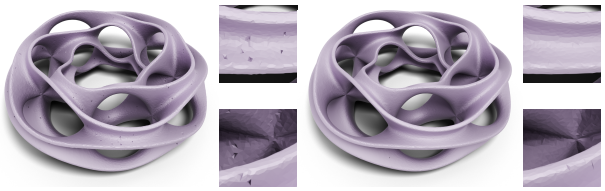


Fig. 5. We compare the reconstructed meshes with and without the use of directional signed distance (spherical harmonics). Using spherical harmonics enhances detail preservation, particularly for shapes with complex topology, where a single grid point can influence multiple parts of the shape.

effectively preventing artifacts when edges of the tetrahedral grid span separated regions of the mesh (Fig. 5).

Our directional signed distance is encoded as spherical harmonics (see Wieczorek and Meschede [2018] for an accessible reference). Each point p_i is associated with a base sign distance s_i and spherical harmonic coefficients $\boldsymbol{c}_i \in \mathbb{R}^q$ with $q = (d+1)^2$, where d is the desired degree of the spherical harmonics representation. Given an active edge e connecting points p_i and p_j, we define the directional signed distance at point p_i for edge e as

$$\hat{s}_i(e) = \left(1 + \tanh\left(SH\left(\theta_{i\to j}, \phi_{i\to j}, \boldsymbol{c}_i\right)\right)\right) s_i, \tag{1}$$

where $\theta_{i\to j}, \phi_{i\to j}$ are the polar and azimuthal angle of the vector $p_j - p_i$ and $SH(\theta, \phi, \boldsymbol{c})$ evaluates the spherical harmonics at the given polar angles for coefficients $\boldsymbol{c}$. Note that our formulation enforces that $\hat{s}_i(e)$ has the same sign as s_i for every edge e and takes values in $(0, 2s_i)$. These two properties align our directional signed distance with Marching Tetrahedra while allowing for more flexibility.

We choose spherical harmonics (SH) over alternatives, such as gradient vectors at grid points, for the following reasons: SH are evaluated as a linear combination of basis functions, which is simple to differentiate; by optimizing only low-frequency spherical harmonics, we can enforce a smooth directional function; and it is straightforward to expand the degrees of freedom by adding more coefficients. This can be useful to handle cusps in the SDF (e.g., points between the dragon's neck and body in Fig. 2).

3.2 Mesh extraction

TetWeave extracts the final mesh using a variant of the Marching Tetrahedra algorithm. We call *active points* the points of (P, T) that belong to at least one active edge. Each active edge contains exactly one point of the extracted mesh (V, F): given two points (p_1, p_2) from an active edge e, the corresponding vertex is given by

$$v_e = \frac{\hat{s}_2(e)p_1 - \hat{s}_1(e)p_2}{\hat{s}_2(e) - \hat{s}_1(e)}. \tag{2}$$

The surface connectivity F, is retrieved from a lookup table, based on the sign of the edges, which is the same lookup table as for regular Marching Tetrahedra. Configurations are displayed in Fig. 21.

4 Optimization Pipeline

A typical optimization task for TetWeave is multi-view 3D reconstruction, which we use to illustrate the optimization components of our method. Most of the components described in this section could operate in any optimization pipeline where gradients are required. We make clear along the way when a component is specific to multi-view 3D reconstruction.

4.1 Multi-view 3D Reconstruction Objective

For multi-view 3D reconstruction, we are given a set of input views and corresponding camera intrinsic and extrinsics. We aim to recover a mesh (V, F), representing the surface geometry of the captured object. The set of input views can be photographs captured in the wild (subsection 6.3) or, in the case of our validation experiments in section 5, rendered masks M_{gt}, depth maps D_{gt}, and normal maps N_{gt} of a given ground-truth mesh $(V_{\text{gt}}, F_{\text{gt}})$. We render the mesh resulting from TetWeave with a differentiable rasterizer [Laine et al. 2020] and compute a loss in image space. We then use autograd to compute the gradients of the loss function w.r.t. the parameters in TetWeave (point positions p_i and their coefficients s_i and $\mathbf{c}_i$) and use gradient descent to minimize the loss function.

For our multi-view 3D reconstruction experiments in section 5, we use the following loss function (the weights $\lambda_M, \lambda_D, \lambda_N$ are detailed in Appendix A)

$$\begin{aligned}\mathcal{L}_{\text{recons}} = \lambda_{\text{M}}\|M - M_{\text{gt}}\| + \lambda_{\text{D}}\|M_{\text{gt}}(D - D_{\text{gt}})\|^2 \\ + \lambda_{\text{N}}\|M_{\text{gt}}(N - N_{\text{gt}})\|^2.\end{aligned} \tag{3}$$

4.2 Regularization

We propose to use several regularizers that enhance the quality of the mesh output. These regularizers can be used for any application. Because the grid topology in TetWeave is not fixed, our representation allows full flexibility over the position of the grid points while ensuring that the grid is embedded (i.e., there are no self-intersections). This flexibility allows us to use regularizers that improve the quality of the resulting mesh. Marching Tetrahedra generally produces meshes of low quality, containing many sliver triangles. This is particularly visible with DMTet. We propose two regularization terms in our gradient-based mesh optimization pipelines to encourage fair triangulations.

Optimal Delaunay triangulations. We would like to encourage tetrahedra in the background grid to be uniform and well-conditioned, which should lead to better elements in the output triangular mesh. Mesh-quality metrics have been extensively studied in the FEM community [Shewchuk 2002]. Our tetrahedral regularizer adopts the Optimal Delaunay Triangulation (ODT) energy from this literature, which minimizes interpolation error across all possible triangulations for a fixed vertex set [Chen and Xu 2004]. This is well-suited to our approach because TetWeave uses Delaunay triangulation to construct its grid on-the-fly. Since ODT identifies the Delaunay connectivity as optimal for a given point set, it ensures consistency with our pipeline. Furthermore, Alliez et al. [2005] show that ODT generates well-shaped tetrahedra and derive a formulation that can be used in an optimization setting for vertex positions, aligning with our goal of producing fair, adaptive meshes without additional connectivity overhead. The ODT energy for a tetrahedron T_i is expressed as $E_{\text{ODT}}(T_i) = |M_{S_{T_i}} - M_{T_i}|$, where M_{T_i} represents the sum of the principal moments of T_i relative to its circumcenter. Similarly, $M_{S_{T_i}}$ denotes the moment of inertia of S_{T_i}, defined as the spherical

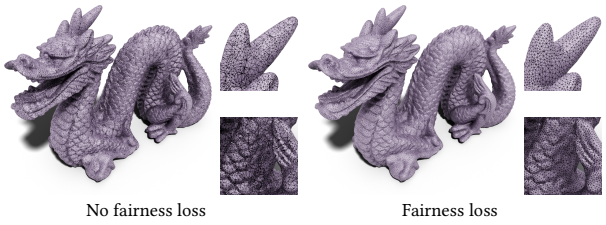


Fig. 6. Comparison of reconstructed meshes with and without the use of the fairness loss. We show that incorporating a simple fairness loss improves tessellation quality without compromising shape fidelity.

shell matching the circumsphere of T_i and having an equivalent mass. We define an ODT loss for the tetrahedral grid (P, T) as

$$\mathcal{L}_{\text{ODT}} = \sum_{T_i \in T} |M_{S_{T_i}} - M_{T_i}|.$$

Minimizing this energy reduces distortions in tetrahedral shapes, leading to more uniform and well-conditioned triangulations. We provide more information regarding implementation and show that this energy vanishes for a regular tetrahedron in Appendix B.

Triangle fairness loss. We want to encourage the formation of uniform triangles on the extracted mesh. Therefore, we encourage equilateral triangles by penalizing angles that deviate from $\frac{\pi}{3}$:

$$\mathcal{L}_{\text{fairness}} = \sum_{f \in F} \frac{1}{3} \sum_{i=1}^{3} \left(\theta_i - \frac{\pi}{3}\right)^2.$$

As shown in Fig. 6, this effectively encourages a fair tessellation.

Sign change regularizer. Similar to FlexiCubes [Shen et al. 2023], our method is prone to spurious geometry in unsupervised parts of the space. We adopt the same regularizer and penalize sign changes of the base sign distance value for every active edge:

$$\mathcal{L}_{\text{sign}} = \sum_{(a,b) \in E_A} H\left(\sigma(s_a), \text{sgn}(s_b)\right),$$

where E_A designates the set of active edges of the tetrahedral grid, H is the cross-entropy function, and σ the sigmoid function.

4.3 Point cloud refinement

Since TetWeave does not rely on a fixed grid, we can adaptively place points where they are necessary. Our approach for sampling focuses on resampling points that are not connected to any active points, referred to as *passive points* (see Fig. 7, top row), as they do not contribute to the optimization. In addition to resampling passive points, we incrementally introduce new points where they matter most during the optimization. We devise a sampling strategy illustrated in Fig. 7, bottom row. Given the current mesh (V, F), we compute a bounding box of the mesh, which is subdivided into a voxel grid $\mathcal{G}$. Each voxel $g_i \in \mathcal{G}$ is assigned an importance value $h(g_i)$, which we normalize over the voxel grid to get the probability distribution $\rho(g_i)$. Hereby, $\rho(\mathcal{G}) = \{\rho(g_i) | g_i \in \mathcal{G}\}$ defines a probability distribution over the grid $\mathcal{G}$.

Given K points to sample, we sample k_i, the number of points to sample in voxel g_i, from a multinomial distribution parameterized by $\rho(\mathcal{G})$ with K trials. We then randomly sample k_i points in the voxel g_i. For each sampled point, we determine its barycentric coordinates within the containing tetrahedron T and initialize its signed distance and spherical harmonics coefficients via barycentric interpolation

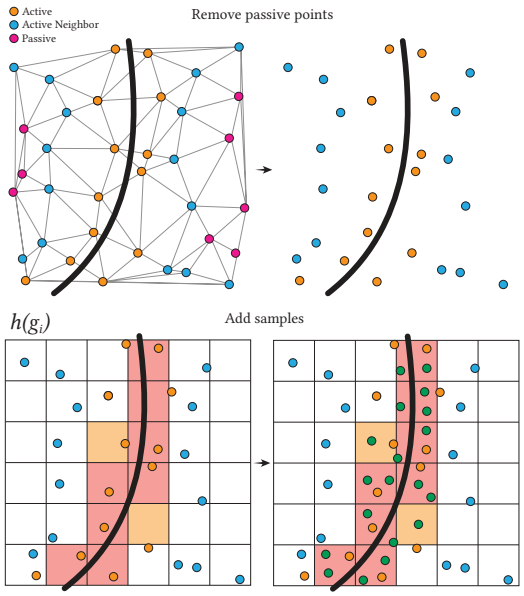


Fig. 7. Point cloud refinement is done in two steps. First, we identify and remove points that do not neighbor an active point (passive points). Next, we voxelize the space around the mesh and estimate h. Then we sample points within the voxels according to h. Finally, the sdf is interpolated to the new points based on a barycentric average of the containing tetrahedron.

of T's vertex parameters. As exemplified in Fig. 12, we can define h according to specific requirements. For example, uniform sampling is achieved by setting $h(g_i) = 1$ for voxels intersecting the mesh and $h(g_i) = 0$ otherwise. In the following section, we introduce a rendering-based method to estimate h. This enables our approach to dynamically adapt the mesh resolution.

Methods that use a predefined grid often use a constant level of detail for every region in space. This leads to significant memory inefficiency when reconstructing surfaces, which sparsely occupy 3D space. With a dense grid representation, much of the space remains unused, and the scaling of output vertices is suboptimal. For instance, in the case of FlexiCubes, doubling the grid resolution results in a cubic increase of the number of parameters, but only a quadratic increase in the number of output vertices. As a result, FlexiCubes scales poorly and requires using an octree-based datastructure to recover high-frequency details on meshes. DMTet can address this issue by subdividing tetrahedra close to the current mesh but lacks a systematic approach for doing so.

4.4 Adaptive meshing

We propose an importance function h, suitable for inverse rendering applications, such as multi-view 3D reconstruction, showcased in Fig. 8. The underlying strategy – to use error as a guide for sampling – can be applied outside of inverse rendering. We suppose that we have a current mesh (V, F) produced by TetWeave. Let $\mathcal{I}$ be a rasterizer that takes as input camera parameters θ_k for view k. We assume that target images $\tilde{I}_k$ are given, e.g., photographs or renders of an object we aim to reconstruct. For each pixel (u, v), we compute

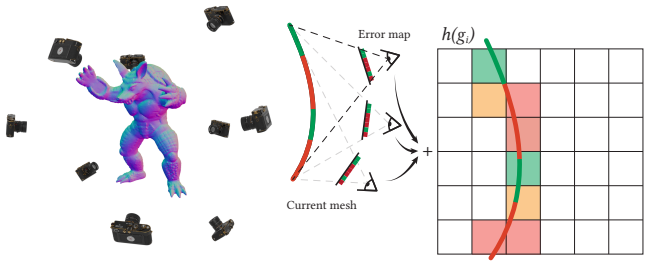


Fig. 8. Our refinement strategy can be applied to adaptive meshing. By rendering the shape from multiple viewpoints, we compute the error for each pixel. Since each visible pixel corresponds to a point on the shape's surface, we accumulate these errors within the voxel that contains the corresponding point. We normalize the accumulated errors across all voxels to define the importance value function h.

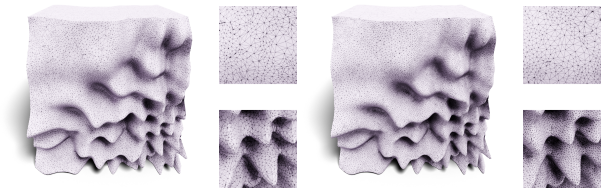


Fig. 9. Comparison between uniform and adaptive meshing. Resampling based on normal map error enables curvature-adaptive sampling, concentrating more triangles in high-curvature regions, which enhances reconstruction quality.

the error $E_k(u,v) = |\tilde{I}_k(u,v) - \mathcal{I}(V,F,\theta_k)(u,v)|$. Next, each pixel is projected onto the current mesh in $\mathbb{R}^3$ using rasterization. The error values for all pixels that land in voxel grid cell g_i are accumulated over all views and then normalized to compute $h(g_i)$

$$h(g_i) = \frac{\sum_k \sum_{(u,v)\in\mathcal{N}_{g_i}} E_k(u,v)}{|\mathcal{N}_{g_i}|}, \tag{4}$$

where $\mathcal{N}_{g_i}$ is the set of all pixels (u,v) that are contained in voxel cell g_i. Pixels that land outside the mesh are ignored. Next to resampling passive points, we progressively add points to our shape representation. Therefore, the resulting meshes adapt to the importance function h. Fig. 9 illustrates an example that demonstrates that denser tessellation occurs in high-frequency areas, enhancing reconstruction in those regions.

4.5 Multi-stage optimization

We propose a multi-stage optimization for TetWeave. The increased flexibility of our method and the higher grid resolution around the reconstructed surface results in more degrees of freedom. As reported by the authors of NVDiffRec (Hasselgren et al. [2022], section 8.5) and observed in our own experiments, these degrees of freedom could lead to noisy geometry when optimized naively. Another motivation to implement a multi-stage approach is to limit the number of Delaunay triangulation calls, which improves the overall speed.

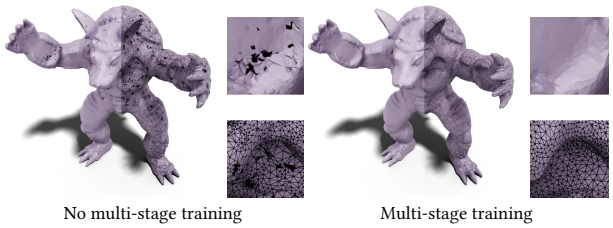


Fig. 10. Comparison between the resulting meshes with and without a multi-stage pipeline during optimization. Recomputing the Delaunay triangulation only periodically and fixing the grid to focus on SDF optimization in a second stage is necessary to avoid artifacts.

During the *main stage* (5000 iterations), we update both grid point positions and SDF values, and apply all regularizers. The number of grid points is incrementally increased via resampling until a target point number is reached. As the number of points grows, recomputing the Delaunay Triangulation becomes computationally expensive. To mitigate this, we only recompute the Delaunay Triangulation every m iterations by default, and keep point positions fixed between updates. We sum point positions' gradients over these m iterations before updating them, which ensures that the grid remains non-degenerate while incorporating information from multiple viewpoints, akin to gradient accumulation in machine learning pipelines.

We also set up a *late stage* (2000 iterations) which acts as a fine-tuning stage to ensure higher-fidelity results. Point positions are fixed, and the Delaunay triangulation is no longer recomputed. The optimization focuses solely on refining the signed distance values and spherical harmonics coefficients. We also do not use the ODT and triangle fairness regularizers. As shown in Fig. 10, the late stage incurs a significant boost in reconstruction quality.

5 Results

This section assesses the ability of our shape representation to reconstruct a mesh from unambiguous image input, as discussed in Sec. 4.1. In our experiments, the ground truth signed distance function is unavailable; the reconstruction is entirely inferred through an inverse rendering pipeline. We conclude this section with a detailed ablation over several key components of our optimization pipeline. Practical applications under real-world conditions are explored in Sec. 6.

5.1 Experimental setting

To conduct our experiments, we render a mask, depth map, and normal map of the target shape and the mesh generated by TetWeave using Nvdiffrast [Laine et al. 2020]. We employ the regularizers discussed in Sec. 4.2, the refinement from Sec. 4.3, use the L_1 loss over the normal maps as a rendering target for adaptive meshing from Sec. 4.4, and the multistage training method from Sec. 4.5. Additional details, including implementation specifics and parameter settings, are provided in Appendix A.

Our dataset comprises shapes from the ThreeDScan repository [Laric 2012], which offers high-quality meshes of up to more than 2 million vertices. This degree of detail demonstrates TetWeave's



Fig. 11. Visual comparison of different grid-adaptive isosurface mesh extraction methods, following the experimental setup described in Sec. 5: DMTet, FlexiCubes and TetWeave evaluated at low, mid and high resolutions. Our method achieves high-quality reconstruction across a wide variety of shapes, including highly detailed statues and objects with complex topologies, and excels in capturing intricate high-frequency details, such as the writing on the Gutenberg statue's socle. Furthermore, our approach provides fairer tessellation. We recommend zooming in on a digital display.

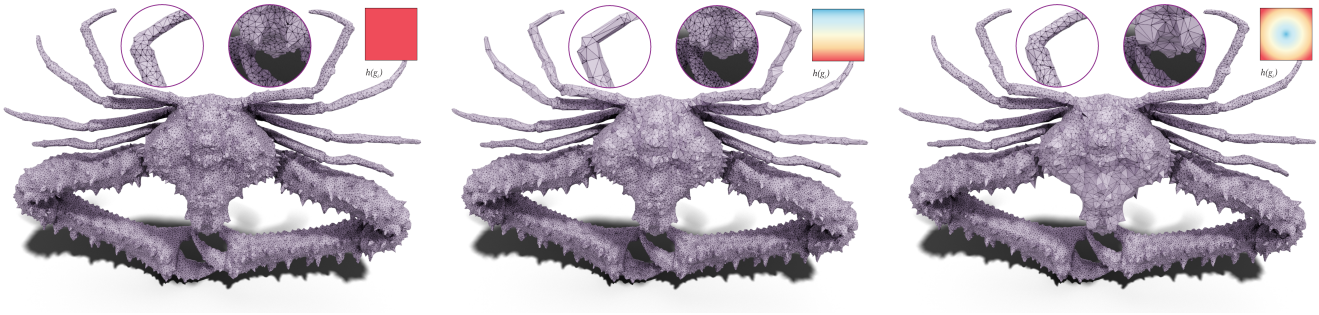
Table 2. Quantitative evaluation on the mesh reconstruction task. We sample 1 million points per shape in our dataset. We report the chamfer distance (CD, 1e-5), the F1 score, the edge chamfer distance (ECD, 1e-2), its F1 score (EF1), the normal consistency (NC), the percentage of inaccurate normals (IN> 5°), the percentage of triangles with aspect ratio (AR) and radius ratio (RR) above 4, the percentage of small angles (SA< 10°) and the percentage of self-intersecting faces (SI). After the generation, only the largest connected component is kept. An expanded table is provided in the appendix.

Method	CD ↓	F1 ↑	ECD ↓	EF1 ↑	NC ↑	IN> 5°(%) ↓	AR> 4(%) ↓	RR> 4(%) ↓	SA< 10°(%) ↓	SI(%) ↓	#V	#F
DMTet (128^3)	1.043	0.339	1.681	0.272	0.965	48.393	12.026	11.826	12.351	**0.000**	20677	41364
FlexiCubes (128^3)	0.752	0.416	1.254	0.393	0.979	36.911	5.418	6.701	4.588	0.203	28430	56873
TetWeave (16K)	0.517	0.409	1.475	0.353	0.974	43.380	2.350	3.344	1.743	**0.000**	26484	53015
TetWeave (64K)	0.419	0.446	0.962	0.518	0.984	33.700	**2.251**	**3.252**	**1.616**	**0.000**	81027	162102
TetWeave (128K)	**0.393**	**0.455**	**0.708**	**0.588**	**0.987**	**29.361**	2.507	3.556	1.829	**0.000**	146514	293074

ability to accurately reconstruct high-frequency details. Additionally, the dataset provides a testbed to highlight the potential of our approach for mesh compression applications, as detailed in Sec. 6.1. We preprocess the dataset by retaining only the largest connected component of each shape and by excluding non-watertight models. Each mesh is then normalized. Additionally, we remove redundant shapes, such as multiple versions of the same object. After processing, the dataset consists of 75 shapes.

5.2 Comparisons

We compare against the two most closely related methods, regarded as the state-of-the-art, DMTet and FlexiCubes. In our comparisons, we only extract the largest connected component from the resulting



$h : \mathbf{p} \to 1$
uniform

$h : \mathbf{p} \to |\mathbf{p}_y - 1|^3$
axis-wise increase

$h : \mathbf{p} \to \|\mathbf{p}\|^2$
radial

Fig. 12. Our resampling is guided by the importance value function h. In our experiments, we compute h based on normal map errors to enable adaptive sampling, but we can use any custom formulation. Here, we present three variations of h. Uniform tessellation is achieved by defining h as a constant value. By setting h as a cubic function along the y-axis, tessellation density increases progressively from top to bottom. Alternatively, using a radial function results in a higher triangle density in regions farther from the center of the mesh.

mesh for each method. This filters out internal geometry, present in each compared method. In the experiments by Shen et al. [2023], a different strategy is employed to solve the problem of internal geometry, however, the ground truth mesh itself is used as input. We found our filtering method to work comparably well, while being more widely applicable in realistic applications.

Quantitative evaluation. We follow the evaluation method from FlexiCubes, inspired by NDC [Chen et al. 2022], and report the average value of the following metrics: the chamfer distance (CD), its F1 score, the edge chamfer distance (ECD), its F1 score (EF1), the normal consistency (NC), the percentage of inaccurate normals (IN$> 5^\circ$(%)), the percentage of triangles whose aspect ratio (AR) and radius ratio (RR) is greater than 4, the percentage of angles below a threshold of $< 10^\circ$ (SA$< 10^\circ$), the percentage of self-intersecting triangles (SI), and the number of vertices and faces. The metrics are explained in more detail in Appendix C.1.

As shown in Table 2, our method consistently yields lower chamfer distances, higher normal consistency, and fewer degenerate triangles compared to both DMTet and FlexiCubes at comparable complexity levels. We also exhibit a near-monotonic improvement across all geometry metrics as the number of points increases from 16K to 128K, while producing no self-intersections at any resolution by design, as shown in Fig. 3. In particular, the lower percentages of angles under 10° and triangles with high aspect or radius ratios demonstrate the high geometric quality of our tessellation. These results indicate that TetWeave not only outperforms existing approaches in reconstructing high-frequency details, but also scales more effectively to larger resolutions.

Visual comparison. Visual comparisons in Fig. 11 feature DMTet and FlexiCubes at a 128^3 grid resolution alongside our method, shown at 16K, 64K, and 128K points in the underlying tetrahedral grid. Notably, FlexiCubes can hardly scale beyond 128^3 due to GPU memory constraints, while TetWeave can handle far higher resolutions. This difference proves critical for capturing fine details—such as the inscription on the Gutenberg statue's base, which FlexiCubes cannot reconstruct at its maximum grid size. Additionally, our meshing is more adaptive, using fewer, larger triangles on flat surfaces, and our overall tessellation tends to be fairer. However, FlexiCubes does achieve slightly sharper edges compared to ours at 16K grid points, consistent with its stronger ECD and EF1 scores at that resolution.

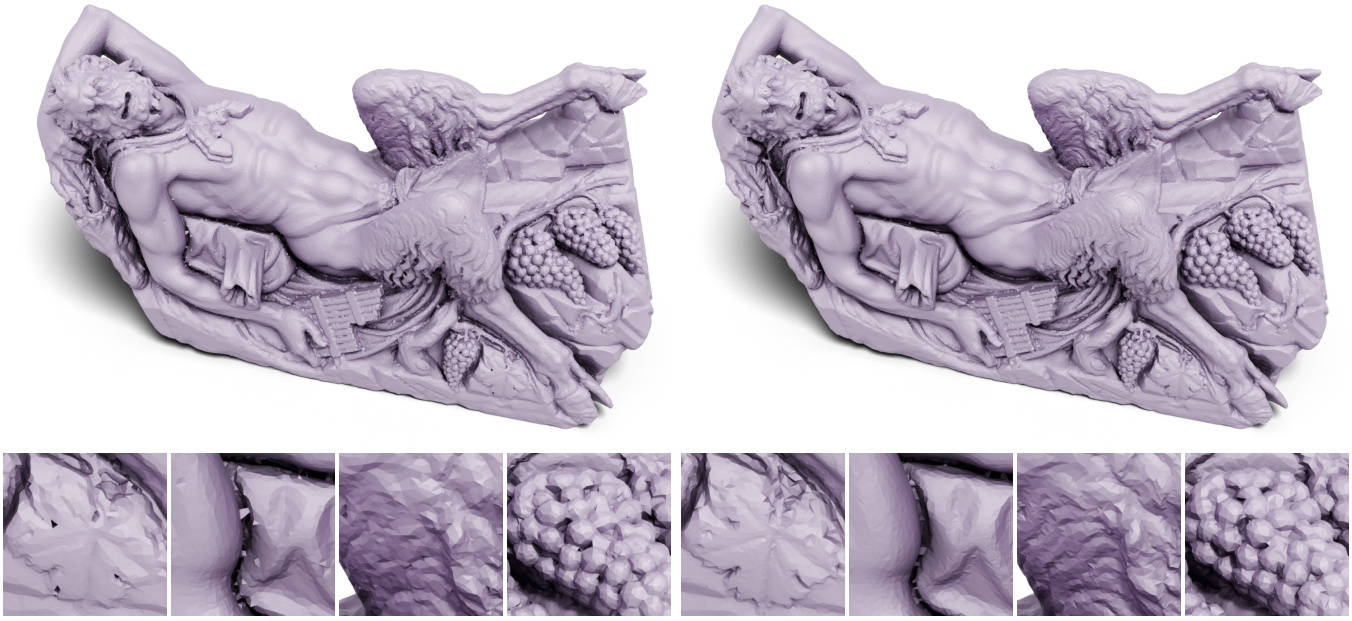
5.3 Ablation

In this section, we conduct ablation studies on isolated components of our technique to highlight their significance within the pipeline.

5.3.1 Directional signed distance. We examine the impact of incorporating spherical harmonics for computing directional signed distances in Table 5. While the use of spherical harmonics does not significantly affect the chamfer distance or F1-score, it consistently enhances the EF1-score and reduces the percentage of inaccurate normals. This improvement aligns with our observations: spherical harmonics enable directional adjustments of the signed distance, effectively aligning normals at a highly localized level. While this has minimal influence on chamfer distance, it proves beneficial and is visible in worst-case scenarios, as shown in Fig. 5.

5.3.2 Regularization. We evaluate the impact of regularizers in our method in the last three rows of Table 5. Introducing a fairness term substantially improves reconstruction, particularly in triangle quality metrics such as aspect ratio, radius ratio, and the reduction of small-angle percentages. This improvement explains the 36% reduction in vertex count, as the fairness term eliminates triangles with extremely small areas while preserving reconstruction quality. However, this comes at the cost of a slight increase in the percentage of inaccurate normals. Additionally, incorporating an ODT energy improves the edge chamfer distance by over 10%.

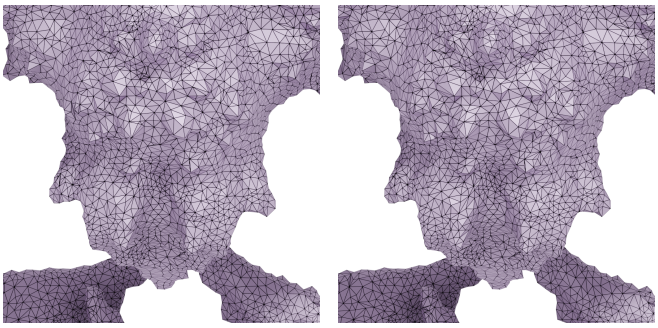
5.3.3 Adaptive meshing. We conducted experiments with TetWeave using our resampling strategy. As described in Sec. 4.3, our resampling relies on an importance value function h, which operates over a voxel grid decomposition of the current reconstructed shape. In the adaptive setting, h is computed based on rendering errors of the normal map. This approach concentrates points in regions where



No Spherical Harmonics
568kB – EF1: 0.568

Spherical Harmonics - degree 1
1.044MB – EF1: 0.617

Fig. 13. Compressed model obtained using a grid with 128K points. Incorporating spherical harmonics provides slight improvements at the edges as evidenced by the reported EF1 metric. However, it also doubles the memory requirements of the representation.



full point cloud - double precision
2.1MB

trimmed point cloud - single precision
587.3kB

Fig. 14. Our shape representation with 64K points and spherical harmonics of degree 1 occupies 2.1MB when stored in double precision. By discarding non-active points and using single precision storage, the memory footprint can be reduced by nearly a factor of four, with minimal changes that remain imperceptible, as shown in this close-up.

normals misalign, typically areas of high curvature like highly detailed regions.

In this ablation, we test alternatives of h. By setting h to a constant value and 0 in voxels that do not intersect the shape, we uniformly sample points within each voxel intersecting the mesh. This configuration is referred to as "Uniform" in Table 5. Compared to adaptive sampling, uniform sampling achieves similar chamfer distance (CD) and F1-score, demonstrating its ability to produce meshes of comparable quality. However, for sharp features, uniform sampling underperforms, as evidenced by significantly worse ECD, EF1, and percentages of inaccurate normals. This highlights the advantage of adaptive sampling in targeting high-frequency details. On the other hand, uniform sampling marginally improves triangle quality, with less than 1% of triangles having an angle smaller than 10°.

To demonstrate the flexibility of our importance value function h, we illustrate three different configurations for h in Fig. 12: $h : \mathbf{p} \to 1$ corresponds to a uniform meshing strategy, where all regions are sampled equally; $h : \mathbf{p} \to |\mathbf{p}_y - 1|^3$ introduces an axis-wise variation which increases the density of triangles progressively along the y-axis from top to bottom; and $h : \mathbf{p} \to |\mathbf{p}|^2$ concentrates triangles in regions farther from the center. These examples showcase how h can be tailored to prioritize specific regions of interest, resulting in meshes that emphasize different geometric features of the shape.

6 Applications

This section provides examples of gradient-based mesh optimization applications using our method, namely mesh compression, geometric texture generation and photogrammetry.

6.1 Mesh compression

One key advantage of our representation is its relatively low memory footprint, because we do not store connectivity. Instead, we infer it using Delaunay Triangulation and a single point can generate multiple vertices in the final mesh. For N points, the size of our representation amounts to $B(4+q)N$ bytes, where q is the number

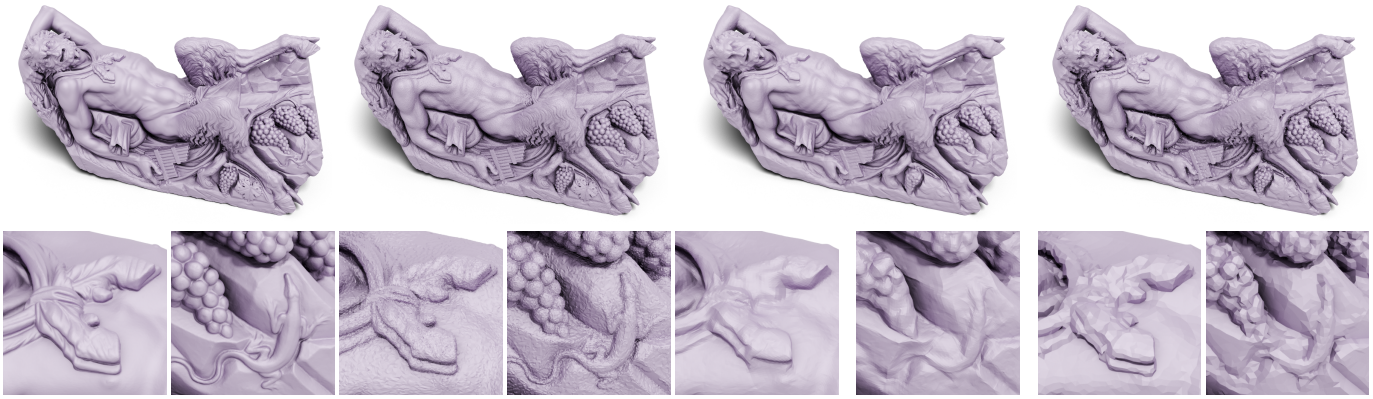


Fig. 15. Comparison of our representation against two mesh-compression techniques, Draco [Google 2017] and Neural Geometry Fields (NGF) [Edavamadathil Sivaram et al. 2024], on a high-resolution statue. The reference mesh occupies 37.4MB with single precision. Draco compresses it to 2.3MB by quantizing vertex connectivity, but suffers from high-frequency artifacts. NGF requires only 267.3kB but produces non-adaptive patches, can over-smoothen the result and self-intersect. Our approach achieves a 319.8kB file size by discarding non-active points, using 16-bit floats, and inferring connectivity from Delaunay triangulation.

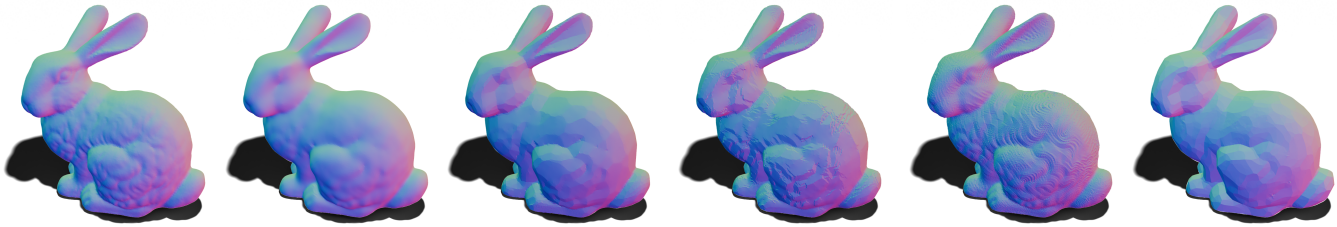


Fig. 16. We showcase how gradient-based optimization allows for creating geometric textures by fine-tuning high-frequency details. The reference is obtained by optimizing our representation of the Stanford bunny. Using gradient-based optimization, we can create geometric textures by applying a style to high-frequency details of a mesh. Using Paparazzi's approach [Liu et al. 2018], we can use any image filter for stylization. We display the stylization using Guided filter, L_0-smoothing, and K-means color quantization. Our method also allows us to backpropagate gradients from any energy function. We apply a CLIP loss on the rendered normal maps with the prompt "a wave-like style", and a developability energy [Stein et al. 2018] directly on the reconstructed mesh, yielding piecewise developable surfaces.

of spherical harmonic coefficients and $B = 4$ or $B = 8$, depending on whether we use single or double precision. Given that spherical harmonics of degree 1 introduce three additional coefficients per point, opting not to use them can maximize compression efficiency with only minor visual degradation, as evidenced in the compression of Fig. 13. After fitting our representation to a given shape, to maximize the compression rate, we discard the non-active vertices of the tetrahedral grid, *i.e.*, vertices that are not connected to at least one vertex whose base SDF value has an opposite sign. While this operation is not strictly lossless, the difference is almost imperceptible, as shown in Fig. 14. We compare TetWeave in Fig. 15 with Neural Geometry Fields (NGF) [Edavamadathil Sivaram et al. 2024], an appearance-based compression method, and Draco [Google 2017], an industry-standard scheme that reduces bit rates in vertex connectivity. NGF compresses geometry more aggressively but exhibits notable drawbacks compared to our method: its patch-based tessellation can be unfair because patches vary in size, it is not adaptive, and it has no mechanism to prevent self-intersections. Although Draco is fast, it only focuses on quantizing geometry and is prone to artifacts, so highly tessellated meshes can still consume high memory in applications that do not require such dense sampling.

6.2 Geometric texture

Gradient-based mesh optimization can be used for geometric texture generation. Adopting a similar pipeline to Paparazzi [Liu et al. 2018], we render normal maps of our generated mesh, $\tilde{I}$, apply an image filter f, and substitute the gradient with $f(\tilde{I}) - \tilde{I}$ to update the parameters of TetWeave. In practice, we first fit our representation to a reference shape using spherical harmonics of degree 2. Once the fit is complete, we fix the point positions and SDF values. Using



Fig. 17. We demonstrate the applicability of our method to photogrammetry by integrating it into the NVDiffRec pipeline [Munkberg et al. 2022]. NVDiffRec uses differentiable rasterization to jointly optimize PBR material properties, the environment map, and geometry to match a target rendered shape. In our approach, we employ our method as the geometry representation in a multi-stage process. We initialize a blue-noise point cloud and jointly optimize the point positions along with their SDF values, compute the tetrahedral grid on the fly, and reconstruct a coarse shape. Second, we fit our representation to the coarse mesh, resulting in a point cloud that closely matches the reconstructed shape. Finally, we refine high-frequency details by optimizing only the SDF values in the last stage.

a chosen filter, we render the shape's normal map from a randomly sampled camera view, apply the filter, and compute gradients to backpropagate, updating only the spherical harmonic coefficients. Since geometric textures primarily affect high-frequency details, adjusting spherical harmonics is sufficient to tilt the normals into the desired orientation without deviating significantly from the reference shape. Additionally, we use a loss on the mask, depth map, and normal map, compared to the reference mesh, to ensure that the generated shape remains close to the reference and does not degenerate. To avoid being constrained too much by the initial reference, we periodically update the reference shape with the current mesh during optimization. We demonstrate results using various image filters in Fig. 16: namely Guided filter [He et al. 2013], L_0-Smoothing [Xu et al. 2011], and K-means image quantization [MacQueen 1967].

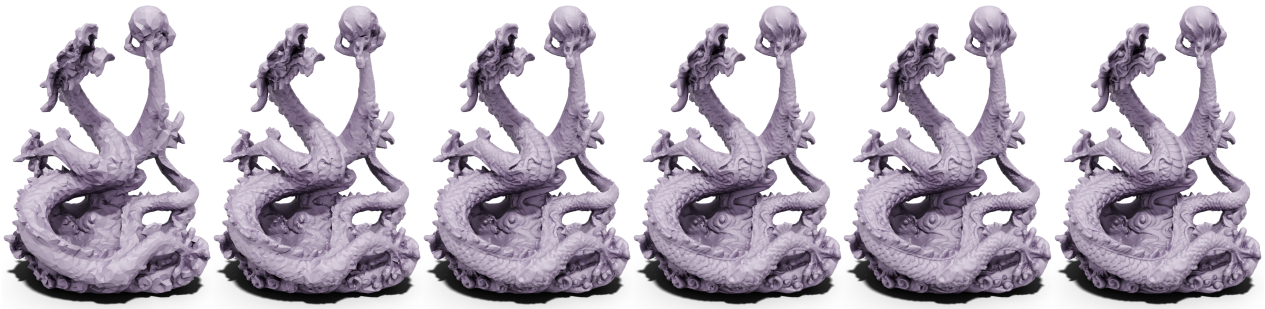
TetWeave can be adapted to any gradient-based mesh optimization pipelines. Therefore, Fig. 16 also showcases examples where the gradients are given by a user-defined rendering or mesh energy. From this energy, we can use PyTorch's automatic differentiation to update TetWeave's parameters and match the target energy. For instance, we demonstrate how we can apply a CLIP loss [Radford et al. 2021] on the normal map to produce a wave-like effect. Similar to FlexiCubes, our approach supports a developability energy [Stein et al. 2018] for crafting piecewise developable surfaces, though both methods tend to yield a lot of patches.

6.3 Photogrammetry

Photogrammetry involves recovering scene parameters from a set of photographs. NVDiffRec [Munkberg et al. 2022] introduces a differentiable rendering framework that jointly reconstructs PBR materials, environment lighting, and geometry. The original implementation employs DMTet for its reconstruction pipeline and the authors provide an implementation of the framework with FlexiCubes. We evaluate each method on the recent benchmark dataset, Stanford ORB [Kuang et al. 2023].

As noted in Sec. 4.5, NVDiffRec can output noisy geometry when the reconstruction problem is underconstrained. Munkberg et al. [2022] addressed similar issues in DMTet by regularizing SDF values through an MLP, by truncating the positional encoding to lower the frequency. TetWeave has more degrees of freedom and optimizes the geometry and the background grid concurrently. This is a strength when it comes to expressivity, but can be challenging when optimizing a highly ambiguous inversion problem, where lighting, geometry and materials are unknown. Thus, rather than using our method as a direct replacement for DMTet [Shen et al. 2021], we employ a multi-stage approach. We initialize TetWeave's point cloud using a precomputed blue noise sampling and construct a tetrahedral grid via Delaunay triangulation. This serves as the foundation for our method during the main stage, where we update point positions and SDF values to extract a coarse mesh. Next, we fit an adaptive grid of 8K points around the coarse mesh using our standard mesh fitting algorithm described in section 5. In the late stage, we focus on refining the adaptive point cloud by exclusively optimizing the SDF values. This stage focuses on capturing fine-grained details and eliminating artifacts introduced in earlier stages. The gentle introduction of the optimization of the background grid makes it easier to converge to a correct solution. We observe that increasing the number of points to achieve more detailed meshes can lead to artifacts. To reduce additional degrees of freedom, we avoid using spherical harmonics for this application.

The results for the photogrammetry application of TetWeave are shown in Fig. 17 and the quantitative metrics are displayed in Table 6, following the benchmark established by Stanford ORB [Kuang et al. 2023]. While the full benchmark is presented, our discussion focuses on grid-adaptive isosurface extraction techniques, as these are most comparable to our method and were also used with NVDiffRec. Other methods improve on the rendering model and methods to estimate lighting and materials. In our experiments, we find that TetWeave achieves better quality regarding novel lighting- and novel view synthesis. We are also able to retrieve better triangulations and higher resolution meshes. However, we observe that our method does not surpass FlexiCubes and DMTet in geometry metrics. This can be attributed to the fact that competing methods produce lower-resolution meshes that are less susceptible to high-frequency artifacts. Additionally, their grid structures inherently act as regularizers, whereas the high adaptivity of our point cloud prioritizes local geometric detail.



grid points	8K / 6.6K	16K / 13.2K	32K / 24.4K	64K / 43.0K	128K / 76.6K	input
vertices	15K	29K	53K	91K	163K	450K
runtime	137s	157s	181s	276s	456s	
memory	108kB	213kB	393kB	690kB	1.2MB	15.5 MB

Fig. 18. We present the shape utilized in our performance experiments at varying grid point densities. The table shows the target number of grid points used during the optimization and the final number of active points that remained. We also provide details of the final mesh resolution and the practical runtime required to generate each shape. Additionally, we provide the file size of the model, where only active points remained and are stored in float16 format. We used SH with degree 1. The shape showcased here is sourced from Thingi10K [Zhou and Jacobson 2016].

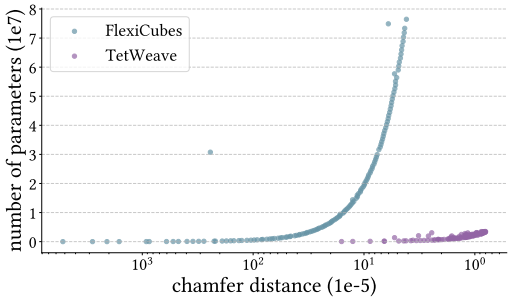


Fig. 19. Relationship between the achieved chamfer distance and the number of parameters required for FlexiCubes and TetWeave. The graph demonstrates that FlexiCubes does not scale as effectively as TetWeave. The maximum grid size for FlexiCubes represented in this figure is 145^3, while TetWeave is shown utilizing up to 500K points. These experiments were conducted on the shape shown in Fig. 18.

7 Discussion

This section assesses TetWeave's performance. We analyze its memory efficiency and runtime characteristics, and examine its limitations to provide a balanced perspective. We conclude by exploring future research directions to further advance unstructured mesh representations.

7.1 Performance

We discuss the performance of our method, focusing on comparisons with FlexiCubes [Shen et al. 2023] in terms of both memory usage and computational speed. Experiments were conducted on an NVIDIA RTX 3090 GPU.

Memory. Our shape representation is particularly memory-efficient. This is due to several reasons: we store only the points, their SDF values, and, optionally, spherical harmonics coefficients. In contrast, FlexiCubes [Shen et al. 2023] requires storing SDF values, deformation parameters per point, and 21 coefficients per voxel. Moreover, our method avoids using a fixed grid and instead adaptively resamples passive points around the shape. This allows each point to contribute to a higher number of output triangles, significantly reducing the number of points needed to achieve a desired level of detail and Chamfer Distance. As illustrated in Fig. 19, our approach achieves better Chamfer Distance results with far fewer points compared to FlexiCubes, translating to significantly lower memory requirements. The graph demonstrates that TetWeave scales more efficiently than FlexiCubes, allowing us to reconstruct shapes at a higher resolution.

Runtime. Our memory efficiency comes with the trade-off of increased runtime. In Table 3, we present the average runtime for both FlexiCubes and TetWeave, measured on the shape shown in Fig. 18. For our method, most of the time in the forward pass is spent on Delaunay triangulation. Although the theoretical time complexity of Delaunay triangulation is $O(n \ln(n))$, this reflects the

Table 3. Quantitative comparison of runtime performance between TetWeave and FlexiCubes [Shen et al. 2023]. The forward pass of TetWeave is divided into two components: the Delaunay triangulation step for generating the tetrahedral grid [Si 2015] and our implementation of the Marching Tetrahedra algorithm [Doi and Koide 1991], adapted to incorporate spherical harmonics coefficients (of degree 1 in this experiment).

Method		Forward Time (ms)		Backward Time (ms)
FlexiCubes	32^3	5.88		3.58
	64^3	6.66		5.07
	128^3	9.63		15.25
		Delaunay Triangulation	Marching Tetrahedra	
TetWeave	8K	22.66	1.48	5.25
	16K	40.25	1.68	6.11
	32K	76.77	2.04	8.54
	64K	162.24	2.94	15.72
	128K	323.64	4.52	29.88
	256K	658.95	8.09	57.34

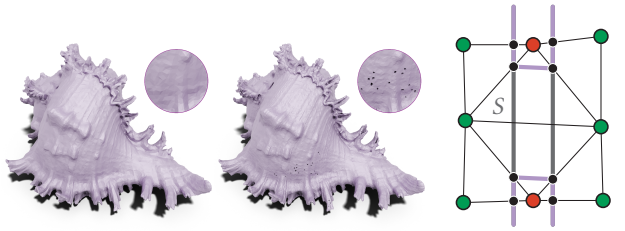


Fig. 20. We evaluate our method against FlexiCubes on a shape featuring a particularly thin structure. In such cases, the Delaunay triangulation can sometimes link two points with similar signed distance function values, resulting in an inaccurate hole that deviates from the target geometry S.

worst-case scenario and, in practice, the runtime of Tetgen [Si 2015] appears to scale linearly. Similarly, our backward pass runtime also exhibits linear scaling, which can be attributed to the linear increase in the number of vertices generated by our approach. In practice, the runtime does not exhibit strict linear scaling due to several factors. First, the number of points is progressively increased throughout the optimization process. All point clouds in Fig. 18 were initialized with 8K points, which were gradually increased to the target number through our resampling strategy. Second, the tetrahedral grid is not recomputed at every step; instead, gradients are accumulated over several iterations. Point positions and the tetrahedral grid are only updated every five steps. Lastly, during late-stage refinement, point positions and the triangulation remain fixed, significantly reducing the runtime for the final optimization steps. As a result, the runtime can vary from two minutes at 8K points to less than eight minutes at 128K.

7.2 Limitations

Isosurface extraction methods are generally unsuitable for handling thin structures, and our approach is particularly sensitive to such cases. As illustrated in Fig. 20, when dealing with meshes containing thin structures, the Delaunay triangulation may connect points with the same sign, resulting in holes in the final shape.

7.3 Conclusion

This paper presents TetWeave, a scalable isosurface representation for gradient-based mesh optimization that constructs tetrahedral grids on the fly via Delaunay triangulation. Our method produces watertight, manifold, and intersection-free meshes, excelling at handling complex shapes with lower memory overhead. However, several challenges remain. While the final mesh resolution grows nearly linearly with the number of points, Delaunay triangulation can exhibit superlinear runtime; although this did not pose problems in our experiments, it implies that time complexity may become the limiting factor before memory. Furthermore, like other grid-adaptive approaches, internal cavities can arise unless regularization is applied, and thin structures remain problematic, to which our method is especially sensitive.

We believe that unstructured representations for mesh-based pipelines still have many opportunities for development. Since Marching Tetrahedra only requires consistently oriented tetrahedral grids, devising a technique to infer non-Delaunay connectivity on the fly could offer additional flexibility. Another promising avenue is a direct pipeline to convert arbitrary meshes into our representation, which could immediately enable near-lossless compression and provide a convenient parameter space for learning-based tasks—particularly valuable for generative models. Determining the feasibility and practical implementation of such a pipeline remains an open challenge.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback and Marcel Padilla for his careful reading of our manuscript. The open source codebases of FlexiCubes and DMTet have been instrumental in the development of TetWeave. This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101003104, ERC CoG MYCLOTH).

References

Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. 2005. Variational tetrahedral meshing. *ACM Trans. Graph.* 24, 3 (July 2005), 617–625. doi:10.1145/1073204.1073238

Jürgen Bey. 1995. Tetrahedral grid refinement. *Computing* 55 (1995), 355–378. https://api.semanticscholar.org/CorpusID:20829446

Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. 2021a. NeRD: Neural Reflectance Decomposition from Image Collections. In *IEEE International Conference on Computer Vision (ICCV)*.

Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan T. Barron, and Hendrik P.A. Lensch. 2021b. Neural-PIL: Neural Pre-Integrated Lighting for Reflectance Decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Long Chen and Jin-chao Xu. 2004. OPTIMAL DELAUNAY TRIANGULATIONS. *Journal of Computational Mathematics* 22, 2 (2004), 299–308. http://www.jstor.org/stable/43693155

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. 2022. Neural Dual Contouring. *ACM Transactions on Graphics (Special Issue of SIGGRAPH)* 41, 4 (2022).

Zhiqin Chen and Hao Zhang. 2021. Neural Marching Cubes. *ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia)* 40, 6 (2021).

B. R. de Araújo, Daniel S. Lopes, Pauline Jepp, Joaquim A. Jorge, and Brian Wyvill. 2015. A Survey on Implicit Surface Polygonization. *ACM Comput. Surv.* 47, 4, Article 60 (may 2015), 39 pages. doi:10.1145/2732197

Boris Delaunay. 1934. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles* 7 (1934), 793–800.

Tamal K. Dey and Joshua A. Levine. 2007. Delaunay Meshing of Isosurfaces. In *IEEE International Conference on Shape Modeling and Applications 2007 (SMI '07)*. 241–250. doi:10.1109/SMI.2007.15

Akio Doi and A. Koide. 1991. An Efficient Method of Triangulating Equivalued Surfaces by using Tetrahedral Cells. *IEICE Transactions on Information and Systems* 74 (01 1991).

Venkataram Edavamadathil Sivaram, Tzu-Mao Li, and Ravi Ramamoorthi. 2024. Neural Geometry Fields For Meshes. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 29, 11 pages. doi:10.1145/3641519.3657399

Clement Fuji Tsang, Maria Shugrina, Jean Francois Lafleche, Towaki Takikawa, Jiehan Wang, Charles Loop, Wenzheng Chen, Krishna Murthy Jatavallabhula, Edward Smith, Artem Rozantsev, Or Perel, Tianchang Shen, Jun Gao, Sanja Fidler, Gavriel State, Jason Gorski, Tommy Xiang, Jianing Li, Michael Li, and Rev Lebaredian. 2022. Kaolin: A Pytorch Library for Accelerating 3D Deep Learning Research. https://github.com/NVIDIAGameWorks/kaolin.

Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2020. Learning Deformable Tetrahedral Meshes for 3D Reconstruction. In *Advances In Neural Information Processing Systems*.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. 2022. GET3D: A Generative Model of High Quality 3D Textured Shapes Learned from Images. In *Advances In Neural Information Processing Systems*.

Georgia Gkioxari, Jitendra Malik, and Justin Johnson. 2019. Mesh R-CNN.

Google. 2017. Draco: 3D Data Compression. https://github.com/google/draco. Accessed: 2024-12-28.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation.

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: a self-prior for deformable meshes. *ACM Trans. Graph.* 39, 4, Article 126 (aug 2020), 12 pages. doi:10.1145/3386569.3392415

Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. 2022. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. 35 (2022), 22856–22869. https://proceedings.neurips.cc/paper_files/paper/2022/file/8fcb27984bf16ca03cad643244ec470d-Paper-Conference.pdf

Kaiming He, Jian Sun, and Xiaoou Tang. 2013. Guided Image Filtering. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 6 (June 2013), 1397–1409. doi:10.1109/TPAMI.2012.213

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep Geometric Texture Synthesis. *ACM Trans. Graph.* 39, 4, Article 108 (2020). doi:10.1145/3386569.3392471

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2021. SAPE: Spatially-Adaptive Progressive Encoding for Neural Optimization. In *Proc. NeurIPS*.

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2022. SPAGHETTI: Editing Implicit Shapes Through Part Aware Generation. *ACM Transactions on Graphics* 41, 4 (2022), 106:1–20.

A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. 1996. Marching triangles: range image fusion for complex object modelling. In *Proceedings of 3rd IEEE International Conference on Image Processing*, Vol. 2. 381–384 vol.2. doi:10.1109/ICIP.1996.560840

K. C. Hui and Z. H. Jiang. 1999. Tetrahedra Based Adaptive Polygonization of Implicit Surface Patches. *Computer Graphics Forum* 18, 1 (1999), 57–68. doi:10.1111/1467-8659.00302

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. *ACM Trans. Graph.* 21, 3 (jul 2002), 339–346. doi:10.1145/566654.566586

Yiwen Ju, Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. 2024. Adaptive grid generation for discretizing implicit complexes. *ACM Trans. Graph.* 43, 4, Article 82 (July 2024), 17 pages. doi:10.1145/3658215

Dae-Hyun Kim, Ulf Doring, and Beat Bruderlin. 2000. Polygonization of Non-manifolds With the Aid of Interval Operators. https://api.semanticscholar.org/CorpusID:14381080

Zhengfei Kuang, Yunzhi Zhang, Hong-Xing Yu, Samir Agarwala, Elliott Wu, Jiajun Wu, et al. 2023. Stanford-ORB: a real-world 3D object inverse rendering benchmark. *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Transactions on Graphics* 39, 6 (2020).

Olivier Laric. 2012. Three D Scans: Free Downloadable 3D Scans. https://threedscans.com/. Scans based on collections from Albertina, Vienna; Kunsthistorisches Museum, Vienna; Musée Guimet, Paris; and other institutions. Supported by Lafayette Anticipation, Secession, and others. No copyright restrictions. Accessed: 2024-12-08.

Yiyi Liao, Simon Donné, and Andreas Geiger. 2018. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Anwei Liu and Barry Joe. 1995. Quality Local Refinement of Tetrahedral Meshes Based on Bisection. *SIAM Journal on Scientific Computing* 16, 6 (1995), 1269–1291. doi:10.1137/0916074

Hsueh-Ti Derek Liu, Michael Tao, and Alec Jacobson. 2018. Paparazzi: Surface Editing by way of Multi-View Image Processing. *ACM Transactions on Graphics* (2018).

Minghua Liu, Xiaoshuai Zhang, and Hao Su. 2020. Meshing Point Clouds with Predicted Intrinsic-Extrinsic Ratio Guidance. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII* (Glasgow, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 68–84. doi:10.1007/978-3-030-58598-3_5

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (aug 1987), 163–169. doi:10.1145/37402.37422

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. https://openreview.net/forum?id=Bkg6RiCqY7

Yiming Luo, Zhenxing Mi, and Wenbing Tao. 2021. DeepDT: Learning Geometry From Delaunay Triangulation for Surface Reconstruction. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2277–2285. doi:10.1609/AAAI.V35I3.16327

J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967).

Nissim Maruani, Roman Klokov, Maks Ovsjanikov, Pierre Alliez, and Mathieu Desbrun. 2023. VoroMesh: Learning Watertight Surface Meshes with Voronoi Diagrams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 14565–14574.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space . 4455-4465 pages. doi:10.1109/CVPR.2019.00459

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. doi:10.1145/3528223.3530127

Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. 2022. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8280–8290.

Alessandro Muntoni and Paolo Cignoni. 2021. *PyMeshLab*. doi:10.5281/zenodo.4438750

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large Steps in Inverse Rendering of Geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40, 6 (Dec. 2021). doi:10.1145/3478513.3480501

G.M. Nielson. 2004. Dual marching cubes. In *IEEE Visualization 2004*. 489–496. doi:10.1109/VISUAL.2004.28

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG] https://arxiv.org/abs/1912.01703

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional Occupancy Networks. In *European Conference on Computer Vision (ECCV)*.

Dmitry Petrov, Pradyumn Goyal, Vikas Thamizharasan, Vladimir Kim, Matheus Gadelha, Melinos Averkiou, Siddhartha Chaudhuri, and Evangelos Kalogerakis. 2024. GEM3D: GEnerative Medial Abstractions for 3D Shape Synthesis. 11 pages. doi:10.1145/3641519.3657415

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-to-3D using 2D Diffusion. *arXiv* (2022).

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8748–8763. https://proceedings.mlr.press/v139/radford21a.html

Marie-Julie Rakotosaona, Paul Guerrero, Noam Aigerman, Niloy J. Mitra, and Maks Ovsjanikov. 2021. Learning Delaunay Surface Elements for Mesh Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 22–31.

Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. 2020. MeshSDF: Differentiable Iso-Surface Extraction. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 22468–22478. https://proceedings.neurips.cc/paper/2020/file/fe40fb944ee700392ed51bfe84dd4e3d-Paper.pdf

Daxuan Ren, Hezi Shi, Jianmin Zheng, and Jianfei Cai. 2025. McGrids: Monte Carlo-Driven Adaptive Grids for Iso-Surface Extraction. In *Computer Vision – ECCV 2024*, Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (Eds.). Springer Nature Switzerland, Cham, 127–144.

Scott Schaefer, Tao Ju, and Joe Warren. 2007. Manifold Dual Contouring. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (May 2007), 610–619. doi:10.1109/TVCG.2007.1012

Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. Article 73, 11 pages.

Silvia Sellán, Yingying Ren, Christopher Batty, and Oded Stein. 2024. Reach For the Arcs: Reconstructing Surfaces from SDFs via Tangent Points. In *SIGGRAPH 2024 Conference Papers*. Article 25, 11 pages.

Nicholas Sharp and Maks Ovsjanikov. 2020. "PointTriNet: Learned Triangulation of 3D Point Sets". In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. 2024. SpaceMesh: A Continuous Representation for Learning Manifold Surface Meshes. In *SIGGRAPH Asia 2024 Conference Papers (SA*

Conference Papers '24) (Tokyo, Japan, December 3-6, 2024). ACM, New York, NY, USA, 11. doi:10.1145/3680528.3687634

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM Trans. Graph.* 42, 4, Article 37 (jul 2023), 16 pages. doi:10.1145/3592430

Jonathan Richard Shewchuk. 2002. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *11th International Meshing Roundtable, {IMR} 2002*. Ithaca, United States. https://hal.science/hal-04614934

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2, Article 11 (Feb. 2015), 36 pages. doi:10.1145/2629697

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proc. NeurIPS*.

Oded Stein, Eitan Grinspun, and Keenan Crane. 2018. Developability of triangle meshes. *ACM Trans. Graph.* 37, 4, Article 77 (July 2018), 14 pages. doi:10.1145/3197517.3201303

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *NeurIPS* (2020).

F. Tonon. 2005. Explicit Exact Formulas for the 3-D Tetrahedron Inertia Tensor in Terms of its Vertex Coordinates. *Journal of Mathematics and Statistics* 1 (Mar 2005), 8–11. doi:10.3844/jmssp.2005.8.11

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV*.

Eric W. Weisstein. 2025. Circumsphere. https://mathworld.wolfram.com/Circumsphere.html From MathWorld–A Wolfram Web Resource.

Mark A. Wieczorek and Matthias Meschede. 2018. SHTools: Tools for Working with Spherical Harmonics. *Geochemistry, Geophysics, Geosystems* 19, 8 (2018), 2574–2592. doi:10.1029/2018GC007529

Haoqian Wu, Zhipeng Hu, Lincheng Li, Yongqiang Zhang, Changjie Fan, and Xin Yu. 2023. NeFII: Inverse Rendering for Reflectance Decomposition With Near-Field Indirect Illumination. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4295–4304.

Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. 2011. Image smoothing via L0 gradient minimization. *ACM Trans. Graph.* 30, 6 (Dec. 2011), 1–12. doi:10.1145/2070781.2024208

Dingdong Yang, Yizhi Wang, Konrad Schindler, Ali Mahdavi Amiri, and Hao Zhang. 2024. GALA: Geometry-Aware Local Adaptive Grids for Detailed 3D Generation. arXiv:2410.10037 [cs.CV] https://arxiv.org/abs/2410.10037

Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. 2020. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. *Advances in Neural Information Processing Systems* 33 (2020).

Lior Yariv, Omri Puny, Oran Gafni, and Yaron Lipman. 2024. Mosaic-SDF for 3D Generative Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4630–4639.

Wang Yifan, Lukas Rahmann, and Olga Sorkine-hornung. 2022. Geometry-Consistent Neural Shape Representation with Implicit Displacement Fields. In *International Conference on Learning Representations*. https://openreview.net/forum?id=yhCp5RcZD7

Biao Zhang, Matthias Nießner, and Peter Wonka. 2022. 3DILG: Irregular Latent Grids for 3D Generative Modeling. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=RO0wSr3R7y-

Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 2023a. 3DShape2VecSet: A 3D Shape Representation for Neural Fields and Generative Diffusion Models. *ACM Trans. Graph.* 42, 4, Article 92 (jul 2023), 16 pages. doi:10.1145/3592442

Chen Zhang, Ganzhangqin Yuan, and Wenbing Tao. 2023b. DMNet: Delaunay Meshing Network for 3D Shape Representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 14418–14428.

Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. 2021a. PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. 2021b. NeRFactor: Neural Factorization of Shape and Reflectance under an Unknown Illumination. *ACM Trans. Graph.* 40, 6, Article 237 (dec 2021), 18 pages. doi:10.1145/3478513.3480496

Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. 2021. Progressive Discrete Domains for Implicit Surface Reconstruction. *Computer Graphics Forum* 40, 5 (2021), 143–156. doi:10.1111/cgf.14363

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR]

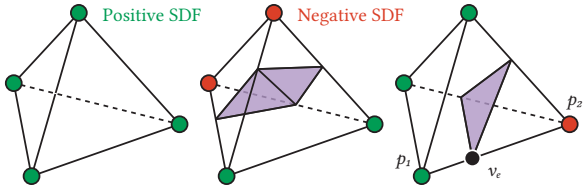


Fig. 21. Marching Tetrahedra lookup configurations. Despite 2^4 possible configurations, they amount to three distinct possibilities up to rotation and sign inversion displayed here. Contrary to Marching Cubes, there is no ambiguous case, and the resulting mesh is guaranteed intersection-free and 2-manifold.

A Implementation Details

In this section, we detail the implementation of our shape reconstruction pipeline, as described in Sec. 5. Differentiable rasterization is performed using nvdiffrast [Laine et al. 2020], and many geometric operations are used from the Kaolin library [Fuji Tsang et al. 2022].

Initialization. Before starting the reconstruction process, the shapes in our dataset are normalized such that their bounding boxes fit within the cube $[-0.9, 0.9]^3$, similar to the preprocessing used in FlexiCubes [Shen et al. 2023]. For grid initialization, we randomly sample 8000 points within a sphere of radius $\sqrt{3}$. Spherical harmonics coefficients are initialized to 0, which leads to an initial state where $\hat{s}_i(e) = s_i$ due to the application of the $1 + tanh$ function. Thanks to our iterative resampling approach, TetWeave is robust to the resolution of the initial sampling, as long as it adequately covers the target shape. In contrast, methods like FlexiCubes and DMTet [Shen et al. 2021] are more sensitive to the initial grid scale, requiring precise tuning to ensure adequate coverage of the target shape. If the grid is too small, some parts of the shape could be missed. If the grid is too large, the resulting mesh lacks detail.

Delaunay triangulation. To create a Delaunay tetrahedral grid from a point cloud, we use TetGen [Si 2015]. In practice, we observe that TetGen can stall when points are too close to one another. To remove this issue we perturb points that are too close to each other.

Resampling and adaptive meshing. Our approach involves resampling passive points while progressively adding new points during optimization. We gradually increase the number of points from the initial 8000 to the desired target count. For the mesh reconstruction experiment, we sample camera positions uniformly on a sphere using a Fibonacci lattice. We compute the normal maps of both the target shape and the current shape and evaluate the per-pixel difference using an L_1 loss. To facilitate adaptive resampling, we divide the space around the current shape into voxels using a grid resolution of 32^3. For each voxel we determine a target number of sample points k. To achieve this, we start with a precomputed blue noise point set which we scale such that k points fall within the voxel.

Parameters. The final optimization objective can be written as

$$\mathcal{L} = \mathcal{L}_{\text{recons}} + \lambda_{\text{fairness}}\mathcal{L}_{\text{fairness}} + \lambda_{\text{ODT}}\mathcal{L}_{\text{ODT}} + \lambda_{\text{sign}}\mathcal{L}_{\text{sign}}$$

In our base experiment, we use $\lambda_M = 10$, $\lambda_D = 250$ $\lambda_N = 1$, $\lambda_{\text{fairness}} = 0.35$, $\lambda_{\text{ODT}} = 0.1$, and $\lambda_{\text{sign}} = 1.0$. We use spherical harmonics of degree 2. Because the computation of the vertices of the generated mesh is differentiable with respect to the grid points, the signed distance function and the spherical harmonics coefficients, we can leverage PyTorch's automatic differentiation [Paszke et al. 2019] and update parameters using the AdamW optimizer [Loshchilov and Hutter 2019]. The step size (0.002 for SDF values, 0.0003 for point positions) remains constant throughout the optimization process.

B Optimal Delaunay Triangulation

In this section, we provide an in-depth exploration of the Optimal Delaunay Triangulation loss E_{ODT} [Alliez et al. 2005; Chen and Xu 2004]. We explain how it is computed and prove that this energy is minimized on a regular tetrahedron. While Chen and Xu derive the update step to minimize the ODT loss, we are interested in computing the loss itself, to be included in a gradient-descent optimizer with autograd. This allows us to simultaneously optimize mesh quality and reconstruction losses.

B.1 Computation details

The vertex positions of a tetrahedron T are given as $\mathrm{v}_0, \ldots, \mathrm{v}_3$. The ODT energy for T is given as

$$E_{\text{ODT}}(T) = |M_{S_T} - M_T|. \tag{5}$$

where M_T represents the sum of the principal moments of T relative to its circumcenter c_T. M_{S_T} denotes the moment of inertia of S_T, defined as the circumsphere of T and having an equivalent mass. Assuming unit mass density, the mass of T is simply its volume V_T. With

$$\mathbf{a} = \mathrm{v}_1 - \mathrm{v}_0, \quad \mathbf{b} = \mathrm{v}_2 - \mathrm{v}_0, \quad \mathbf{c} = \mathrm{v}_3 - \mathrm{v}_0, \quad D = \det([\mathbf{a}, \mathbf{b}, \mathbf{c}]).$$

We obtain

$$V_T = \frac{|D|}{6}.$$

The circumcenter c_T is computed as [Weisstein 2025]

$$c_T = \mathrm{v}_0 + \frac{\|\mathbf{a}\|^2(\mathbf{b}\times\mathbf{c}) + \|\mathbf{b}\|^2(\mathbf{c}\times\mathbf{a}) + \|\mathbf{c}\|^2(\mathbf{a}\times\mathbf{b})}{2D}.$$

The circumradius is given by $R_T = \|c_T - \mathrm{v}_0\|$. Hence, the moment of inertia of S_T is given as

$$M_{S_T} = \frac{2}{5}V_T R_T^2. \tag{6}$$

An explicit formula for computing M_T is given by Tonon [2005]. We define the relative coordinates of the vertices with respect to the circumcenter c_T:

$$x_i = \mathrm{v}_i^x - c_T^x, \quad y_i = \mathrm{v}_i^y - c_T^y, \quad z_i = \mathrm{v}_i^z - c_T^z, \quad \text{for } i = 0, 1, 2, 3.$$

The coordinate quadratic sums are given by

$$S_x = \sum_{i=0}^{3} x_i^2 + \sum_{0\le i<j\le 3} x_i x_j.$$

We define S_y and S_z similarly for the y and z coordinates. Computing the moments of inertia along each axis yields:

$$I_x = \frac{V_T}{10}(S_y + S_z), \quad I_y = \frac{V_T}{10}(S_x + S_z), \quad I_z = \frac{V_T}{10}(S_x + S_y),$$

where V_T is the volume of the tetrahedron. Hence, the sum of the principal moments of inertia is given by

$$M_T = I_x + I_y + I_z = \frac{1}{5} V_T \left(S_x + S_y + S_z\right). \tag{7}$$

B.2 Theoretical Analysis

We show the following property:

Proposition B.1. *For any regular tetrahedron T, $E_{ODT}(T) = 0$.*

Proof. M_{S_T} and M_T are invariant under rigid transformations of T. The regular tetrahedron T_α is given by

$$\mathrm{v}_0 = (\alpha, \alpha, \alpha)\,,\ \mathrm{v}_1 = (\alpha, -\alpha, -\alpha)\,,\ \mathrm{v}_2 = (-\alpha, \alpha, -\alpha)\,,\ \mathrm{v}_3 = (-\alpha, -\alpha, \alpha)\,.$$

Because the circumcenter of a regular tetrahedron is its barycenter, we have $c_{T_\alpha} = (0, 0, 0)$ and therefore $R^2_{T_\alpha} = 3\alpha^2$. This evaluates to $S_x = S_y = S_z = 2\alpha^2$. Combining these with Eq. (6) and (7), we obtain

$$M_{S_{T_\alpha}} = M_{T_\alpha} = \frac{6}{5} V_{T_\alpha} \alpha^2.$$

Since E_{ODT} is the absolute value of the difference of these quantities, it attains the value zero for the regular tetrahedron. □

C Evaluation details

We explain the metrics used in our evaluation and provide additional rendering-based metrics.

C.1 Metric definition

In this section we define the metrics used for our evaluation. They are consistent with the ones used in FlexiCubes [Shen et al. 2023]. The chamfer distance (**CD**) quantifies the similarity between two point clouds. For each point in one of the point sets we compute the distance to the closest point in the other set. Averaging all distances yields the chamfer distance. To compare meshes, we sample points on both the ground-truth and reconstructed mesh, resulting in a point cloud with one million points for each mesh. We also compute the F1-score (**F1**) as

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

based on the sampled points. If the nearest point has a distance below a certain threshold, we count it as a true positive (TP). If its distance is above the threshold, it is counted as a false positive (FP) if it belongs to the reconstructed mesh, and a false negative (FN) if it belongs to the ground truth. We use a threshold of 0.001. The edge chamfer distance (**ECD**) and edge F1-score (**EF1**) are similar to the chamfer distance and F1 score, but apply to edge points [Chen et al. 2022], *i.e.* points whose normal has an average dot product with neighboring points' normals that falls below a threshold of 0.2. The inaccurate-normals metric (**IN** $> 5°$) captures the percentage of points for which the angle difference between predicted and ground truth normals exceeds 5 degrees. Normals of sampled points are copied from the face containing the sample point. The nearest point pairs between the predicted and ground truth meshes are identified to compute these angular differences. For a triangle, the aspect ratio (**AR**) is defined as the ratio of the longest edge to the shortest altitude, while the radius ratio (**RR**) is defined as the ratio of the inradius to the circumradius. Both metrics assess triangle quality on the extracted mesh, with lower values indicating better triangle regularity. The small angles metric (**SA**$< 10°$) calculates the percentage of triangles with smallest internal angle below $10°$, which is a proxy for the amount of sliver triangles. We also compute the percentage of intersecting faces (**SI**) using PyMeshLab [Muntoni and Cignoni 2021].

C.2 Rendering metrics

The rendering metrics presented in Table 4 evaluate the fidelity of the reconstructed meshes to the ground truth through image-based comparisons, rather than purely geometric metrics. These metrics – mask error, depth error, and normals error – offer insight into how well the reconstructed shape aligns perceptually with the target shape when rendered from various viewpoints. TetWeave demonstrates competitive performance compared to FlexiCubes [Shen et al. 2023] and DMTet [Shen et al. 2021], consistently achieving lower errors across all metrics as the number of points is scaled up. This indicates that TetWeave generates surfaces that are more visually aligned with the target shape in terms of silhouette, depth, and normal accuracy.

Table 4. Quantitative evaluation on the mesh reconstruction task with rendering metrics. We sample two thousand points using Fibonacci sampling over a sphere to position cameras and render the mask, depth map, and normal map of the reconstructed mesh and ground truth. We use an L_2 distance as comparison metric. The mask error is scaled by 10^3, the depth error by 10^2, and the normal error by 10^1.

Method	Mask Error $\times 10^3 \downarrow$	Depth Error $\times 10^2 \downarrow$	Normals Error $\times 10^1 \downarrow$
DMTet (128^3)	0.56	1.12	2.15
FlexiCubes (128^3)	0.29	0.54	1.51
TetWeave (16K)	0.39	0.70	1.79
TetWeave (64K)	0.23	0.36	1.30
TetWeave (128K)	0.17	0.26	1.10

Table 5. Expanded version of Table 2 on quantitative evaluation on the mesh reconstruction task. We refer to the text in the Appendix for a more detailed explanation of the different quantities. The chamfer distance (CD) is scaled by 1e5, and the edge chamfer distance (ECD) by 1e2. Ablation measures are done with respect to TetWeave with 64K grid points.

Method	CD ↓	F1 ↑	ECD ↓	EF1 ↑	NC ↑	IN> 5°(%) ↓	AR> 4(%) ↓	RR> 4(%) ↓	SA< 10°(%) ↓	SI(%) ↓	#V	#F
DMTet (32^3)	28.054	0.132	4.384	0.150	0.895	71.095	12.755	12.649	13.044	0.000	1355	27102
DMTet (64^3)	7.036	0.219	2.983	0.187	0.936	61.504	11.613	11.425	11.908	0.000	5030	10066
DMTet (128^3)	1.043	0.339	1.681	0.272	0.965	48.393	12.026	11.826	12.351	0.000	20677	41364
FlexiCubes (32^3)	12.350	0.210	3.784	0.191	0.931	62.187	7.275	8.714	6.362	0.775	1776	3556
FlexiCubes (64^3)	2.900	0.329	2.378	0.257	0.961	49.814	6.055	7.327	5.236	0.341	7086	14181
FlexiCubes (128^3)	0.752	0.416	1.254	0.393	0.979	36.911	5.418	6.701	4.588	0.203	28430	56873
TetWeave (8K)	1.176	0.376	1.952	0.283	0.967	48.635	2.511	3.534	1.922	0.000	14715	29468
TetWeave (16K)	0.517	0.409	1.475	0.353	0.974	43.380	2.350	3.344	1.743	0.000	26484	53015
TetWeave (32K)	0.489	0.431	1.157	0.433	0.979	38.367	2.292	3.287	1.675	0.000	46640	93330
TetWeave (64K)	0.419	0.446	0.962	0.518	0.984	33.700	2.251	3.252	1.616	0.000	81027	162102
TetWeave (128K)	0.393	0.455	0.708	0.588	0.987	29.361	2.507	3.556	1.829	0.000	146514	293074
TetWeave - 64K												
No SH	0.415	0.436	0.919	0.466	0.981	37.712	2.180	3.147	1.556	0.000	83866	167785
SH - degree 1	0.402	0.446	0.787	0.528	0.984	33.633	2.254	3.245	1.624	0.000	80865	161775
SH - degree 2	0.419	0.446	0.962	0.518	0.984	33.700	2.251	3.252	1.616	0.000	81027	162102
SH - degree 3	0.411	0.446	0.916	0.507	0.984	33.864	2.245	3.253	1.611	0.000	81141	162329
SH - degree 4	0.412	0.446	1.212	0.508	0.984	33.949	2.261	3.271	1.623	0.000	81699	163446
TetWeave - 64K												
No Fairness	0.781	0.455	1.119	0.511	0.987	28.866	17.091	19.353	15.611	0.000	127428	255107
No ODT	0.419	0.446	1.071	0.517	0.984	33.733	2.253	3.248	1.617	0.000	81320	162682
Uniform	0.427	0.436	1.051	0.427	0.980	36.453	1.567	2.514	0.975	0.000	61794	123635

Table 6. Comparison on Stanford ORB [Kuang et al. 2023] using NVDiffRec [Munkberg et al. 2022] as a backbone. Depth SI-MSE $\times 10^{-3}$. Shape Chamfer distance $\times 10^{-3}$. We highlight in bold the best-performing techniques overall, as well as the top-performing technique specifically within the isosurface extraction category, which is the primary focus of our analysis.

	Geometry			Novel Scene Relighting				Novel View Synthesis			
	Depth↓	Normal↓	Shape↓	PSNR-H↑	PSNR-L↑	SSIM↑	LPIPS↓	PSNR-H↑	PSNR-L↑	SSIM↑	LPIPS↓
IDR [Yariv et al. 2020]	0.35	0.05	**0.30**		N/A			**30.11**	**39.66**	**0.990**	**0.017**
NeRF [Mildenhall et al. 2020]	2.19	0.62	62.05		N/A			26.31	33.59	0.968	0.044
Neural-PIL [Boss et al. 2021b]	0.86	0.29	4.14		N/A			25.79	33.35	0.963	0.051
PhySG [Zhang et al. 2021a]	1.90	0.17	9.28	21.81	28.11	0.960	0.055	24.24	32.15	0.974	0.047
NeRD [Boss et al. 2021a]	1.39	0.28	13.70	23.29	29.65	0.957	0.059	25.83	32.61	0.963	0.054
NeRFactor [Zhang et al. 2021b]	0.87	0.29	9.53	23.54	30.38	0.969	0.048	26.06	33.47	0.973	0.046
InvRender [Wu et al. 2023]	0.59	0.06	0.44	23.76	30.83	0.970	0.046	25.91	34.01	0.977	0.042
NVDiffRecMC [Hasselgren et al. 2022]	0.32	**0.04**	0.51	24.43	31.60	0.972	0.036	28.03	36.40	0.982	0.028
DMTet [Shen et al. 2021]	**0.31**	0.06	0.62	22.91	29.72	0.963	0.039	21.94	28.44	0.969	0.030
FlexiCubes [Shen et al. 2023]	0.32	**0.05**	**0.49**	23.26	29.99	0.964	**0.037**	22.21	28.72	**0.970**	**0.028**
TetWeave	0.35	0.07	0.58	**23.48**	**30.30**	**0.965**	0.038	**22.30**	**28.98**	**0.970**	0.031