LEARNING-BASED DELAUNAY TETRAHEDRAL MESH GENERATION FROM SIGNED DISTANCE FIELD

Anonymous authors
Paper under double-blind review

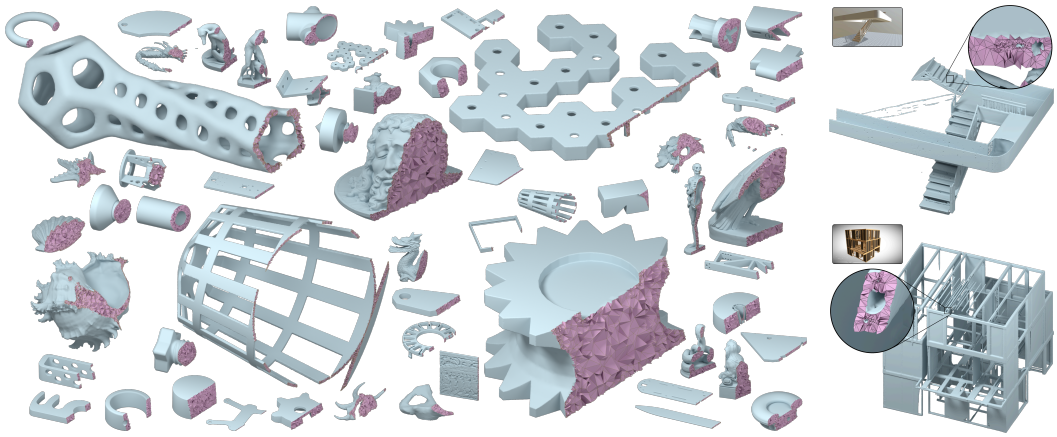


Figure 1: Our framework generates adaptive Delaunay tetrahedral meshes from SDFs across diverse shapes. **Left**: meshes from clean SDFs with ground-truth meshes. **Right**: meshes from SDFs generated by an image-to-3D model from the input images shown in the insets. CGAL (Jamin et al., 2015) fails to mesh these within a 10-minute time budget, while ours succeeds under a minute.

ABSTRACT

Converting signed distance fields (SDFs) into tetrahedral meshes requires balancing geometric accuracy, element quality, and computational cost. We present a learning-guided framework that combines adaptive sampling with classical geometric construction to accelerate this process. A neural network predicts density fields from the input SDF, learning where to allocate surface and volume samples from density targets derived from reference tetrahedral meshes. Adaptive blue-noise sampling, Delaunay tetrahedralization, and marching tetrahedra then construct a mesh whose boundary follows the input field. A post-processing stage then fixes geometric and topological errors and improves element quality. To validate our approach, we evaluate accuracy, element quality, topology, complexity, and runtime. On four held-out datasets at grid resolutions from 128^3 to 384^3, our method matches the geometric accuracy of CGAL while running up to $2.4\times$ faster, and *every* output mesh is closed, manifold, and free of slivers. Applied to SDFs produced by an image-to-3D generative model, it robustly and efficiently meshes shapes on which classical algorithms fail.

1 INTRODUCTION

Signed distance fields (SDFs) provide a convenient description of three-dimensional geometry, and are widely used in various ML frameworks, including generative models (Li et al., 2025; Zhao et al., 2025; Xiang et al., 2024). However, many downstream applications require an explicit discretization of the enclosed volume. Tetrahedral meshes support finite-element analysis, deformation, and physically compatible object modeling (Shewchuk, 2002; Müller et al., 2002; Guo et al., 2024). Generating a useful mesh requires more than extracting a surface: it must resolve geometric detail, avoid poorly shaped elements, and allocate resolution adaptively to keep the volume mesh compact.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

These requirements are tightly coupled. Dense sampling can recover thin structures and fine detail, but increases both meshing cost and the size of subsequent computations. Element shape also matters: interpolation accuracy and stiffness-matrix conditioning depend on geometric properties beyond element size (Shewchuk, 2002). Consequently, simulation-oriented meshing must balance geometric fidelity, element quality, and element count (Cutler et al., 2004).

Classical methods achieve this balance through geometric refinement and optimization. CGAL supports implicit domains directly using Delaunay refinement and mesh optimization (Jamin et al., 2015). TetWild and fTetWild robustly tetrahedralize surface meshes (Hu et al., 2018; 2020); applying them to an SDF requires first extracting a surface mesh. Although effective, these methods perform geometric refinement and local improvement anew for each input, making construction costly when processing many detailed shapes. This motivates our central question: can learning from previously generated meshes reduce this per-shape work while retaining adaptive resolution and geometric accuracy?

However, learning tetrahedral meshes directly is difficult: a valid mesh requires discrete, conforming connectivity, and Delaunay connectivity changes discontinuously with point positions. Prior work sidesteps this by learning on grids with fixed connectivity (Shen et al., 2021; Gao et al., 2022b; Shen et al., 2023) or by predicting sizing fields for conventional meshers (Zhang et al., 2025). We instead target fast adaptive meshing of a given SDF that preserves its geometry and controls element quality.

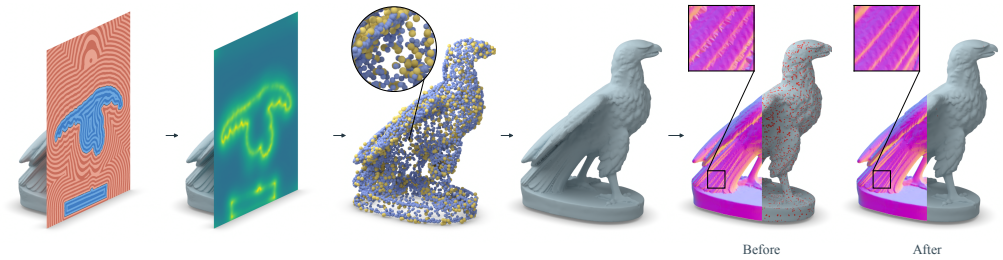
Our key design is to learn geometric decisions without directly predicting connectivity. Specifically, we learn *where to place seed points for Delaunay tetrahedralization*. A density network directly consumes the SDF and predicts surface and volume sampling densities, supervised by the spatial allocation of reference mesh vertices rather than their ordering or connectivity. Adaptive blue-noise sampling then draws well-separated seed points from these densities, which efficiently yields well-shaped tetrahedra. Delaunay tetrahedralization and marching tetrahedra construct the volume mesh, and a post-processing stage repairs geometric and topological errors and removes slivers.

We evaluate on 797 unseen shapes from four datasets, Thingi10K and ABC (Zhou & Jacobson, 2016; Koch et al., 2019) and the zero-shot SHREC11 and Threedscans (Lian et al., 2011; Laric, 2012), at three SDF resolutions (128^3, 256^3, 384^3). Our method matches the accuracy of CGAL while running 1.8–$2.4\times$ faster, and every mesh it produces is a closed manifold without slivers, whereas up to 8% of CGAL meshes violate this criterion. This comes at 1.4–$1.6\times$ the tetrahedron count. On SDFs generated by TripoSG (Li et al., 2025) from input images, our method meshes every shape, including those on which CGAL fails (Figure 1). Our key contributions are:

- A learning-guided SDF-to-tetrahedra framework that separates continuous density prediction from discrete connectivity construction through adaptive blue-noise sampling and Delaunay tetrahedralization.
- A post-processing pipeline tailored to these Delaunay meshes, which repairs geometric and topological errors by point insertion and removes slivers with edge flips and vertex perturbation.
- An evaluation on four mesh datasets and on image-to-3D generated SDFs, showing faster and more robust construction than CGAL with comparable accuracy and sliver-free, manifold meshes.

2 Related Work

Classical tetrahedral meshing. Classical tetrahedral meshing aims to balance geometric fidelity, adaptive element sizing, and element quality. Delaunay-based methods include CGALmesh for surface, image, and implicit domains (Jamin et al., 2015), TetGen for polyhedral domains (Si, 2015), and DistMesh, which combines distance functions with point relaxation (Persson & Strang, 2004); Gmsh also provides Delaunay-based tetrahedral meshing (Geuzaine & Remacle, 2009). Related reconstruction theory specifies sufficient sampling conditions through local feature size for smooth surfaces (Amenta & Bern, 1999). Lattice-based alternatives include isosurface stuffing, with dihedral-angle bounds and an internally graded variant (Labelle & Shewchuk, 2007), and lattice cleaving for multimaterial interfaces (Bronson et al., 2012). More recent methods address imperfect triangle soups: TetWild emphasizes robust tetrahedralization (Hu et al., 2018), while fTetWild accelerates construction using valid floating-point meshes (Hu et al., 2020). Classical pipelines generally



(a) Input SDF (b) Density (c) Point Sampling (d) DT + MT (e) Post-processing

Figure 2: **Pipeline overview.** (a) Given an input SDF, (b) we predict surface and volume sampling density fields. (c) We sample surface (blue) and volume (yellow) seeds, then (d) construct a tetrahedral mesh using Delaunay tetrahedralization (DT) followed by the marching tetrahedra (MT) algorithm. (e) Post-processing improves element quality and geometric consistency. Each before/after view shows normals on the left and a shaded mesh with slivers in red on the right.

rely on sequential refinement and optimization steps. Our method also builds on Delaunay tetrahedralization, but uses learned sampling densities to accelerate construction, parallelizing prediction and sampling on the GPU. We combine these learned components with classical post-processing to maintain high element quality.

Learning-based shape generation and meshing. Learning-based shape modeling produces explicit meshes directly (Nash et al., 2020; Siddiqui et al., 2024) or generates implicit fields (Cheng et al., 2023; Zhang et al., 2023). In the latter paradigm, many recent generation and reconstruction pipelines decode a field and extract its surface through isosurfacing (Li et al., 2025; Zhao et al., 2025; Wu et al., 2025; Xu et al., 2024; Wang et al., 2024; Xiang et al., 2024; Wei et al., 2024). Differentiable extraction (Liao et al., 2018; Remelli et al., 2020; Shen et al., 2021; 2023) and mesh formulations (Son et al., 2024; 2025; Liu et al., 2024) make geometry amenable to optimization and support generative modeling (Gao et al., 2022a; Liu et al., 2023; Kalischek et al., 2024). These efforts primarily target surface geometry, with adaptive Delaunay grids further improving surface extraction (Binninger et al., 2025). Some approaches extend shape modeling to tetrahedral representations (Gao et al., 2020; 2022b; Guo et al., 2025), but do not demonstrate volume meshing that jointly controls geometry-driven element sizing and tetrahedron quality. Similarly, FlexiCubes can produce tetrahedra, yet its demonstrated volume meshes retain uniform grid resolution at the boundary; its adaptive extraction extension targets surfaces (Shen et al., 2023).

Recent approaches predict application-specific sizing fields for conventional meshers (Freymuth et al., 2024; Zhang et al., 2025). We instead provide an end-to-end SDF-to-tetrahedra pipeline that automatically adapts resolution to geometric features, producing compact volume meshes. Its SDF input allows existing SDF-based shape generation pipelines to supply geometry without changes to their generation stages.

3 Method

3.1 Problem and Overview

Given a sampled signed distance field $f : \Omega \subset \mathbb{R}^3 \to \mathbb{R}$, we seek a tetrahedral mesh $\mathcal{M} = (V, T)$ approximating the solid $\{x : f(x) \leq 0\}$. Its boundary should follow the supplied zero level set, while its tetrahedra should have good element quality and spatially adaptive sizes. We evaluate these properties together with construction time, rather than treating a dense surface approximation as sufficient.

Our framework separates continuous prediction from discrete mesh construction (Figure 2). A density network predicts where to place seed points (Section 3.2), and blue-noise sampling draws well-separated seeds from these densities. Delaunay tetrahedralization (DT) of the seeds followed by marching tetrahedra (MT) yields an initial mesh fitted to the SDF (Section 3.3). A post-processing stage then repairs geometric and topological errors by inserting points into the retained DT and removes slivers with edge flips and vertex perturbation (Section 3.4). The network only guides where points go; connectivity is always constructed by geometric algorithms.

3.2 LEARNING SAMPLING DENSITY FIELDS

Given an input signed distance field f, we predict two nonnegative sampling density fields,

$$(\rho_s, \rho_v) = D_\theta(f), \tag{1}$$

where $\rho_s(x)$ is the sampling intensity for seeds that will be projected onto the surface, and $\rho_v(x)$ is the sampling intensity for volume seeds. Both fields are defined on the three-dimensional grid and have units of points per unit volume. Integrating each field gives the corresponding point count,

$$M_c = \int_\Omega \rho_c(x)\, dx \approx \Delta V \sum_{i \in \mathcal{G}} \rho_c(x_i), \qquad c \in \{s, v\}, \tag{2}$$

where $\mathcal{G}$ indexes the grid samples and $\Delta V = \Delta x\, \Delta y\, \Delta z$ is the volume represented by each sample. Thus, summing density values and multiplying by ΔV approximates the spatial integral. We use these masses to determine integer sampling counts:

$$N \approx M_s + M_v, \qquad N_s \approx N \frac{M_s}{M_s + M_v}, \qquad N_v = N - N_s. \tag{3}$$

Here N_s and N_v specify how many surface and volume seeds to sample, respectively; subsequent proximity-based selection may change the retained counts. In this way, the predicted densities specify both the spatial distribution of points and the number of points to sample for the given SDF.

Target Density. We construct target sampling densities from reference tetrahedral meshes generated by CGAL under surface-approximation, element-size, and quality criteria (Jamin et al., 2015; CGAL Project, 2026). However, different random seeds can produce different vertex allocations for the same shape, since multiple point configurations can represent the same geometry. The *Raw* panels in Figure 3 illustrate this variation. Directly using raw point counts as density targets can therefore introduce inconsistent supervision for the same input shape. To reduce this variation and obtain smooth target densities, we separate boundary and interior vertices and apply Gaussian kernel density estimation (KDE) (Rosenblatt, 1956) to each set: each vertex contributes a Gaussian kernel, and their sum defines a continuous point-intensity field. The resulting grid-aligned targets are smoother than raw densities and less sensitive to individual vertex placements, as illustrated by the *KDE* panels in Figure 3.

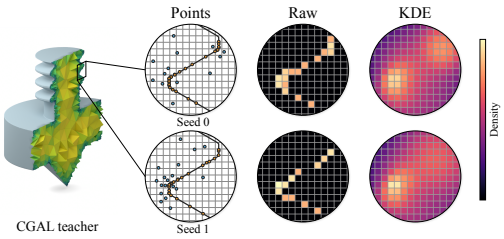
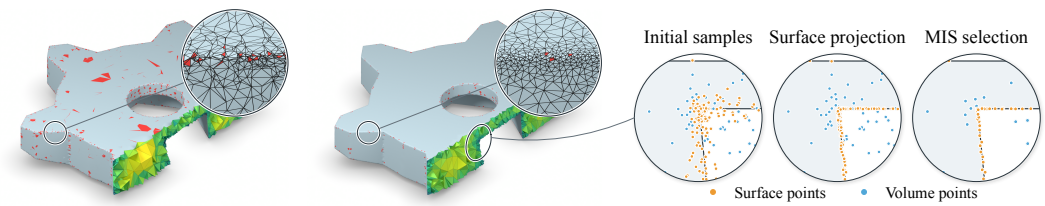


Figure 3: **KDE-based Target Density.** Two CGAL seeds in the same screw-groove region. This local 2D illustration uses only the displayed surface points (orange): Raw is point count per cell area; KDE is cell-averaged Gaussian density. Volume points (blue) are shown for context; their density targets use the same KDE procedure.

Architecture. We aim to train a single density predictor that accepts SDF grids at 128^3, 256^3, and 384^3 using one shared set of parameters. To this end, we define D_θ in Eq. (1) using a global 3D U-Net G_ϕ and a local residual branch R_ψ, with $\theta = (\phi, \psi)$, trained jointly with targets represented in log-density space, $\ell_c = \log_{10}(1 + \rho_c)$ for $c \in \{s, v\}$. G_ϕ processes the input SDF on a subsampled 128^3 grid and predicts coarse log-density fields, which we interpolate to the input resolution. R_ψ uses the original-resolution SDF, its geometric features, the interpolated prediction, and the grid resolution to predict a residual. During inference, we evaluate R_ψ on spatial tiles with 64^3 cores and surrounding context to limit the memory cost of high-resolution features. We add this residual to the global prediction in log-density space, weighted by an SDF-based mask that vanishes outside a narrow band around the surface.

Loss. We jointly minimize $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{global}} + \mathcal{L}_{\text{patch}}$, supervising the output of G_ϕ on the 128^3 grid and the combined prediction from G_ϕ and R_ψ on sampled 64^3 patches at the input resolution. These losses combine log-density regression, integrated-mass and normalized-distribution matching, multiscale and gradient consistency, and one-sided penalties for missing surface density. Together, they encourage accurate spatial allocation and point counts. Appendix A provides the component definitions, weights, and augmentation details.



(a) Mesh from random sampling (b) Mesh from blue-noise sampling (c) Blue-noise sampling process

Figure 4: **Blue-noise sampling and initial mesh quality.** (a,b) Meshes from the same predicted densities and matched point counts, using DT/MT without post-processing. Blue-noise sampling significantly improves mesh quality; red exterior faces mark sliver tetrahedra. (c) Density-sampled points, surface projection, and MIS selection in the marked region.

3.3 Initial Mesh Construction

3.3.1 Point sampling

After predicting density fields, we sample points for Delaunay tetrahedralization according to the two density channels. For each channel, the predicted density specifies the number of points per unit volume. We normalize this field by its integrated mass to obtain a probability density for the location of a single point, so regions with higher predicted density are more likely to be sampled. This distribution describes initial positions in the three-dimensional domain:

$$p_c(x) = \frac{\rho_c(x)}{M_c}, \qquad \int_\Omega p_c(x)\,dx = 1, \qquad c \in \{s, v\}, \tag{4}$$

where $M_c > 0$ is the integrated mass, or point count, defined in Eq. (2).

For surface samples, we draw $x_s^{(0)} \sim p_s$ and project toward the zero level set by iterating

$$x_s^{(k+1)} = \Pi_\Omega\left(x_s^{(k)} - \frac{f(x_s^{(k)})\,\nabla f(x_s^{(k)})}{\max\left(\|\nabla f(x_s^{(k)})\|^2, \varepsilon\right)}\right), \tag{5}$$

where Π_Ω clamps to the grid domain, ∇f is estimated by finite differences, and $\varepsilon > 0$ stabilizes the denominator. We use three projection iterations and retain only samples satisfying $|f(x_s^{(3)})| < 10^{-3}$ in the normalized coordinate system. Volume samples follow p_v without this projection.

Blue Noise Sampling. Although we can sample independently from the probability density in Eq. (4), this can produce uneven spacing and poor-quality tetrahedra (Figure 4a). We therefore use blue-noise sampling to suppress clustering (Bowers et al., 2010), motivated by its use in tetrahedral meshing (Guo et al., 2016). Using the point counts in Eq. (3), we draw $4N_s$ surface and $4N_v$ volume candidates before applying maximal-independent-set (MIS) selection (Luby, 1986) on a graph connecting candidates that violate density-dependent exclusion distances. For both channels, these distances are based on the local spacing

$$d(x) = \rho(x)^{-1/3}, \tag{6}$$

where $\rho = \rho_s + \rho_v$. Note that higher local density permits closer spacing. We first select projected surface candidates, then keep them fixed during volume selection, enforcing separation within and between channels. Figure 4b shows the improved initial mesh quality with blue-noise sampling, and Figure 4c illustrates the entire blue-noise sampling process in 2D. Sampling runs on the GPU, with custom CUDA kernels for conflict-pair construction.

3.3.2 Delaunay Tetrahedralization (DT) and Marching Tetrahedra (MT)

For the sampled seed points, we compute the DT using CGAL and evaluate the original SDF at its vertices. Then, MT subdivides cells crossed by the zero level set and retains the interior volume. We refine edge intersections with 6 bisection iterations against the sampled SDF to locate zero crossings more accurately. Adjacent cells share intersection vertices so that the construction is consistent across their common faces. Although blue-noise sampling produces well-spaced seeds, cutting near a vertex or edge can still create small number of poor-quality tetrahedra (Figure 4b). This motivates the following post-processing stage.

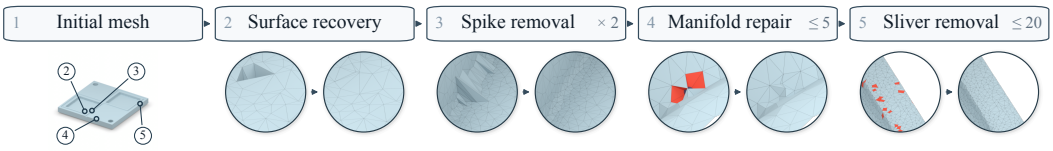


Figure 5: **Post-processing pipeline.** Starting from the initial mesh, surface recovery fills dents, spike removal and manifold repair insert points to remove spikes and non-manifold configurations, and sliver removal eliminates slivers; the numbers give how many times each stage repeats. Local before/after pairs mark the affected faces in red.

3.4 Post-processing

The initial mesh can contain four kinds of defects: dents where the boundary misses the zero level set, spikes that protrude outside it, non-manifold configurations, and ill-shaped elements. Following prior angle-based sliver-removal work (Jamin et al., 2015; Guo et al., 2016), we identify ill-shaped elements as slivers, tetrahedra whose minimum dihedral angle is below $5°$. Post-processing combines two operations to remove these defects – point insertion and flip with perturbation, which the stages in Figure 5 apply in sequence.

3.4.1 Point insertion

We retain the Delaunay tetrahedralization from initial construction. A point insertion adds new points to it, assigns each point the trilinearly interpolated SDF value, updates the DT incrementally, and re-extracts the mesh with MT. The operations below differ only in where points are placed.

Surface recovery. We sample the input SDF at $(N + 1)$ points along every DT edge and, for each sign change, locate the zero crossing by bisection on the input field. These roots are inserted with SDF value zero, so the extracted boundary passes through them. We use $N = 128$.

Spike removal. We draw N random points in each tetrahedron and mark it as a spike when at least 90% of them lie outside the SDF. We insert their circumcenters, followed by surface recovery. We apply this operation twice, with $N = 128$.

Manifold repair. We detect non-manifold edges, shared by more than two boundary faces, and non-manifold vertices, whose incident boundary faces do not form a single connected fan. We insert the circumcenters of all tetrahedra incident to these elements, followed by surface recovery, and repeat for at most five rounds or until no non-manifold element remains.

3.4.2 Flip and perturbation

Slivers are removed mostly without adding points. Each cycle runs three steps and repeats up to 20 times or until no sliver remains.

Greedy bistellar flips. We repeatedly take the sliver with the smallest minimum dihedral angle and apply the bistellar $2 \to 3$ or $3 \to 2$ flip (Joe, 1995) that most improves the affected tetrahedra under a lexicographic score: the number of tetrahedra below $5°$, then below $10°$, then the smallest dihedral angle. Flips must preserve the enclosed volume and the boundary; slivers without an improving flip are skipped until their neighborhood changes, and the loop stops when no improving flip remains.

Vertex perturbation. We then relocate vertices for up to 40 rounds, visiting the vertices of the current slivers in order of increasing angle. Each vertex moves along the gradient of its worst incident dihedral angle; for boundary vertices, this direction is projected onto the SDF tangent plane and the point is pulled back onto its original isovalue. A line search over eight halving steps keeps the best move that improves the same score without inverting any tetrahedron, within a quarter voxel of the original position.

Insertion at residual slivers. Slivers that survive both steps receive the circumcenters of the DT tetrahedra overlapping them. Because these candidates cluster, we process them in order of increasing circumradius and keep one only if its circumsphere contains no kept candidate; the kept points are inserted, followed by surface recovery, and the next cycle begins.

4 EXPERIMENTAL RESULTS

We evaluate our pipeline on SDFs constructed from existing mesh datasets (Section 4.2) and on SDFs generated from images by TripoSG (Li et al., 2025) (Section 4.3).

4.1 EXPERIMENTAL SETUP

Training, hardware, and timing details are given in Appendix B.

Inputs. We use SDFs sampled on regular grids over $[-1, 1]^3$ at 128^3, 256^3, and 384^3, with the same input field for all methods. Generated SDFs are first redistanced with a CUDA jump-flooding method (Rong & Tan, 2006) to turn them into valid SDFs.

Baselines. We compare against four baselines: CGAL (Jamin et al., 2015), fTetWild (Hu et al., 2020), Gmsh (Geuzaine & Remacle, 2009), and TetGen (Si, 2015). Among them only CGAL is directly comparable to our method, because it consumes the SDF itself. The other three accept only a surface mesh, so we first extract one from the same field with marching cubes (Lorensen & Cline, 1987) and feed it to them, and their reported runtimes include this extraction. For this reason we treat those three as a secondary comparison and report them in the mesh-derived experiment only; the generated-shape experiment compares against CGAL alone. Appendix C reports the parameters we use for each baseline.

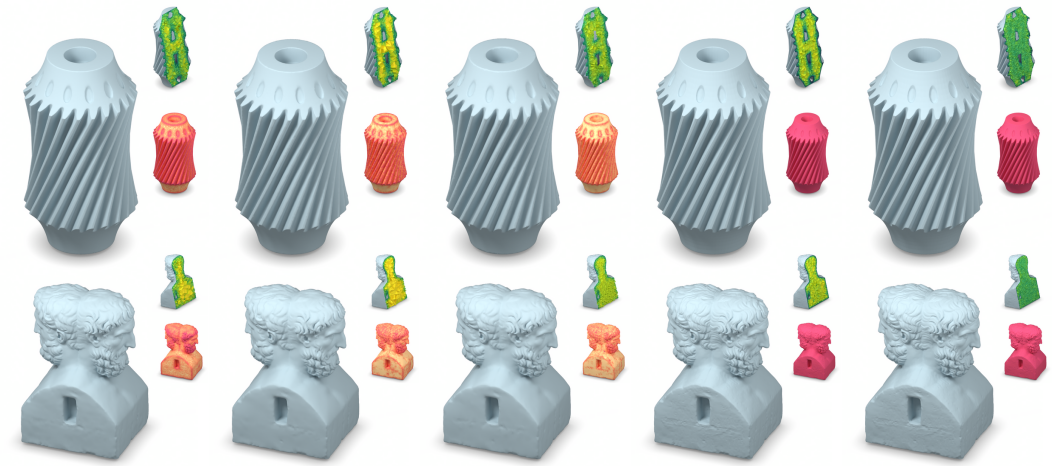
Metrics. For accuracy, we compare the output boundary to the marching-cubes surface of the input field at the original grid resolution, which we take as the reference. We independently sample 100K points uniformly by surface area from the generated boundary and the reference surface. *CD* is the sum of the two directed mean nearest-sample distances; *Normal* is the bidirectional mean absolute dot product of corresponding triangle face normals from the two meshes. For output mesh complexity and computational cost, we report the number of tetrahedra (T), alongside the entire runtime (*Time*). For robustness we report the failure rate, the percentage of cases in which the mesher reports an error or does not finish within the 10 minute limit we allow per case. For element quality and topology, we report S_0, the percentage of shapes whose output mesh is a closed manifold and free of *slivers*, tetrahedra whose minimum dihedral angle falls below 5°.

4.2 MESHING MESH-DERIVED SDFS

Datasets. We use four datasets to cover diverse geometry: Thingi10K (Zhou & Jacobson, 2016), ABC (Koch et al., 2019), SHREC11 (Lian et al., 2011), and Threedscans (Laric, 2012). Training uses only Thingi10K and ABC. We select 2,505 and 2,000 shapes from these datasets, respectively, and partition the combined set into 3,603 training, 451 validation, and 451 test shapes. We additionally use 458 meshes from SHREC11 and 20 meshes from Threedscans, which serve only for zero-shot testing. We center each mesh at its bounding-box center and scale it uniformly to fit within $[-0.95, 0.95]^3$, leaving padding inside the SDF domain $[-1, 1]^3$. We then compute input SDFs from the normalized meshes using `mesh2sdf` (Wang et al., 2022), on uniform grids of 128^3, 256^3, and 384^3.

Discussion. For quantitative comparison we used 929 shapes in total, but baseline methods did not succeed on every shape. Failure rates over all attempted cases are 0.33% for CGAL, 0.14% for fTetWild, 4.52% for Gmsh, 0.54% for TetGen, and 0.00% for ours across all 2,787 cases (929 shapes at three resolutions). *It is notable that our method is the only one with a zero failure rate, which shows the robustness of our algorithm.* Every CGAL failure had the same cause: its initialization found no point on the surface, so refinement never started and it returned an empty mesh. This supports our approach of drawing samples from density fields predicted for the given shape: as the network tells us where to place both surface and volume samples, generation never depends on first discovering the surface by random search.

In Table 1 and Figure 6, we provide the quantitative and qualitative comparison of our method to the baseline methods. Compared to CGAL, on the common cases our method reaches accuracy comparable to it, and in general slightly better. At the same time our method attains an S_0 of 100%, whereas CGAL does not reach 100% S_0 at any resolution, mainly because of manifoldness issues that persist even though we enable its manifold option (Appendix C). Even more, our method achieves a **1.8x to 2.5x speedup** over CGAL while generating 1.4x to 1.6x as many tetrahedra, across all three resolutions; Figure 6(a, b) shows the same trade-off, our boundary matching CGAL's while



(a) Ours (b) CGAL (c) fTetWild (d) TetGen (e) Gmsh

Figure 6: **Qualitative comparison on unseen shapes.** Tetrahedral meshes generated from 384^3 input SDFs. Upper insets show interior tetrahedra colored by volume; lower insets visualize surface complexity through triangle area.

Table 1: **Quantitative comparison across four datasets.** Average accuracy and quality metrics over the same 797 unseen shapes from four mesh datasets that were successfully processed by all compared methods at all three resolutions. *CD* is given in 10^{-3} of the normalized domain $[-1, 1]^3$, *Normal* is a mean absolute dot product in $[0, 1]$, T counts tetrahedra in thousands, S_0 is a percentage of shapes, and *Time* is in seconds.

	128^3					256^3					384^3				
Method	CD$^{\downarrow}$	Normal$^{\uparrow}$	$T^{\downarrow}$	$S_0^{\uparrow}$	Time$^{\downarrow}$	CD$^{\downarrow}$	Normal$^{\uparrow}$	$T^{\downarrow}$	$S_0^{\uparrow}$	Time$^{\downarrow}$	CD$^{\downarrow}$	Normal$^{\uparrow}$	$T^{\downarrow}$	$S_0^{\uparrow}$	Time$^{\downarrow}$
CGAL (2015)	6.037	0.9901	70.2	92.10	1.54	6.027	0.9916	74.9	94.73	1.48	6.028	0.9915	81.0	93.10	1.66
fTetWild (2020)	6.121	0.9897	21.2	98.49	8.57	6.087	0.9901	31.4	99.25	17.65	6.092	0.9896	33.5	99.12	28.40
Gmsh (2009)	5.976	0.9922	130.4	0.00	3.51	5.992	0.9930	875.0	0.00	31.10	5.995	0.9930	2749.6	0.00	110.46
TetGen (2015)	5.977	0.9922	134.7	0.00	1.80	5.991	0.9930	583.6	0.00	10.86	5.995	0.9930	1353.4	0.00	29.10
w/o post-proc.	6.064	0.9872	98.4	0.00	0.14	6.065	0.9880	118.1	0.00	0.21	6.067	0.9876	130.5	0.00	0.23
w/o blue noise.	6.073	0.9884	100.6	88.28	4.80	6.063	0.9908	117.0	89.84	3.90	6.060	0.9909	132.5	92.19	5.42
Ours	6.033	0.9897	101.0	100.00	0.63	6.019	0.9920	117.9	100.00	0.71	6.019	0.9920	130.5	100.00	0.92

our interior carries more elements. This shows that our method improves on CGAL in accuracy, computational efficiency and robustness, at a modest cost in mesh complexity. When it comes to the other three baselines, the first thing to note is that they become much more expensive at higher resolution, which is undesirable when the input is a high-resolution field produced by a generative pipeline. This is because they consume an intermediate marching-cubes surface mesh rather than the SDF itself. fTetWild is better in terms of mesh complexity, producing very few tetrahedra, as it remeshes the surface (Figure 6(c)), but its accuracy is worse than both ours and CGAL, with a small number of its meshes either retaining slivers or failing to be closed manifolds. Gmsh and TetGen are not only computationally expensive on high-resolution inputs, their mesh complexity is also very high (Figure 6(d, e)) – making them difficult to use efficiently for downstream tasks. They also attain a 0% S_0 at every resolution, because no mesh of theirs is free of slivers. Both of these costs follow from the option we enable to preserve the input surface mesh (Appendix C), which was unavoidable due to their failure rate.

At the bottom of Table 1, we also provide the ablation studies about our major components—the blue noise sampling (Section 3.3) and post-processing (Section 3.4). Removing the post-processing makes generation roughly four times faster, but no mesh is then both closed and free of slivers, so S_0 drops to 0% at every resolution, and accuracy degrades slightly as well. Removing the blue noise instead leaves accuracy and complexity almost unchanged, yet S_0 falls to between 88% and 92% and the runtime grows by five to eight times. The sampling step itself is cheaper without blue noise; the cost reappears later, because the post-processing has to repair far more slivers in an independently

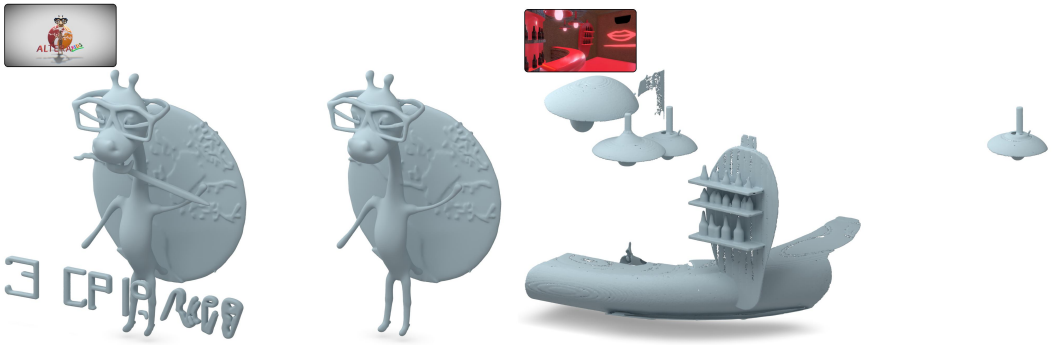


Figure 7: **Meshing SDFs generated by TripoSG.** For each example, mesh from ours (left; the input image is shown in the inset) and from CGAL (right) on the same TripoSG (Li et al., 2025) field at 384^3. CGAL misses disconnected parts, whereas ours meshes every part.

sampled point set. The two components are therefore complementary: blue noise produces a point set whose defects the post-processing can remove cheaply, and the post-processing is what turns that point set into a closed, sliver-free mesh.

4.3 Meshing Generated SDFs

We next mesh SDFs produced by an image-to-3D generative model. Such fields are harder than mesh-derived ones: they often contain thin structures, many disconnected parts, and small-scale noise. We take 55 shapes from the TripoSG (Li et al., 2025) test set, sample their fields at 384^3, and apply the same SDF redistancing before meshing with either method. We compare against CGAL, the only baseline that consumes the SDF directly, and use our density network trained on Thingi10K and ABC without fine-tuning.

Table 2: **Quantitative comparison on generated SDFs.** Metrics are the same as in Table 1.

Method	CD$^{\downarrow}$	Normal$^{\uparrow}$	$T^{\downarrow}$	$S_0^{\uparrow}$	Time$^{\downarrow}$
CGAL (2015)	28.935	0.9718	431.9	48.08	7.26
Ours	8.110	0.9775	643.1	100.00	7.23

Table 2 and Figure 7 show that our method is far more robust on these inputs. It meshes every shape without a single failure, whereas CGAL fails on three, and on the shapes that both methods mesh, all of our meshes satisfy S_0 compared with only 48% of CGAL's. Our meshes are also substantially more accurate in terms of CD, because CGAL often loses entire parts of the shape, as in Figure 7. The average runtimes are similar, but this comparison favors CGAL: in these catastrophic cases it meshes only a fraction of the shape and therefore spends little time, and the table excludes the shapes on which CGAL exceeded the ten-minute budget and was terminated. Accounting for those cases, we expect our method to be faster in practice.

5 Conclusion

We presented a learning-guided framework for converting SDFs into tetrahedral meshes, which learns where to place seed points for Delaunay tetrahedralization and relies on geometric algorithms and a dedicated post-processing stage for connectivity, element quality, and topology. On four held-out datasets, our method matches the accuracy of CGAL while running faster and producing closed, sliver-free manifold meshes, and when connected to an image-to-3D generative pipeline, it meshes every generated shape, including those on which CGAL fails.

Limitations and future work. Although our meshes are spatially adaptive, they still contain more tetrahedra than necessary: about 1.4–$1.6\times$ as many as CGAL, and several times more than fTetWild, which remeshes the surface. A notable source of this overhead is sharp features, where our unweighted Delaunay construction must place many seeds to resolve the boundary. Extending the framework to weighted Delaunay tetrahedralization, whose weights can protect sharp features with few points, is a promising way to reduce the element count toward the level of fTetWild while preserving accuracy and element quality.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks. (i) *Writing*: aiding and polishing the manuscript, such as improving wording, grammar, and concision. (ii) *Retrieval and discovery*: finding and summarizing related work. (iii) *Implementation*: writing and optimizing parts of the code, including acceleration of the post-processing stage, evaluation and aggregation scripts, and rendering scripts for figures. (iv) *Result analysis*: aggregating experimental results and assisting the analysis of failure cases and metric differences between methods. (v) *Figure creation*: rendering and composing figures from our experimental outputs. The research ideas, the design of the method, and the experimental protocol were developed by the authors. All AI-assisted code was reviewed and tested by the authors, and implementation changes were verified by comparing outputs against the original implementation and by re-running the evaluation. All AI-assisted text, analyses, and retrieved references were checked against the underlying data and original sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22:481–504, 1999. URL https://www.cs.ucdavis.edu/~amenta/pubs/crustTheory.pdf.

Alexandre Binninger, Ruben Wiersma, Philipp Herholz, and Olga Sorkine-Hornung. TetWeave: Isosurface extraction using on-the-fly Delaunay tetrahedral grids for gradient-based mesh optimization. *ACM Transactions on Graphics*, 2025. doi: 10.1145/3730851. URL https://alexandrebinninger.com/TetWeave/.

John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Transactions on Graphics*, 29(6):166:1–166:10, 2010. doi: 10.1145/1882261.1866188. URL https://doi.org/10.1145/1882261.1866188.

Jonathan R. Bronson, Joshua A. Levine, and Ross T. Whitaker. Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality. In *Proceedings of the 21st International Meshing Roundtable*, pp. 191–209, 2012. URL https://www.sci.utah.edu/publications/bronson12/Bronson_cleaving2012.pdf.

CGAL Project. Mesh_criteria_3: Refinement criteria for 3d mesh generation. CGAL User and Reference Manual, 2026. URL https://doc.cgal.org/latest/Mesh_3/classCGAL_1_1Mesh__criteria__3.html. Accessed September 7, 2026.

Yen-Chi Cheng, Hsin-Ying Lee, Sergey Tulyakov, Alexander G. Schwing, and Liang-Yan Gui. SDFusion: Multimodal 3D shape completion, reconstruction, and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4456–4465, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Cheng_SDFusion_Multimodal_3D_Shape_Completion_Reconstruction_and_Generation_CVPR_2023_paper.html.

Barbara Cutler, Julie Dorsey, and Leonard McMillan. Simplification and improvement of tetrahedral models for simulation. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pp. 93–102, 2004. doi: 10.1145/1057432.1057445. URL https://graphics.cs.yale.edu/publications/simplification-and-improvement-tetrahedral-models-simulation.

Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Philipp Becker, Aleksandar Taranovic, Onno Grönheim, Luise Kärger, and Gerhard Neumann. Iterative sizing field prediction for adaptive mesh generation from expert demonstrations. In *ICML Workshop on AI for Science*, 2024. URL https://arxiv.org/abs/2406.14161.

Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3D reconstruction. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2011.01437.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. GET3D: A generative model of high quality 3D textured shapes learned from images. In *Advances in Neural Information Processing Systems*, volume 35, pp. 31841–31854, 2022a. URL https://arxiv.org/abs/2209.11163.

William Gao, April Wang, Gal Metzer, Raymond A. Yeh, and Rana Hanocka. TetGAN: A convolutional neural network for tetrahedral mesh generation. In *British Machine Vision Conference*, 2022b. URL https://arxiv.org/abs/2210.05735.

Christophe Geuzaine and Jean-François Remacle. Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi: 10.1002/nme.2579. URL https://gmsh.info/.

Jianwei Guo, Dong-Ming Yan, Li Chen, Xiaopeng Zhang, Oliver Deussen, and Peter Wonka. Tetrahedral meshing via maximal Poisson-disk sampling. *Computer Aided Geometric Design*, 43: 186–199, 2016. doi: 10.1016/j.cagd.2016.02.004. URL https://doi.org/10.1016/j.cagd.2016.02.004.

Minghao Guo, Bohan Wang, Pingchuan Ma, Tianyuan Zhang, Crystal Elaine Owens, Chuang Gan, Joshua B. Tenenbaum, Kaiming He, and Wojciech Matusik. Physically compatible 3D object modeling from a single image. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2405.20510.

Minghao Guo, Bohan Wang, Kaiming He, and Wojciech Matusik. TetSphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2405.20283.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Transactions on Graphics*, 37(4), 2018. doi: 10.1145/3197517.3201353. URL https://github.com/Yixin-Hu/TetWild.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics*, 39(4), 2020. doi: 10.1145/3386569.3392385. URL https://github.com/wildmeshing/fTetWild.

Clément Jamin, Pierre Alliez, Mariette Yvinec, and Jean-Daniel Boissonnat. CGALmesh: A generic framework for Delaunay mesh generation. *ACM Transactions on Mathematical Software*, 41 (4), 2015. doi: 10.1145/2699463. URL https://www-sop.inria.fr/geometrica/software/cgalmesh/.

Barry Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16(6):1292–1307, 1995. doi: 10.1137/0916075. URL https://doi.org/10.1137/0916075.

Nikolai Kalischek, Torben Peters, Jan D. Wegner, and Konrad Schindler. TetraDiffusion: Tetrahedral diffusion models for 3D shape generation. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2211.13220.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. URL https://arxiv.org/abs/1812.06216.

François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, 26(3), 2007. doi: 10.1145/1275808.1276448. URL https://people.eecs.berkeley.edu/~sequin/CS285/PAPERS/SIGGRAPH_07/057-labelle_IsoSurfStuffing.pdf.

Oliver Laric. Three D Scans, 2012. URL https://threedscans.com/info/. Accessed September 2026.

Yangguang Li, Zi-Xin Zou, Zexiang Liu, Dehu Wang, Yuan Liang, Zhipeng Yu, Xingchao Liu, Yuan-Chen Guo, Ding Liang, Wanli Ouyang, and Yan-Pei Cao. TripoSG: High-Fidelity 3D Shape Synthesis using Large-Scale Rectified Flow Models. *arXiv preprint arXiv:2502.06608*, 2025. URL https://arxiv.org/abs/2502.06608.

Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. V. Nguyen, R. Ohbuchi, Y. Ohkita, Y. Ohishi, F. Porikli, M. Reuter, I. Sipiran, D. Smeets, P. Suetens, H. Tabia, and D. Vandermeulen. SHREC '11 track: Shape retrieval on non-rigid 3D watertight meshes. In *Eurographics Workshop on 3D Object Retrieval*, pp. 79–88. The Eurographics Association, 2011. doi: 10.2312/3DOR/3DOR11/079-088.

Yiyi Liao, Simon Donné, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2916–2925, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Liao_Deep_Marching_Cubes_CVPR_2018_paper.html.

Zhen Liu, Yao Feng, Michael J. Black, Derek Nowrouzezahrai, Liam Paull, and Weiyang Liu. MeshDiffusion: Score-based generative 3D mesh modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0cpM2ApF9p6.

Zhen Liu, Yao Feng, Yuliang Xiu, Weiyang Liu, Liam Paull, Michael J. Black, and Bernhard Schölkopf. Ghost on the shell: An expressive representation of general 3D shapes. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.15168.

William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi: 10.1137/0215074. URL https://doi.org/10.1137/0215074.

Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002. URL https://matthias-research.github.io/pages/publications/warp.pdf.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *Proceedings of the International Conference on Machine Learning*, volume 119, pp. 7220–7229, 2020. URL https://proceedings.mlr.press/v119/nash20a.html.

Per-Olof Persson and Gilbert Strang. A simple mesh generator in MATLAB. *SIAM Review*, 46 (2):329–345, 2004. URL https://persson.berkeley.edu/pub/persson04mesh.pdf.

Edoardo Remelli, Artem Lukoianov, Stephan R. Richter, Benoît Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. MeshSDF: Differentiable iso-surface extraction. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/file/fe40fb944ee700392ed51bfe84dd4e3d-Paper.pdf.

Guodong Rong and Tiow-Seng Tan. Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pp. 109–116, 2006.

Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956. doi: 10.1214/aoms/1177728190. URL https://doi.org/10.1214/aoms/1177728190.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: A hybrid representation for high-resolution 3D shape synthesis. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/30a237d18c50f563cba4531f1db44acf-Abstract.html.

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics*, 42(4), 2023. doi: 10.1145/3592430. URL https://research.nvidia.com/labs/toronto-ai/flexicubes/.

Jonathan Richard Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures, 2002. URL https://people.eecs.berkeley.edu/~jrs/papers/elemj.pdf. Unpublished preprint, University of California, Berkeley.

Hang Si. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software*, 41(2), 2015. doi: 10.1145/2629697. URL https://wias-berlin.de/software/tetgen/.

Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. MeshGPT: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19615–19625, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Siddiqui_MeshGPT_Generating_Triangle_Meshes_with_Decoder-Only_Transformers_CVPR_2024_paper.html.

Sanghyun Son, Matheus Gadelha, Yang Zhou, Zexiang Xu, Ming C. Lin, and Yi Zhou. DMesh: A differentiable representation for general meshes. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2404.13445.

Sanghyun Son, Matheus Gadelha, Yang Zhou, Matthew Fisher, Zexiang Xu, Yi-Ling Qiao, Ming C. Lin, and Yi Zhou. DMesh++: An efficient differentiable mesh for complex shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 26590–26599, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Son_DMesh_An_Efficient_Differentiable_Mesh_for_Complex_Shapes_ICCV_2025_paper.html.

Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (SIGGRAPH)*, 41(4), 2022. URL https://github.com/wang-ps/mesh2sdf.

Zhengyi Wang, Yikai Wang, Yifei Chen, Chendong Xiang, Shuo Chen, Dajiang Yu, Chongxuan Li, Hang Su, and Jun Zhu. CRM: Single Image to 3D Textured Mesh with Convolutional Reconstruction Model. *arXiv preprint arXiv:2403.05034*, 2024. URL https://arxiv.org/abs/2403.05034.

Xinyue Wei, Kai Zhang, Sai Bi, Hao Tan, Fujun Luan, Valentin Deschaintre, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. MeshLRM: Large Reconstruction Model for High-Quality Meshes. *arXiv preprint arXiv:2404.12385*, 2024. URL https://arxiv.org/abs/2404.12385.

Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Yajie Bao, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, and Yao Yao. Direct3D-S2: Gigascale 3D Generation Made Easy with Spatial Sparse Attention. *arXiv preprint arXiv:2505.17412*, 2025. URL https://arxiv.org/abs/2505.17412.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D Latents for Scalable and Versatile 3D Generation. *arXiv preprint arXiv:2412.01506*, 2024. URL https://arxiv.org/abs/2412.01506.

Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. InstantMesh: Efficient 3D Mesh Generation from a Single Image with Sparse-view Large Reconstruction Models. *arXiv preprint arXiv:2404.07191*, 2024. URL https://arxiv.org/abs/2404.07191.

Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics*, 42(4), 2023. URL https://arxiv.org/abs/2301.11445.

Zhiyuan Zhang, Amir Vaxman, Stefanos-Aldo Papanicolopulos, and Kartic Subr. Learned adaptive mesh generation. *arXiv preprint arXiv:2505.20457*, 2025. URL https://arxiv.org/abs/2505.20457.

Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, Mingxin Yang, Sheng Zhang, Xianghui Yang, Huiwen Shi, Sicong Liu, Junta Wu, Yihang Lian, Fan Yang, Ruining Tang, Zebin He, Xinzhou Wang, Jian Liu, Xuhui Zuo, Zhuo Chen, Biwen Lei, Haohan Weng, Jing Xu, Yiling Zhu, Xinhai Liu, Lixin Xu, Changrong Hu, Shaoxiong Yang, Song Zhang, Yang Liu, Tianyu Huang, Lifu Wang, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Yulin Cai, Jiaao Yu, Yixuan Tang, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Chao Zhang, Yonghao Tan, Jie Xiao, Yangyu Tao, Jianchen Zhu, Jinbao Xue, Kai Liu, Chongqing Zhao, Xinming Wu, Zhichao Hu, Lei Qin, Jianbing Peng, Zhan Li, Minghui Chen, Xipeng Zhang, Lin Niu, Paige Wang, Yingkai Wang, Haozhao Kuang, Zhongyi Fan, Xu Zheng, Weihao Zhuang, YingPing He, Tian Liu, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, Jingwei Huang, and Chunchao Guo. Hunyuan3D 2.0: Scaling Diffusion Models for High Resolution Textured 3D Assets Generation. *arXiv preprint arXiv:2501.12202*, 2025. URL https://arxiv.org/abs/2501.12202.

Qingnan Zhou and Alec Jacobson. Thingi10K: A dataset of 10,000 3D-printing models. *arXiv preprint arXiv:1605.04797*, 2016. URL https://arxiv.org/abs/1605.04797.

A Density training objective

A.1 Notation

We supervise both the global prediction on the 128^3 grid and the combined prediction on a sampled 64^3 patch at the input resolution. For either supervision grid $\mathcal{Q}$, let $\hat{\ell}_{c,i}$ and $\ell^*_{c,i}$ denote predicted and target log-densities, respectively. The corresponding densities are $\rho = 10^{\min(\ell,7)} - 1$, where the upper cap is a numerical safeguard. We write $\langle\cdot\rangle$ for the mean over the indicated indices, $[a]_+ = \max(a, 0)$, and h for the voxel spacing.

A.2 Density regression

The two main terms regress the predicted log-densities toward the targets. $\mathcal{L}_v$ is a plain L1 error for the volume channel, and $\mathcal{L}_s$ is a weighted L1 error for the surface channel whose weights emphasize the surface band, geometrically complex regions, density peaks, and under-prediction. The volume and surface regression terms are

$$\begin{aligned} \mathcal{L}_v &= \big\langle |\hat{\ell}_{v,i} - \ell^*_{v,i}| \big\rangle_i, \\ \mathcal{L}_s &= \frac{\sum_{i\in\mathcal{Q}} w_i |\hat{\ell}_{s,i} - \ell^*_{s,i}|}{\sum_{i\in\mathcal{Q}} w_i}, \qquad w_i = 1 + b_i + 4g_i + 2p_i + \mathbf{1}[\hat{\ell}_{s,i} < \ell^*_{s,i}]. \end{aligned} \tag{7}$$

Here $b_i = \mathbf{1}[|f_i| \le 2h]$ emphasizes the surface band, and $p_i = (\ell^*_{s,i} / \max(\max_j \ell^*_{s,j}, 10^{-6}))^2$ emphasizes target peaks. The geometric weight is $g_i = \min\{1, \|n_i - \mathcal{B}n_i\|_2 + \min(1, |f_i - \mathcal{B}f_i|/h)\}$, where n_i is the normalized finite-difference SDF gradient and $\mathcal{B}$ is a $3 \times 3 \times 3$ averaging operator.

A.3 Point count and allocation

These terms control how many points are sampled and where the total budget goes, rather than the per-voxel value. To supervise point counts, define $\hat{M}_c = \Delta V \sum_{i\in\mathcal{Q}} \hat{\rho}_{c,i}$, $M^*_c = \Delta V \sum_{i\in\mathcal{Q}} \rho^*_{c,i}$, and $m(M) = \log_{10}(\max(M, 1))$. We use a symmetric mass loss and a one-sided surface-count deficit, which matches the total point count of each channel and penalizes too few surface points:

$$\mathcal{L}_{\text{mass}} = \big\langle H_{0.05}(m(\hat{M}_c) - m(M^*_c)) \big\rangle_c, \qquad \mathcal{L}_{\text{count}} = [m(M^*_s) - m(\hat{M}_s)]_+. \tag{8}$$

Here $H_\beta(a) = a^2/(2\beta)$ for $|a| < \beta$ and $|a| - \beta/2$ otherwise. A normalized-distribution term compares spatial allocation independently of total mass:

$$\mathcal{L}_{\text{dist}} = \frac{\Delta V}{2} \left\langle \sum_{i\in\mathcal{Q}} \left| \frac{\hat{\rho}_{c,i}}{\max(\hat{M}_c, \epsilon)} - \frac{\rho^*_{c,i}}{\max(M^*_c, \epsilon)} \right| \right\rangle_c, \qquad \epsilon = 10^{-12}. \tag{9}$$

A.4 Spatial consistency

$\mathcal{L}_{\text{multi}}$ matches the fields after coarsening, so that errors that average out locally are still penalized at coarser scales, and $\mathcal{L}_{\text{grad}}$ matches the spatial variation of the surface channel. Let $\mathcal{A}_k$ denote k successive $2 \times 2 \times 2$ average-pooling operations with stride two. Coarse-scale agreement and local surface variation are supervised by

$$\begin{aligned} \mathcal{L}_{\text{multi}} &= \frac{1}{2} \sum_{k=1}^{2} \big\langle |\mathcal{A}_k \hat{\ell}_c - \mathcal{A}_k \ell^*_c| \big\rangle_{i,c}, \\ \mathcal{L}_{\text{grad}} &= \frac{1}{3} \sum_{a\in\{x,y,z\}} \big\langle |\delta_a \mathcal{A}_1 \hat{\ell}_s - \delta_a \mathcal{A}_1 \ell^*_s| \big\rangle_i, \end{aligned} \tag{10}$$

where δ_a is an unscaled forward difference along grid axis a.

A.5 MISSING SURFACE DENSITY

Finally, two one-sided terms emphasize missing surface density: because under-sampled surface regions produce dents that post-processing must repair, they penalize only the worst deficits, in the surface band and at several pooled scales. Let $\mathrm{TopMean}_q$ average the largest $\max(1, \lceil qn \rceil)$ of n entries, and let $\mathcal{K} = \{i : b_i = 1 \text{ or } g_i > 0.1\}$, using all of $\mathcal{Q}$ if this set is empty. We define

$$\begin{aligned} \mathcal{L}_{\mathrm{cov}} &= \mathrm{TopMean}_{0.02}\Big(\{[\ell^*_{s,i} - \hat{\ell}_{s,i}]_+ : i \in \mathcal{K}\}\Big), \\ \mathcal{L}_{\mathrm{local}} &= \frac{1}{3}\sum_{k=1}^{3} \mathrm{TopMean}_{0.05}([\log_{10}(1 + \mathcal{A}_k \rho^*_s) - \log_{10}(1 + \mathcal{A}_k \hat{\rho}_s)]_+). \end{aligned} \tag{11}$$

Unlike $\mathcal{L}_{\mathrm{multi}}$, the local term pools linear densities before taking logarithms.

A.6 TOTAL OBJECTIVE

The loss on each supervision grid is

$$\begin{aligned} \mathcal{L}_{\mathcal{Q}} = \mathcal{L}_v + \mathcal{L}_s + \lambda_{\mathrm{mass}}\mathcal{L}_{\mathrm{mass}} + \lambda_{\mathrm{count}}\mathcal{L}_{\mathrm{count}} + \lambda_{\mathrm{dist}}\mathcal{L}_{\mathrm{dist}} \\ + 0.25\mathcal{L}_{\mathrm{multi}} + 0.1\mathcal{L}_{\mathrm{grad}} + 0.5\mathcal{L}_{\mathrm{cov}} + 0.5\mathcal{L}_{\mathrm{local}}. \end{aligned} \tag{12}$$

We use $(\lambda_{\mathrm{mass}}, \lambda_{\mathrm{count}}, \lambda_{\mathrm{dist}}) = (1, 1, 0.25)$ for the global grid and $(0.25, 0.5, 0)$ for the patch. The total objective is

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{global}} + \mathcal{L}_{\mathrm{patch}}, \tag{13}$$

where the patch loss supervises the combined global and local prediction, allowing gradients to update both branches. We average this objective over sampled training shapes and resolutions, with 24-way cube-rotation augmentation.

B TRAINING, HARDWARE, AND TIMING

Training. We train one density network jointly across all three resolutions from random initialization, using AdamW with a global batch size of eight, an initial learning rate of 3×10^{-5}, and weight decay of 10^{-4}. We apply the 24 orientation-preserving cube rotations and use a 1,000-step warmup followed by cosine decay over a 400K step update schedule.

Hardware and timing. For training our density network, we used eight NVIDIA L40S GPUs. For measuring performance on validation and test dataset, we used a common workstation with an Intel Core Ultra 9 275HX CPU, 32 GB of system memory, and an NVIDIA GeForce RTX 5090 Laptop GPU with 24 GB of device memory. Each method is allocated eight CPU cores and 16 GB of system memory, with multithreading enabled where supported; our method additionally uses the GPU. End-to-end timing starts with an SDF in CPU memory and ends with the generated mesh in memory, including post-processing but excluding SDF input loading, final mesh archival, and separate evaluation. For fTetWild, Gmsh, and TetGen, the reported time includes marching-cubes extraction, native input serialization, meshing, and output parsing and validation. Method-specific parameters are described in Appendix C.

C BASELINE CONFIGURATIONS

We follow each baseline's default settings wherever possible, selectively tuning quality and surface-conformity parameters to obtain high-quality meshes in practical time. The following configurations specify these choices.

CGAL (Jamin et al., 2015). We mesh an implicit interior domain with `facet_angle`$= 30°$, `facet_size`$= 0.08$, `facet_distance`$= 0.001$, `cell_radius_edge_ratio`$= 2$, and `cell_size`$= 0.12$. We set the implicit-surface intersection tolerance to an absolute distance of 3×10^{-4} in input coordinates, three tenths of `facet_distance`, so that the error in locating the isosurface along a query segment stays well below the surface approximation bound we ask the mesher to satisfy. The updated configuration enables `manifold()` and uses parallel processing with at most eight threads.

fTetWild (Hu et al., 2020). We retain the default envelope tolerance $10^{-3}b$, ideal edge length $0.05b$, and optimization settings, where b is the input bounding-box diagonal. We enable `--manifold-surface`, set `--max-threads 8`, and suppress console output with `--is-quiet`.

TetGen (Si, 2015). We use `-pqY`: tetrahedralization of the input piecewise-linear complex, quality refinement with the default radius–edge ratio bound of 2, and preservation of the input surface mesh. We supply no additional volume or dihedral-angle bound. We preserve the input surface because letting TetGen remesh it left many cases unfinished within our time budget, which sharply increased the failure rate. Surface preservation can limit the quality attainable by refinement. TetGen exposes no parallel option, so it runs on a single thread despite the eight cores allocated to it.

Gmsh (Geuzaine & Remacle, 2009). We retain the default three-dimensional Delaunay algorithm (`Mesh.Algorithm3D= 1`), sizing rules, and optimization settings. We set `Mesh.MeshOnlyEmpty= 1` to retain the imported marching-cubes surface, without surface classification or reparametrization, for the same reason: when Gmsh remeshes the surface instead, many cases do not finish within the same budget and are recorded as failures. The general and dimension-specific thread limits are set to eight. Gmsh parallelizes three-dimensional meshing only in its HXT algorithm, which this configuration does not select, so raising the thread limits leaves tetrahedralization effectively single-threaded as well.