Marching Neurons:
Accurate Surface Extraction for Neural Implicit Shapes

CHRISTIAN STIPPEL, TU Wien, Austria
FELIX MUJKANOVIC, Max-Planck-Institute for Informatics, Germany
THOMAS LEIMKÜHLER, Max-Planck-Institute for Informatics, Germany
PEDRO HERMOSILLA, TU Wien, Austria

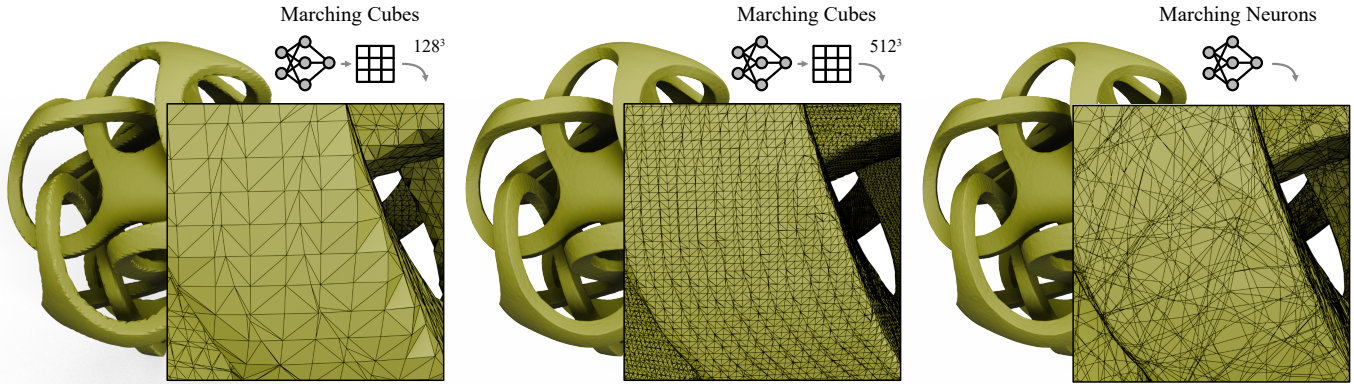

Fig. 1. Surfaces extracted from a signed distance function (SDF) represented by a neural network using Marching Cubes with different grid resolutions (left and center) compared to our analytic method (right). While Marching Cubes struggles to reconstruct sharp edges even at high grid resolutions, our analytic method is able to reconstruct the surface accurately.

Accurate surface geometry representation is crucial in 3D visual computing. Explicit representations, such as polygonal meshes, and implicit representations, like signed distance functions, each have distinct advantages, making efficient conversions between them increasingly important. Conventional surface extraction methods for implicit representations, such as the widely used Marching Cubes algorithm, rely on spatial decomposition and sampling, leading to inaccuracies due to fixed and limited resolution. We introduce a novel approach for analytically extracting surfaces from neural implicit functions. Our method operates natively in parallel and can navigate large neural architectures. By leveraging the fact that each neuron partitions the domain, we develop a depth-first traversal strategy to efficiently track the encoded surface. The resulting meshes faithfully capture the full geometric information from the network without ad-hoc spatial discretization, achieving unprecedented accuracy across diverse shapes and network architectures while maintaining competitive speed.

ACM Reference Format:
Christian Stippel, Felix Mujkanovic, Thomas Leimkühler, and Pedro Hermosilla. 2025. Marching Neurons: Accurate Surface Extraction for Neural Implicit Shapes. *ACM Trans. Graph.* 44, 6, Article 222 (December 2025), 12 pages. https://doi.org/10.1145/3763328

Authors' Contact Information: Christian Stippel, TU Wien, Austria; Felix Mujkanovic, Max-Planck-Institute for Informatics, Germany; Thomas Leimkühler, Max-Planck-Institute for Informatics, Germany; Pedro Hermosilla, TU Wien, Austria.

ACM 1557-7368/2025/12-ART222
https://doi.org/10.1145/3763328

1 Introduction

Accurately representing surface geometry is a cornerstone of 3D visual computing and beyond [De Berg 2000; Watt 1999]. A continually growing repertoire of representations exists, broadly divided into two categories: *Explicit* representations, which directly describe the surface using geometric elements such as polygons or points; and *implicit* representations, which define the surface indirectly as the set of points satisfying a mathematical equation, such as level sets, with signed distance functions (SDFs) being a prominent example. The choice of representation is typically guided by the downstream algorithms and applications that use the geometry. For example, explicit representations are well-suited for classical tasks such as rendering [Akenine-Moller et al. 2019] and editing [Botsch et al. 2010], while implicit representations excel in gradient-based geometry optimization, a core operation in modern neural workflows [Park et al. 2019; Wang et al. 2021]. In this work, we propose a novel, accurate, and flexible technique for analytically converting a neural implicit shape into an explicit surface.

The vast majority of existing techniques for surface extraction from implicit representations rely on spatial decomposition and sampling [De Araújo et al. 2015]. The most commonly used method in this domain is Marching Cubes [Lorensen and Cline 1987], which samples the implicit function on a regular grid and uses linear interpolation to approximate the desired level set. While popular for its simplicity and agnostic to the implicit function representation, this method's accuracy is limited by the chosen sampling resolution and interpolation scheme. Many follow-up works have further developed the basic approach [Newman and Yi 2006], e.g.

by pruning samples using hierarchical structures [Wilhelms and Van Gelder 1992], moving to tetrahedral meshes to enable irregular sample placement [Doi and Koide 1991], optimizing the sample locations [Shen et al. 2023], or employing more advanced interpolation strategies [Ju et al. 2002; Sellán et al. 2024]. Yet, for all of these approaches, a fundamental problem remains: Reconstructing complex geometry from a finite set of samples inevitably introduces inaccuracies.

The surge of deep learning [Goodfellow et al. 2016] over the past decade has advanced not only signal processing but also signal representation. In particular, neural fields [Sitzmann et al. 2019; Xie et al. 2022], which use coordinate-based networks for signal representation, have become a powerful and ubiquitous paradigm for continuous functions, such as surface implicits [Park et al. 2019]. The key insight of this work is that these neural representations enable *efficient analytic surface extraction* without the need for adhoc sampling.

We propose a novel method for extracting a surface from an implicit function represented by a neural network. We leverage the fact that a (deep) composition of piecewise linear functions remains piecewise linear, with each layer subdividing existing regions [Montufar et al. 2014] and shaping the target function through convex linear regions separated by hyperplanes. This enables us to traverse the deep network neuron by neuron ("marching") in a depth-first fashion, identifying progressively finer regions that contain the desired surface. Although the number of linear regions increases exponentially with network depth [Montufar et al. 2014], most do not contain the surface. Using range analysis [Sharp and Jacobson 2022] to eliminate empty regions, our approach efficiently traverses ReLU-based neural architectures. This process produces a polygonal mesh that accurately captures the encoded surface geometry.

Unlike previous work on analytic surface extraction [Berzins 2023; Lei and Jia 2020; Lei et al. 2021], our approach is not only adaptively accurate – extracting surfaces at the level of the underlying neural field – but also capable of extracting multiple disconnected shapes while maintaining competitive reconstruction speed.

In summary, our contributions are:

- A novel method for analytically extracting a polygonal mesh from an implicit neural representation.
- An algorithm that is natively parallel and easy to implement.
- Capabilities and accuracy that significantly exceed the current state of the art.

We provide all source code and datasets in our project page.

2 Related Work

2.1 The Geometry of Neural Networks

Deep neural networks are universal function approximators in theory [Hornik et al. 1989] and demonstrate the ability to represent complex functions across diverse domains in practice [Goodfellow et al. 2016]. This remarkable flexibility arises from their compositional structure, which enables exponential expressivity with respect to the number of layers [Montufar et al. 2014], allowing virtually any data topology to be handled [Naitzat et al. 2020]. Networks with ReLU activation functions are particularly well-suited for analyzing this property [Hanin and Rolnick 2019; Pascanu et al. 2014; Raghu et al. 2017], as they model continuous piecewise linear functions. These networks partition the input domain into polyhedral regions, formed by an arrangement of folded hyperplanes that correspond to the decision boundaries of neurons [Grigsby and Lindsey 2022; Vallin et al. 2023]. We propose a novel algorithm for tracing the geometry of a neural network to extract an explicit representation from the implicit function it encodes.

Algorithms for analytically extracting the polyhedral complex of a ReLU network typically rely on mixed-integer linear programming [Serra et al. 2018], neuron state flipping [Lei and Jia 2020; Lei et al. 2021], or recursive intersection and cutting [Humayun et al. 2023; Wang 2022], with applications in, e.g. , safety verification [Vincent and Schwager 2021] and robustness [Hein et al. 2019]. However, the computational complexity of these approaches is typically substantial due to the combinatorial explosion in the number of polyhedra. Addressing this problem, Berzins [2023] proposes increasing efficiency by eliminating redundancy through an edge-centric approach. Our key insight is that, for the critical task of surface extraction from neural implicit representations, a massively parallel implementation can be achieved by a bespoke depth-first traversal of the network. This approach results in accuracy that surpasses the state of the art, while maintaining competitive speed.

2.2 Iso-Surface Extraction

The representation of surfaces as level sets of implicit functions has a rich history across various scientific disciplines and decades [Bloomenthal et al. 1997; Osher et al. 2004]. Extracting an explicit (typically polygonal) surface from this representation is a well-studied problem for which several classes of algorithms have been developed [De Araújo et al. 2015].

While surface tracking [Hilton et al. 1996] and shrink-wrapping approaches [Hanocka et al. 2020; Stander and Hart 1997; Van Overveld and Wyvill 2004] have been applied with some success, the majority of research focuses on spatial decomposition techniques [Bloomenthal 1988]. Here, space is divided into cells, and polygons are constructed within each cell containing the surface, typically by interpolating discrete samples. The popular Marching Cubes algorithm [Lorensen and Cline 1987; Wyvill et al. 1986] and numerous follow-up works [Chernyaev 1995; Hege et al. 1997; Montani et al. 1994; Newman and Yi 2006; Wilhelms and Van Gelder 1992] use a regular grid of cubes. The rigidity of this structure has been relaxed to allow for more adaptive discretizations, e.g. by using tetrahedra [Doi and Koide 1991; Ren et al. 2025]. Unlike the sampling-based techniques, our method determines the geometry analytically.

Dual representations have been shown to outperform traditional approaches by offering more flexible and accurate surface reconstruction, especially in handling sharp features and complex topologies [Azernikov and Fischer 2005; Ju et al. 2002; Nielson 2004]. Furthermore, replacing hand-crafted extraction rules with learned ones [Chen et al. 2022; Chen and Zhang 2021] or adopting more sophisticated interpolation schemes [Kohlbrenner and Alexa 2025; Sellán et al. 2023; Sellán et al. 2024] can lead to significant quality improvements. By using an analytical approach, our algorithm eliminates the need for any sampling and interpolation schemes.

With the proliferation of neural fields [Sitzmann et al. 2019; Xie et al. 2022], surface-encoding implicit functions are now commonly represented using neural networks [Park et al. 2019]. Due to their closed-form nature, analytic surface extraction is feasible; however, the complexity of the underlying geometry presents significant challenges. Analytic Marching [Lei and Jia 2020; Lei et al. 2021] addresses this problem by explicitly enumerating the polyhedral cells that contain the surface, recursively visiting neighboring cells. However, this approach faces challenges with multiple disconnected components. Berzins [2023] addresses this problem by relying on an edge-centric approach instead of linear regions. However, Berzins struggles with complex architectures due to the lack of a filtering mechanism for empty regions, producing at the same time an excessive number of polygons. In contrast, we propose a method capable of extracting shapes that can consist of multiple components while maintaining competitive speed.

3 Background

In this section, we introduce the concepts and notation for neural implicit shape representations relevant to our approach.

We focus on solid shapes $\mathcal{S}$ represented as the level set, or iso surface, of an implicit function $f \in \Omega \to \mathbb{R}$ [Bloomenthal et al. 1997], with $\Omega \subset \mathbb{R}^3$. Without loss of generality, we consider the zero level set (Fig. 2):

$$\mathcal{S} := \{\mathbf{x} \in \Omega \mid f(\mathbf{x}) = 0\}. \tag{1}$$

This general setting encompasses several widely used special cases, including signed distance fields (SDFs) [Osher et al. 2004; Park et al. 2019], unsigned distance fields [Chibane et al. 2020], and their variations, such as truncated [Curless and Levoy 1996], distorted [Seyb et al. 2019], or otherwise weaker forms [Marschner et al. 2023; Sharp and Jacobson 2022].

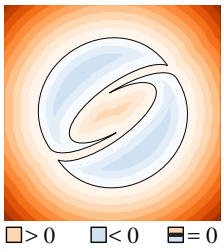

Fig. 2. An implicit function and its zero level set.

Nowadays, a widely adopted approach to representing f is through a neural field f_θ, i.e. a neural architecture with trainable parameters θ, optimized via gradient descent [Xie et al. 2022]. In the simplest case, f_θ is represented by a Multilayer Perceptron (MLP). The number of neurons in each layer $l \in \{1, \dots, L\}$ is denoted as d_l. Given a weight matrix $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$, a bias vector $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$, and a non-linear activation function $\sigma^{(l)} \in \mathbb{R} \to \mathbb{R}$ for each layer, the network recursively applies

$$\mathbf{p}^{(l)} = W^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \qquad \mathbf{q}^{(l)} = \sigma^{(l)}\left(\mathbf{p}^{(l)}\right), \tag{2}$$

where σ is applied element-wise. We set $\mathbf{q}^{(0)} := \mathbf{x}$ and $f_\theta(\mathbf{x}) := \mathbf{p}^{(L)}$ and refer to $\mathbf{p}_i^{(l)}$ and $\mathbf{q}_i^{(l)}$ as the i-th *pre-activation neuron* and *post-activation neuron*, respectively.

While there is an overwhelming variety of nonlinear activation functions available [Kunc and Kléma 2024], we consider the special case of *piecewise linear*[1] activation functions for now. Given the recursive structure of Eq. 2, f_θ is a composition of piecewise linear functions and is therefore itself piecewise linear. Each call of σ potentially subdivides the domain of f_θ into an increasingly larger number of convex linear regions, separated by planes [Hanin and Rolnick 2019; Montufar et al. 2014], with each region referred to as a cell. Fig. 3 illustrates this property in 2D using arguably the most common choice in this space: the rectified linear unit (ReLU):

$$\sigma(\mathbf{p}) = \max(0, \mathbf{p}). \tag{3}$$

In this context, we refer to a post-activation neuron $\mathbf{q}_i^{(l)}$ as *active* if it is positive and *inactive* if it is zero. The cells of f_θ are separated by the set of planes $\left\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{p}_i^{(l)}(\mathbf{x}) = 0\right\}$, which correspond to the locations where a neuron switches from inactive to active. Within each cell, the active/inactive pattern of neurons in the entire network remains fixed, implying that f_θ is linear within each cell and can be represented by collapsing the corresponding submatrices of $W^{(l)}$ and subvectors of $\mathbf{b}^{(l)}$ across layers. Similar observations hold for any choice of σ that is piecewise linear.

4 Method

Given an implicit neural representation f_θ with ReLU activation functions, we aim to extract its zero level set $\mathcal{S}$ as a polygonal mesh. Importantly, our goal is an *analytic* extraction that captures all the details encoded in f_θ, unlike the widely used sampling-based methods [Lorensen and Cline 1987].

For piecewise-linear activation functions, f_θ can be viewed as a collection of linear branches [Lei and Jia 2020; Lei et al. 2021]. Our key insight is that by adaptively subdividing the domain and tracking activation patterns, we can efficiently reduce the network to an explicit piecewise-linear function in each cell, represented as a polyhedral mesh. Using range analysis [Sharp and Jacobson 2022], we can discard many cells early on, as they do not contain the zero level set, leading to significant efficiency gains. As a result, we obtain an accurate solution for the zero iso-surface in the form of a polygonal mesh.

In Sec. 4.1, we introduce the basic structure of our approach using a 2D domain, before extending the method to the 3D setting in Sec. 4.2. Finally, we provide implementation details in Sec. 4.3.

4.1 Level Set Extraction for 2D Networks

For $\Omega \subset \mathbb{R}^2$, each neuron can define a polyline that partitions Ω into two regions, resulting in a collection of convex polygonal cells (Fig. 3). We propose a novel scheme that explicitly tracks these cells by a depth-first traversal of f_θ. As we traverse f_θ, the cells, with layer-specific geometries and internal states, undergo transformations and splitting, and may also be discarded. The goal of this process is to identify the set of cells containing the zero level set of f_θ, enabling its explicit representation as one or more closed line strips to be extracted.

Each polygonal cell at layer l is represented as the tuple

$$C^{(l)} = \left\{\widetilde{V}; \widetilde{W}, \widetilde{\mathbf{b}}\right\}, \tag{4}$$

where $\widetilde{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m] \in \mathbb{R}^{2\times m}$ specifies the 2D coordinates of the m vertices $\mathbf{v}_j \in \mathbb{R}^2$ forming the polygon. Further, $\widetilde{W} \in \mathbb{R}^{d_l \times 2}$

[1] For simplicity, we use "linear" for both linear and affine functions.

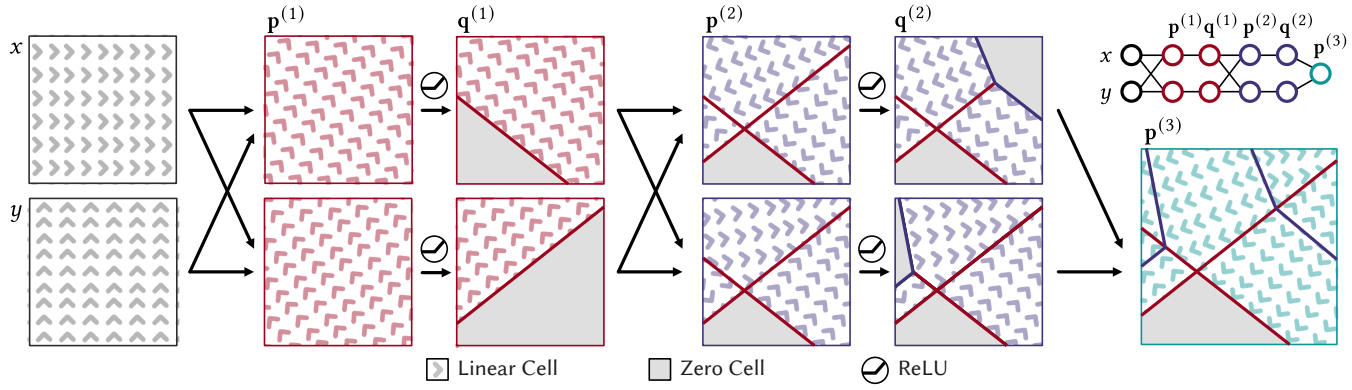

Fig. 3. The geometry of ReLU MLPs, illustrated here for a 2D input domain with two hidden layers containing two neurons each. In each layer l, the pre-activations $\mathbf{p}^{(l)}$ are computed as linear combinations of the neurons from the previous layer. The ReLU activation then produces post-activations $\mathbf{q}^{(l)}$, introducing lines that subdivide the existing linear regions. As a result, the network's output is composed of convex cells within which it remains linear, i.e. we can collapse all corresponding weight matrices and biases.

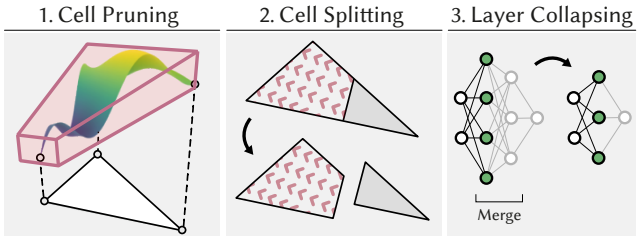

Fig. 4. The three steps of our network traversal. First, we bound the network response within each cell (purple volume) to eliminate those that do not intersect the zero level set. Next, we split the remaining cell to capture the nonlinearities introduced by piecewise linear activation functions. Finally, the current layer (green neurons) is combined with the next layer. The entire process (1. - 3.) repeats until the final network layer is reached.

and $\widetilde{\mathbf{b}} \in \mathbb{R}^{d_l}$ are the parameters defining the linear function

$$\widetilde{\mathbf{p}}(\mathbf{x}) = \widetilde{W}\mathbf{x} + \widetilde{\mathbf{b}} \tag{5}$$

inside the polygon by combining the linear functions contributing up to this layer, corresponding to the pre-activation response of the current layer. For any hidden layer l, $\widetilde{\mathbf{p}}(\mathbf{x})$ produces a d_l-dimensional vector, where each component corresponds to a neuron (rows in Fig. 3). In the final layer L, $\widetilde{\mathbf{p}}(\mathbf{x})$ outputs a scalar ($d_L = 1$) that represents the linear function describing the full f_θ within the cell.

Without loss of generality, we assume a rectangular domain Ω and initialize the cell set with a single quadrilateral, $\left\{C_1^{(0)}\right\}$, covering the entire domain. The corresponding function $\widetilde{\mathbf{p}}(\mathbf{x})$ is initialized as the identity function, with $\widetilde{W} = \mathbb{I}_{2\times 2}$ and $\widetilde{\mathbf{b}} = \mathbf{0}$.

From this point, we traverse the layers of f_θ, iteratively updating the cell set until only the final-layer cells containing the zero level set remain. Progressing through layers, we prune or split cells while updating their parameters, as detailed below and shown in Fig. 4.

4.1.1 Step 1: Cell Pruning. As a first step, we eliminate cells that do not intersect the zero level set of f_θ. Since the number of linear regions within the domain Ω grows exponentially with each layer [Montufar et al. 2014], and most do not contain the zero level set, this step is crucial for maintaining efficiency. We use range analysis [Duff 1992; Rump and Kashiwagi 2015] on the remaining layers to accomplish this task. Specifically, we follow Sharp and Jacobson [2022] and compute affine bounds [Comba and Stolfi 1993] of f_θ within each cell $C_i^{(l)}$. This gives us a conservative estimate of the minimum and maximum values of f_θ within $C_i^{(l)}$. If

$$\min_{\mathbf{x}\in C_i^{(l)}} f_\theta(\mathbf{x}) > 0 \quad \text{or} \quad \max_{\mathbf{x}\in C_i^{(l)}} f_\theta(\mathbf{x}) < 0, \tag{6}$$

we discard $C_i^{(l)}$ because it does not intersect the zero level set. In practice, we query the affine bounds using the cell's axis-aligned bounding box. This may overestimate ranges for degenerated cells, for example, ones that have a long, not axis-aligned diagonal; however, it suffices to ensure scalability to larger networks.

4.1.2 Step 2: Cell Splitting. In this step, we first identify neurons in the current layer l that divide the polygon $C_i^{(l)}$ into two smaller linear regions. We refer to these neurons as *critical* and observe that their pre-activation responses $\widetilde{\mathbf{p}}(\mathbf{x})$ change sign within the polygon, leading to the non-linear kink of the subsequent ReLU activation occurring inside the polygon. Since we are dealing with linear functions, the vertices $\mathbf{v}_j$ naturally correspond to the locations of the extrema of $\widetilde{\mathbf{p}}(\mathbf{x})$ within the polygon. Therefore, to check for criticality, it suffices to evaluate $\widetilde{\mathbf{p}}$ at the vertices. Specifically, neuron i is considered critical if

$$\min_j \left(\widetilde{\mathbf{p}}(\mathbf{v}_j)\right)_i < 0 \quad \text{and} \quad \max_j \left(\widetilde{\mathbf{p}}(\mathbf{v}_j)\right)_i > 0. \tag{7}$$

If one or more critical neurons are associated with the polygon, it is split along the (linear) zero level set of the corresponding output dimensions of $\widetilde{\mathbf{p}}(\mathbf{x})$ with the Sutherland-Hodgman [1974] algorithm. New vertices are inserted on the intersection of polygon edges with the cutting plane, obtaining two sub-polygons separated by the neuron. Polygons that do not need to be split are retained unchanged.

4.1.3 Step 3: Layer Collapsing. As we move to the next layer $l+1$, it is necessary to update $\widetilde{W}$ and $\widetilde{\mathbf{b}}$. Recall that only neurons active

within a cell contribute their linear functions to it. To track this, we define a binary mask $\mathbf{m} \in \mathbb{R}^{d_l}$, where each entry indicates whether the corresponding neuron in the current layer l is active within the cell. Using this mask, we update the function parameters of the cell as follows:

$$\begin{aligned} \widetilde{W} &:= W^{(l+1)} \operatorname{diag}(\mathbf{m})\, \widetilde{W} \\ \widetilde{\mathbf{b}} &:= W^{(l+1)} \left(\mathbf{m} \odot \widetilde{\mathbf{b}}\right) + \mathbf{b}^{(l+1)}, \end{aligned} \tag{8}$$

where $\operatorname{diag}(\cdot)$ converts a vector into its diagonal matrix representation, and $\odot$ denotes the Hadamard product. The updated cells form the new set $\left\{C_i^{(l+1)}\right\}$, which will be processed by step 1 again.

4.1.4 Network Traversal. The three steps outlined above specify how cells should be updated when transitioning from one layer to the next, providing flexibility in choosing the global traversal scheme. A straightforward approach processes all cells in a layer before moving to the next, corresponding to a breadth-first traversal. However, this approach has the disadvantage of requiring more memory than is typically available on a contemporary GPU. In the worst case, every neuron splits all active cells. A breadth-first traversal must then hold the full frontier of all cells at once, resulting in space requirements of $O(2^d)$, where d is the number of neurons. Therefore, we choose a depth-first traversal where we process cells in a last-in-first-out principle to reduce the memory burden. To fully utilize the GPU, we schedule the traversal to maintain a sufficient number of cells processed in parallel in each step. In this case, we only store the current branches of the subdivision tree, resulting in $O(bd)$, where b is the number of cells processed in parallel.

In practice, the average space complexity is lower: not every neuron splits every cell, and the likelihood of a split decreases the more often a cell has already been subdivided. Additionally, range-analysis pruning removes many cells before they would be split.

4.1.5 Level Set Extraction. Once all cells have reached the final layer, the set $\left\{C_i^{(L)}\right\}$ contains the cells that cover the piecewise linear regions of the level set of f_θ. At this point, the linear level set $\mathcal{S} = \{\mathbf{x} \mid \widetilde{\mathbf{p}}(\mathbf{x}) = 0\}$ can be analytically extracted from each cell. This is straightforward as $\widetilde{\mathbf{p}}$ is a linear function. The union of the extracted per-cell level sets forms the desired explicit representation as a line strip.

4.2 Extension to 3D

Extending our approach to 3D, where $\mathbf{x} \in \Omega \subset \mathbb{R}^3$, is straightforward: operations on polygons are adapted to handle *3D polyhedra*. Specifically, we now maintain m 3D vertices, $\widetilde{V} \in \mathbb{R}^{3\times m}$, and the per-cell function parameter $\widetilde{W} \in \mathbb{R}^{d_l \times 3}$ accounts for the additional dimension. Critical neurons now split polyhedra along *planes*. Once the final layer L of f_θ is reached, we analytically extract the zero level set from each polyhedron, producing the polygonal mesh that explicitly represents the encoded surface $\mathcal{S}$. The mesh is then tessellated into triangles for compatibility with standard pipelines.

4.3 Implementation Details

We realized our approach using a reasonably optimized JAX [Bradbury et al. 2018] implementation that takes advantage of our natively parallel algorithm design. All active cells reside in a shared buffer with indexing stacks for each operation. To prevent numerical inaccuracies that can occasionally arise from our deep recursive subdivision scheme, we found a mixed-precision implementation to be essential. Specifically, while cell pruning can be safely executed with 32-bit floating-point precision, we use 64-bit precision for cell splitting and layer collapsing. However, this does not impose any restrictions on the bit depth of the input network f_θ.

5 Evaluation

In this section, we describe the experiments conducted to evaluate our method. We compare the quality of the meshes extracted by our algorithm with those reconstructed by other methods. Moreover, we also measure the time required for the reconstruction of the resulting meshes and the average number of triangles generated. Additionally, we also evaluate the quality of the meshes generated and the effect of mesh simplification algorithms on the reconstruction. Lastly, we evaluate the effect of our filtering step and extend our approach to arbitrary activation functions. We assume that all shapes are defined by the zero-level set of an SDF encoded by a neural network.

5.1 Experimental Setup

Metrics. To measure the error introduced by the reconstruction algorithms, we compare the extracted mesh directly to the SDF encoded within the neural network. We define two metrics to measure such error: Soft-Precision (SP) and Soft-Recall (SR). Soft-Precision aims to quantify how far the reconstructed mesh is from the zero-level set defined by the SDF. This metric is computed by sampling 2^{20} points on the surface of the reconstructed mesh, evaluating the SDF at these point locations, and taking the mean absolute value. If the reconstructed surface lies in the zero-level set of the SDF, Soft-Precision will be exactly zero.

However, Soft-Precision does not capture cases in which large portions of the zero-level set are not reconstructed. To remedy this, we introduce a second metric, Soft-Recall. First, we sample 2^{20} points on the surface of the original mesh from our dataset. Then, we use gradient descent to move such points to the zero-level set of the SDF. Once converged, we measure the average distance of the resulting coordinates to the reconstructed mesh. If the method is able to perfectly reconstruct the complete zero-level set, Soft-Recall will be exactly zero.

Additionally, we compare the time required to perform the extraction and the number of resulting triangles.

Datasets. We evaluate our method and all other baselines on 84 watertight shapes collected from five different datasets: 20 shapes from Thingi10K [Zhou and Jacobson 2016], 20 shapes from the ABC Dataset [Koch et al. 2019], 19 shapes from ShapeNet [Chang et al. 2015], 20 shapes from FAUST [Bogo et al. 2014], and 5 shapes from the Stanford 3D Scanning Repository [Curless and Levoy 1996]. Fig. 5 depicts some samples from our dataset. Then, for each shape, we fit neural SDFs with all architectures.

SDF Training Protocol. Before training, meshes are converted into an SDF. Each mesh's bounding box is normalized to the range $[-.95, .95]^3$, and then densely sampled to produce 10^7 training points per shape. For non-ShapeNet meshes, 2^{20} samples ($\approx$ 6% of the total)

Table 1. **Mesh extraction.** Soft Precision (SP) and Soft Recall (SR) are multiplied by 10^6, triangles divided by 10^3. Runtime is measured in seconds.

	d4_w128				d4_w256			
	SP	SR	Runtime	Triangles	SP	SR	Runtime	Triangles
Approx. - 64^3								
•Marching Cubes	1682.28	10293.05	0.01	14.44	1802.69	9832.66	0.01	14.21
•Dual Contouring	1278.87	10044.56	0.20	14.45	1412.82	9530.62	0.20	14.23
•Hier. Marching Cubes	1686.56	10352.45	4.77	14.42	1814.43	9982.32	5.72	14.18
•Reach for the Arcs	4072.33	3144.99	538.65	355.01	3480.32	2851.51	481.15	349.09
Approx. - 128^3								
•Marching Cubes	518.48	1373.29	0.04	62.18	551.65	1248.17	0.08	60.89
•Dual Contouring	425.45	1319.47	1.17	62.13	461.12	1207.23	1.22	60.90
•Hier. Marching Cubes	543.82	1420.48	8.83	62.11	561.32	1269.78	11.69	60.76
•Reach for the Arcs	6302.02	2810.83	5726.60	310.95	4826.59	2406.64	4696.52	337.86
Approx. - 256^3								
•Marching Cubes	162.36	356.69	0.30	258.50	175.50	330.28	0.62	252.57
•Dual Contouring	126.58	327.90	6.92	258.38	134.16	301.52	7.07	252.47
•Hier. Marching Cubes	221.47	436.62	11.35	259.51	196.49	363.15	16.81	252.52
•Reach for the Arcs	–	–	–	–	–	–	–	–
Approx. - 512^3								
•Marching Cubes	53.61	113.88	2.36	1048.80	60.33	101.94	4.91	1025.29
•Dual Contouring	41.75	101.98	48.56	1048.69	45.24	88.62	50.33	1025.15
•Hier. Marching Cubes	150.22	235.37	15.05	1060.95	103.01	153.12	21.54	1028.08
•Reach for the Arcs	–	–	–	–	–	–	–	–
Exact								
•Analytic Marching	**0.03**	675.76	3.21	425.92	**0.02**	188.80	32.99	2306.06
•Edge Subdivision	**0.03**	0.61	46.72	429.08	**0.02**	0.90	21147.39	2321.70
•Ours	**0.03**	**0.07**	11.84	429.08	**0.02**	**0.03**	169.40	2321.70

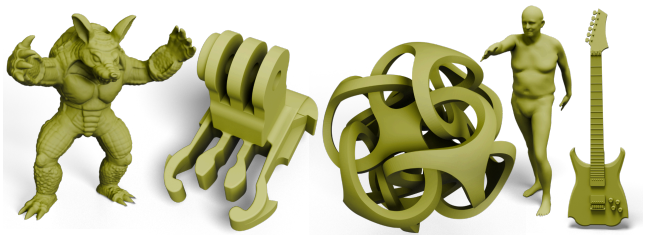

Fig. 5. Visualization of samples from our dataset. Our dataset covers a large range of mesh complexities, from simple CAD objects composed of a few thousand triangles to highly detailed shapes with millions of triangles.

are drawn uniformly from the bounding box using a Sobol sequence, while the remaining points (≈ 94%) are concentrated near the surface. These are split equally into on-surface points and near-surface points (each ≈ 47% of the total). The near-surface samples are created by adding Gaussian noise with $\sigma = 0.05$ to surface points. Signed distances are computed for the uniform and near-surface samples using Open3D's raycasting [Zhou et al. 2018] with 9 evaluation rays per query, while the on-surface samples are assigned distance zero. For ShapeNet meshes, which are often non-watertight, DeepSDF's dedicated preprocessing tool [Park et al. 2019] is used to generate the same total number of samples distributed across inside and outside regions. Networks are trained with the Adam [Kingma and Ba 2015] optimizer at a fixed learning rate of 10^{-4} and a batch size of 10^4. Batches are sampled with replacement from the full dataset. Training runs for four hours.

Baselines. We compare our method to several baselines that approximate the mesh surface by spatial decomposition and sampling of the SDF, and other exact methods like ours. In particular, we select Marching Cubes [Lorensen and Cline 1987], Dual Contouring [Ju et al. 2002], Hierarchical Marching Cubes [Sharp and Jacobson 2022], and Reach for the Arcs [Sellán et al. 2024] as representative baselines for approximation methods. For such methods, we use a grid resolution of 64, 128, 256 and 512. Moreover, we compare to two existing exact methods: Analytic Marching [Lei et al. 2021] and Edge Subdivision [Berzins 2023].

Neural Architectures. We chose a neural architecture commonly used to encode SDFs: a ReLU multi-layer perceptron (MLP). In particular, we selected two variants: a ReLU MLP with four layers and 128 neurons each (d4_w128), and a more complex architecture also composed of four layers but with 256 neurons each (d4_w256).

5.2 Main Results

Tab. 1 presents the main result of our comparison. Naturally, all approximation methods produce meshes with high SP and SR when

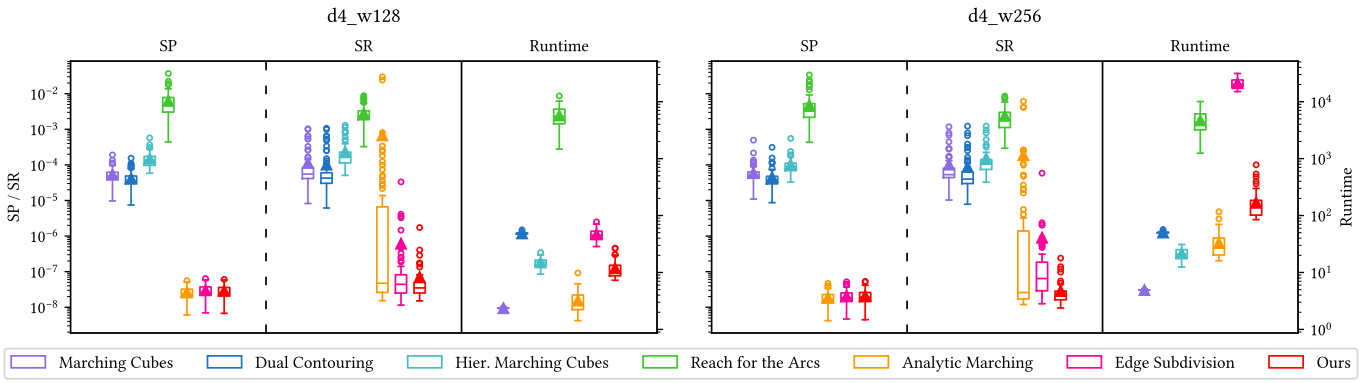

Fig. 6. Box plots for soft precision (SP), soft recall (SR), and runtime per architecture and method. Little triangles indicate the means. Runtime is measured in seconds. Most approximation methods use a grid of resolution 512, while Reach for the Arcs uses 128.

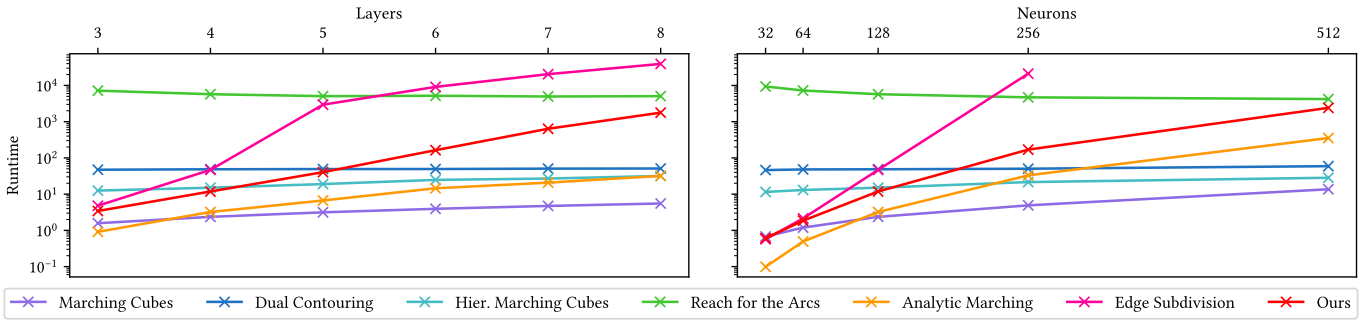

Fig. 7. Scalability study with runtime measured in seconds. We vary the depth and width of the d4_w128 architecture and extract meshes with every method.

low-resolution sampling grids are used. Employing higher-resolution grids decreases these metrics, but leads to an increase in processing time and generated triangles. Yet, even for the finest grids with resolution 512, the reconstructed meshes significantly deviate from the SDF. Moreover, the recent method Reach for the Arcs [Sellán et al. 2024] is only able to process low-resolution grids due to its long processing times.

Analytic methods, on the other hand, produce meshes with almost perfect SP. Regarding SR, however, Analytic Marching [Lei et al. 2021] exhibits high metrics as it misses disconnected parts of the mesh. This is the result of the nature of their algorithm, which relies on a set of seed points from which the mesh is reconstructed iteratively. In contrast, this is not the case for Edge Subdivision [Berzins 2023], which produces meshes with low SR. Unfortunately, it takes a long time to reconstruct large networks. When we look at the distribution of these metrics over the different shapes in our dataset in Fig. 6, analytic methods present high variability in SR, indicating that they struggle with certain shapes. Our method, on the other hand, produces more accurate meshes with fewer triangles than other analytic methods, while maintaining a reasonable runtime comparable to approximation methods.

Fig. 8 provides qualitative results of several meshes reconstructed by all baselines. These show that our method is able to produce more accurate results than approximation methods, which struggle to reconstruct sharp edges, and than Analytic Marching [Lei and Jia 2020; Lei et al. 2021], which fails to reconstruct certain disconnected components. Additionally, Fig. 8 also shows that reconstructions produced by Edge Subdivision [Berzins 2023] result in denser meshes with more triangles.

5.3 Scalability

We also evaluate how the runtime of each method scales for network architectures with increasing numbers of layers and neurons in Fig. 7. For most approximation methods, the runtime slightly increases with both the depth and width of the network, since inference becomes more costly. For Reach for the Arcs [Sellán et al. 2024], this effect is drowned by the method's generally long runtime. As expected, analytic methods exhibit a steeper increase in runtime as we increase the network size. Among them, Edge Subdivision [Berzins 2023] is particularly sensitive to the number of neurons, and fails to process a network with width 512. On the other hand, both Analytic Marching [Lei and Jia 2020; Lei et al. 2021] and our method scale more gracefully with the number of neurons, while ours unfortunately scales worse with the number of layers. Still, these results show that our method is able to process a large range of network architectures in a reasonable runtime similar to Analytic Marching [Lei and Jia 2020; Lei et al. 2021] while being substantially more accurate.

Fig. 8. Qualitative results of all the baselines for two network architectures with different numbers of neurons.

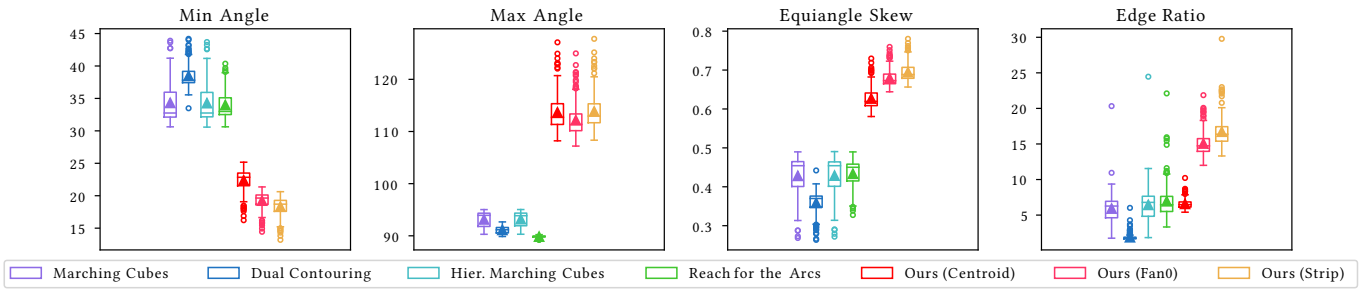

Fig. 9. Evaluation of the quality of the triangle meshes generated by different reconstruction methods using the d4_w256 architecture.

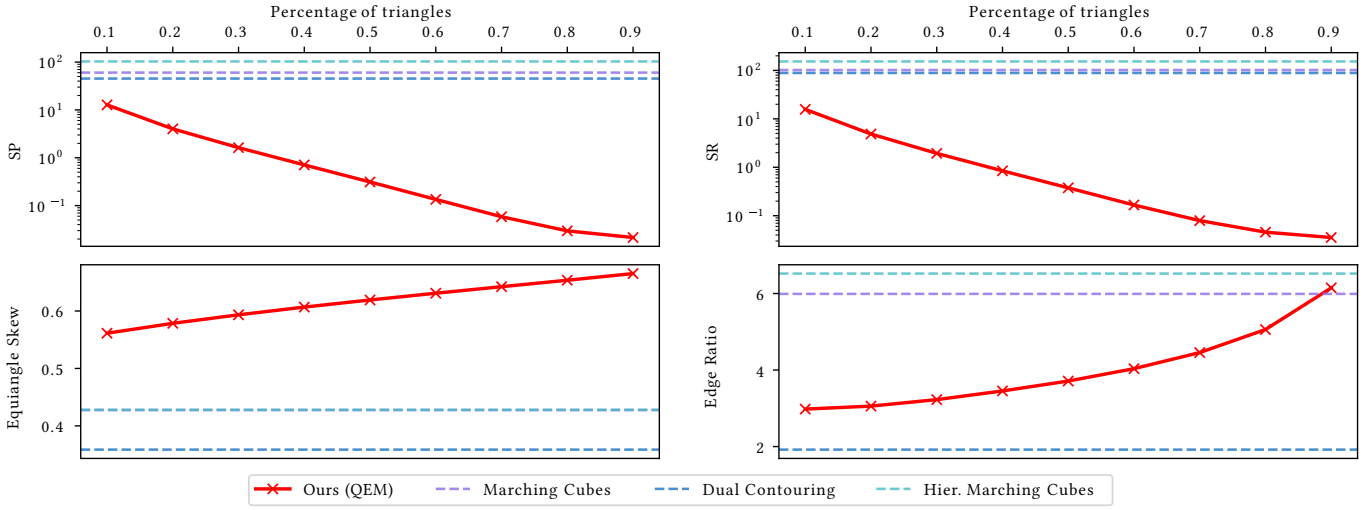

Fig. 10. Soft precision (SP), soft recall (SR), and triangle quality metrics of our meshes after reducing the number of triangles using Quadric Error Metrics (QEM). Additionally, for reference, we include results from several approximation methods without any post-processing. Note that Marching Cubes and Hier. Marching Cubes overlap on some of the triangle quality metrics.

5.4 Triangle Mesh Quality

All analytic extractors output polygonal faces that we tessellate to triangles for downstream use. In the following experiments, we evaluate the quality of the triangle meshes generated by analytic methods with different tessellation approaches compared to approximation methods. We report standard triangle-quality metrics.

Tessellation. Several strategies were considered for tessellating the k-gon faces produced by the analytic extractor. The fan_0 method connects every triangle to the first vertex i_0, yielding the triangle set $\{(i_0, i_j, i_{j+1}) | 1 \leq j \leq k-2\}$. However, it often generates long triangles. *Centroid* introduces the vertex $c = \frac{1}{k}\sum_{j=1}^{k} v_{i_j}$ as an additional vertex. Triangles $\{(c, i_j, i_{j+1}) | 0 \leq j \leq k-2\}$ are emitted cyclically, producing a uniform hub structure, though additional triangles are needed. Lastly, *strip* generates a triangle strip from the polygon, i.e. list of triangles created by iteratively generating triangles that share an edge with the previous triangle in the list.

Metrics. Mesh quality metrics quantify how well a triangulation avoids poorly shaped elements that cause numerical instability. Good elements have angles close to 60°, balanced edge lengths, and near-equilateral proportions. In order to measure such properties, we rely on the following well-established metrics. We analyze the angles of the generated triangles using the maximum, θ_M, and minimum angle, θ_m, [Rupert 1995; Shewchuk 1999] as well as the equiangle skew [Pébay et al. 2007; Stimpson et al. 2007], which measures the deviation from an equilateral triangle:

$$\theta_s = \max\big((\theta_M - 60)/120,\ (60 - \theta_m)/60\big) \in [0, 1]$$

Additionally, we use the ratio between the largest and shortest edge, $L_r = \ell_M/\ell_m$, [Sorgente et al. 2023].

Results. Fig. 9 presents the results of this experiment. While all approximation methods provide a similar triangle quality, analytic methods generate triangle meshes with lower quality according to all metrics. Among the different tessellation approaches, *centroid* provides a slight improvement over fan_0 and *strip* at the cost of additional triangles, with *strip* providing the meshes with the lowest quality. Fig. 11 presents qualitative results of the different tessellation methods in comparison to the original extracted polygons.

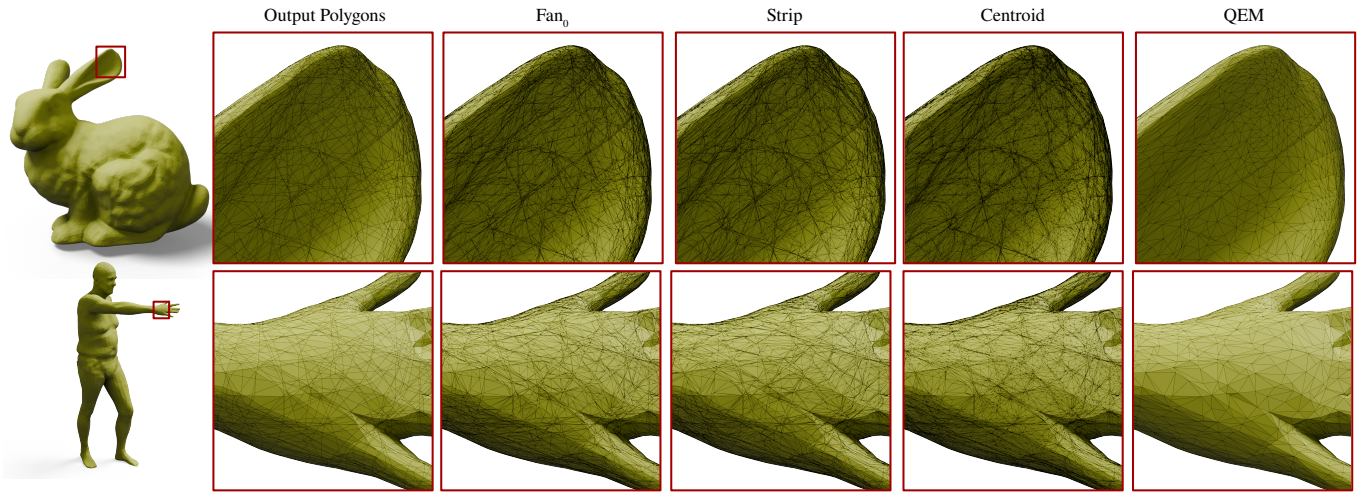

Fig. 11. Different approaches of tessellating the reconstructed polygons compared to the simplified mesh containing 10 % of the original number of triangles.

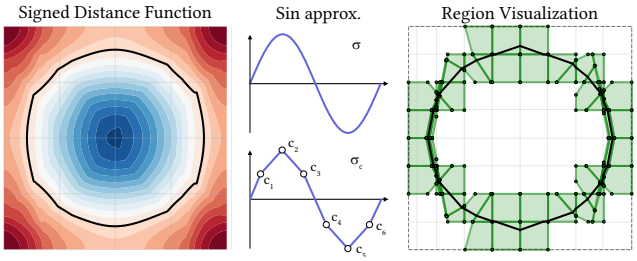

Fig. 12. SDF (left) encoded using a neural architecture with positional encoding. We use piece-wise linear surrogates for the sinusoids (center) in our reconstruction (right). Green polygons represent the linear regions containing the zero-level set.

5.5 Post-processing

The output meshes from analytic methods are usually composed of many small triangles, which arise from the optimization process. However, many of these small triangles provide little information about the underlying shape. Therefore, in this section, we post-process the resulting meshes from our analytic extraction method with a simplification algorithm based on Quadric Error Metrics (QEM) [Garland and Heckbert 1997]. Fig. 10 presents the resulting SP and SR for meshes containing different percentages of the original number of triangles. While reducing the number of triangles increases the error of our meshes, those remain smaller than the error produced by all approximation methods, even when we reduce the number of triangles to 10 % of the original mesh. Moreover, when we analyze the triangle quality metrics of the simplified meshes, we observe that reducing the number of triangles increases the quality of the triangle mesh, reducing the gap between analytic and approximation methods. Lastly, Fig. 11 presents some qualitative results of the simplified mesh compared to different tessellation methods, where we observe that the simplified mesh preserves the original shape while drastically reducing the number of small triangles.

5.6 Range Analysis Filtering

Filtering of linear regions using range analysis is a crucial step of our algorithm. In this section, we evaluate the gains introduced by this step by comparing our marching neurons algorithm to a version of our method where the filtering step is deactivated. We run both algorithms on SDFs encoded by both of our architectures, d4_w128, and d4_w256. Unfortunately, for the d4_w256 architecture, the version without filtering was not able to finish in a window time of two hours for a single mesh due to the exponential grow of the number of linear regions. For the d4_w128 architecture, the filtering step lead to an average speed up of $\times$12.2, highlighting the importance of this step.

5.7 Extension to Other Activations

Our method was designed for neural network architectures with ReLU activation functions, but it can easily generalize to piecewise-linear activations such as leaky ReLU by storing slope and intercept instead of a binary mask. Recent implicit neural representations rely on continuous activation functions such as sine and cosine [Sitzmann et al. 2020; Tancik et al. 2020] to better capture high-frequency details of the surface. In order to use our method with these architectures, such activation functions should be approximated with piecewise-linear surrogates.

To evaluate the viability of such an approach, we train a small ReLU network with two layers and four neurons each to approximate the SDF of a circle in 2D. In this architecture, we process the input coordinates with a positional encoding layer with two frequencies [Mildenhall et al. 2020] During reconstruction, we approximate all periodic functions of the positional encoding layer using a piecewise linear function composed using 6 knots per period, see Fig. 12 right. Fig. 12 also presents the result of such experiment, where our method is able to reconstruct the underlying SDF via the piecewise linear approximations. Unfortunately, our method introduces certain errors in the reconstruction. The magnitude of this error is

defined by the number of linear segments used in the approximation, visible as the square pattern of the linear regions in Fig. 12. However, this error can be reduced by increasing the number of linear segments of the piece-wise linear surrogate.

6 Limitations

Our method is not exempt from limitations. Notably, we rely on range analysis [Duff 1992; Rump and Kashiwagi 2015] to reduce the number of linear cells processed. If the bounds yielded by this analysis are too conservative, or the SDF encodes a complex shape with little empty space, our method needs to retain a lot of cells, leading to increased memory usage and long reconstruction time.

7 Conclusion

We have introduced a novel method for analytic surface extraction from neural implicit shapes that achieves unprecedented accuracy. This was achieved through a natively parallel algorithm design that combines recursive polygon splitting with range analysis to filter empty regions. Our meshes accurately capture the full detail encoded in the neural implicit function by departing from the common practice of treating neural networks as black boxes.

These properties open up promising avenues for future work. Our recursive subdivision scheme is likely to be well-suited for level-of-detail generation, offering greater flexibility for subsequent processing steps that require lower-resolution meshes. Finally, embedding our mesh extractor into an end-to-end differentiable pipeline would enable optimizing a neural SDF with mesh-based supervision, thus unlocking the potential of our approach on a wide range of tasks.

Acknowledgments

This work was partially funded by the Austrian Research Promotion Agency (FFG) under the project “AUTARK – Energy-Aware AMR Safety” (No. 999922723), within the call “Key Technologies for the Future – National 2024”. We gratefully acknowledge Wolfgang Koch for the insightful and valuable research discussions.

References

Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. 2019. *Real-time rendering*. AK Peters/crc Press.

Sergei Azernikov and Anath Fischer. 2005. Anisotropic meshing of implicit surfaces. In *International Conference on Shape Modeling and Applications*. 94–103.

Arturs Berzins. 2023. Polyhedral Complex Extraction from ReLU Networks using Edge Subdivision. In *International Conference on Machine Learning (ICML)*.

Jules Bloomenthal. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5, 4 (1988), 341–355.

Jules Bloomenthal, Brian Wyvill, Geoff Wyvill, Alan H. Barr, and Alyn P. Rockwood. 1997. *Introduction to Implicit Surfaces*.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: Composable transformations of Python+ NumPy programs. (2018).

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012* (2015).

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. 2022. Neural dual contouring. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–13.

Zhiqin Chen and Hao Zhang. 2021. Neural marching cubes. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–15.

Evgeni Chernyaev. 1995. *Marching cubes 33: Construction of topologically correct isosurfaces*. Technical Report.

Julian Chibane, Aymen Mir, and Gerard Pons-Moll. 2020. Neural Unsigned Distance Fields for Implicit Function Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Joao L. D. Comba and J. Stolfi. 1993. Affine arithmetic and its applications to computer graphics. *SIBGRAPI'93* (1993).

Brian Curless and Marc Levoy. 1996. A volumetric method for building complex models from range images. In *Conference on Computer Graphics and Interactive Techniques*. 303–312.

Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–39.

Mark De Berg. 2000. *Computational geometry: algorithms and applications*. Springer Science & Business Media.

Akio Doi and Akio Koide. 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions on Information and Systems* 74, 1 (1991), 214–224.

Tom Duff. 1992. Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry. *ACM SIGGRAPH Computer Graphics* 26, 2 (1992), 131–138.

Michael Garland and Paul S. Heckbert. 1997. *Surface Simplification Using Quadric Error Metrics*.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

J Elisenda Grigsby and Kathryn Lindsey. 2022. On transversality of bent hyperplane arrangements and the topological expressiveness of ReLU neural networks. *SIAM Journal on Applied Algebra and Geometry* 6, 2 (2022), 216–242.

Boris Hanin and David Rolnick. 2019. Deep relu networks have surprisingly few activation patterns. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019).

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: a self-prior for deformable meshes. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 126–1.

Hans-Christian Hege, Martin Seebass, Detlev Stalling, and Malte Zöckler. 1997. A generalized marching cubes algorithm based on non-binary classifications. (1997).

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. 2019. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 41–50.

Adrian Hilton, Andrew J Stoddart, John Illingworth, and Terry Windeatt. 1996. Marching triangles: range image fusion for complex object modelling. In *IEEE International Conference on Image Processing (ICIP)*, Vol. 2. 381–384.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G. Baraniuk. 2023. SplineCam: Exact Visualization and Characterization of Deep Network Geometry and Decision Boundaries. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3789–3798.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Conference on Computer Graphics and Interactive Techniques*. 339–346.
Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.
Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
M. Kohlbrenner and M. Alexa. 2025. Isosurface Extraction for Signed Distance Functions using Power Diagrams. *Computer Graphics Forum* (2025).
Vladimír Kunc and Jiří Kléma. 2024. Three Decades of Activations: A Comprehensive Survey of 400 Activation Functions for Neural Networks. *arXiv preprint arXiv:2402.09092* (2024).
Jiabao Lei and Kui Jia. 2020. Analytic Marching: An Analytic Meshing Solution from Deep Implicit Surface Networks. In *International Conference on Machine Learning (ICML)*.
Jiabao Lei, Kui Jia, and Yi Ma. 2021. Learning and Meshing from Deep Implicit Surface Networks Using an Efficient Implementation of Analytic Marching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2021), 1–1.
William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (1987), 163–169.
Zoë Marschner, Silvia Sellán, Hsueh-Ti Derek Liu, and Alec Jacobson. 2023. Constructive solid geometry on neural signed distance fields. In *SIGGRAPH Asia*. 1–12.
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 *European Conference on Computer Vision (ECCV)*.
Claudio Montani, Riccardo Scateni, and Roberto Scopigno. 1994. Discretized marching cubes. In *Proceedings Visualization*. 281–287.
Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems (NeurIPS)* 27 (2014).
Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. 2020. Topology of deep neural networks. *Journal of Machine Learning Research* 21, 184 (2020), 1–40.
Timothy S. Newman and Hong Yi. 2006. A survey of the marching cubes algorithm. *Computers & Graphics* 30, 5 (2006), 854–879.
Gregory M Nielson. 2004. Dual marching cubes. In *Visualization*. 489–496.
Stanley Osher, Ronald Fedkiw, and K Piechor. 2004. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.* 57, 3 (2004), B15–B15.
Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 165–174.
Razvan Pascanu, Guido Montufar, and Yoshua Bengio. 2014. On the number of inference regions of deep feed forward networks with piece-wise linear activations. In *International Conference on Learning Representations (ICLR)*.
Philippe P Pébay, David Thompson, Jason Shepherd, Patrick Knupp, Curtis Lisle, Vincent A Magnotta, and Nicole M Grosland. 2007. New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *International Meshing Roundtable*. 535–552.
Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. 2017. On the expressive power of deep neural networks. In *International Conference on Machine Learning (ICML)*. 2847–2854.
Daxuan Ren, Hezi Shi, Jianmin Zheng, and Jianfei Cai. 2025. McGrids: Monte Carlo-Driven Adaptive Grids for Iso-Surface Extraction. In *European Conference on Computer Vision (ECCV)*. Springer, 127–144.
Siegfried M Rump and Masahide Kashiwagi. 2015. Implementation and improvements of affine arithmetic. *Nonlinear Theory and Its Applications, IEICE* 6, 3 (2015), 341–359.
Jim Rupert. 1995. A Delaunay refinement algorithm for quality 2D-mesh generation. *Journal of Algorithms* 18, 3 (1995), 548–585.
Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia*.
Silvia Sellán, Yingying Ren, Christopher Batty, and Oded Stein. 2024. Reach For the Arcs: Reconstructing Surfaces from SDFs via Tangent Points. In *SIGGRAPH*. Article 25.
Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. 2018. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning (ICML)*. 4558–4566.
Dario Seyb, Alec Jacobson, Derek Nowrouzezahrai, and Wojciech Jarosz. 2019. Nonlinear sphere tracing for rendering deformed signed distance fields. *ACM Transactions on Graphics (TOG)* 38, 6 (2019).
Nicholas Sharp and Alec Jacobson. 2022. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.
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 Transactions on Graphics (TOG)* 42, 4 (2023).
Jonathan Richard Shewchuk. 1999. Lecture notes on Delaunay mesh generation. (1999).
Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems (NeurIPS)*.
Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019).
Tommaso Sorgente, Silvia Biasotti, Gianmarco Manzini, and Michela Spagnuolo. 2023. A survey of indicators for mesh quality assessment. In *Computer Graphics Forum*, Vol. 42. 461–483.
Barton T Stander and John C Hart. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Conference on Computer Graphics and Interactive Techniques*. 279–286.
CJ Stimpson, CD Ernst, David C Thompson, Patrick Michael Knupp, and Philippe Pierre Pébay. 2007. *The verdict geometric quality library*. Number 1751. Sandia National Laboratories.
Ivan E Sutherland and Gary W Hodgman. 1974. Reentrant polygon clipping. *Commun. ACM* 17, 1 (1974), 32–42.
Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), 7537–7547.
Jonatan Vallin, Karl Larsson, and Mats G Larson. 2023. The geometric structure of fully-connected relu-layers. *arXiv preprint arXiv:2310.03482* (2023).
Kees Van Overveld and Brian Wyvill. 2004. Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface. *The Visual Computer* 20 (2004), 362–379.
Joseph A. Vincent and Mac Schwager. 2021. Reachable Polyhedral Marching (RPM): A Safety Verification Algorithm for Robotic Systems with Deep Neural Network Components. In *International Conference on Robotics and Automation (ICRA)*. 9029–9035.
Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. *Advances in Neural Information Processing Systems (NeurIPS)* (2021).
Yuan Wang. 2022. Estimation and Comparison of Linear Regions for ReLU Networks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 3544–3550.
Alan H Watt. 1999. *3D Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc.
Jane Wilhelms and Allen Van Gelder. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)* 11, 3 (1992), 201–227.
Geoff Wyvill, Craig McPheeters, and Brian Wyvill. 1986. Data structure for soft objects. *The Visual Computer* 2 (1986), 227–234.
Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, Vol. 41. 641–676.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).
Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847* (2018).