Surface Extraction from Neural Unsigned Distance Fields

Congyi Zhang1,2*  Guying Lin1*  Lei Yang1,2  Xin Li3  Taku Komura1
Scott Schaefer3  John Keyser3  Wenping Wang3†
1The University of Hong Kong  2TransGP  3Texas A&M University

Abstract

We propose a method, named DualMesh-UDF, to extract a surface from unsigned distance functions (UDFs), encoded by neural networks, or neural UDFs. Neural UDFs are becoming increasingly popular for surface representation because of their versatility in presenting surfaces with arbitrary topologies, as opposed to the signed distance function that is limited to representing a closed surface. However, the applications of neural UDFs are hindered by the notorious difficulty in extracting the target surfaces they represent. Recent methods for surface extraction from a neural UDF suffer from significant geometric errors or topological artifacts due to two main difficulties: (1) A UDF does not exhibit sign changes; and (2) A neural UDF typically has substantial approximation errors.

DualMesh-UDF addresses these two difficulties. Specifically, given a neural UDF encoding a target surface \( S \) to be recovered, we first estimate the tangent planes of \( S \) at a set of sample points close to \( S \). Next, we organize these sample points into local clusters, and for each local cluster, solve a linear least squares problem to determine a final surface point. These surface points are then connected to create the output mesh surface, which approximates the target surface. The robust estimation of the tangent planes of the target surface and the subsequent minimization problem constitute our core strategy, which contributes to the favorable performance of DualMesh-UDF over other competing methods. To efficiently implement this strategy, we employ an adaptive Octree. Within this framework, we estimate the location of a surface point in each of the octree cells identified as containing part of the target surface. Extensive experiments show that our method outperforms existing methods in terms of surface reconstruction quality while maintaining comparable computational efficiency.

1. Introduction

Implicit surfaces are widely used for surface representation in computer vision and computer graphics. An implicit surface is usually defined as a level set of a function, such as the zero-level set of a signed distance function (SDF). Extracting a mesh representation of an implicit surface from its defining equation is therefore a critical task for surface visualization and processing. Recent advances in machine learning have given rise to a new kind of implicit surface, called a neural implicit surface. A neural implicit surface is a level-set of a function encoded by an MLP (multilayer perceptron) and has the advantage of compactness and inherent smoothness thanks to its MLP representation. SDFs or occupancy fields are widely used in these implicit representations [17, 15, 16, 4, 18, 20, 6, 1, 8, 14, 19].

However, neural implicit surfaces based on the SDF or occupancy fields require inside/outside labeling and thus can only represent orientable and closed surfaces. Hence, as an extension, unsigned distance functions (UDFs) have been used to represent surfaces of arbitrary topologies, including open surfaces with boundaries or non-orientable surfaces (e.g., the Möbius strip). Despite its versatility, applications of a UDF-based surface representation are severely hindered by the difficulty in extracting the target surface it represents, as shown in [5] and [7].

Problem formulation: Suppose that a surface \( \hat{S} \), called the target surface, is defined as the zero-level set of its unsigned distance function (UDF) \( \hat{F}(p) \). Then suppose that this UDF \( \hat{F}(p) \) is approximated by a neural network with the resulting network-encoded UDF being referred to as the neural UDF, denoted by \( F(p) \). Given a neural UDF \( F(p) \), the surface extraction problem is to robustly extract a surface \( S \) from \( F(p) \) such that \( \hat{S} \) well approximates the target surface \( S \).

Challenges: The difficulty in surface extraction from a neural UDF arises from two aspects: (1) A UDF does not have zero-crossings (or sign changes) across the surface it represents. As a result, traditional mesh extraction methods
that rely on zero-crossings (e.g., Marching Cubes [13, 11], Dual Contouring [10], and their variants) are not applicable to UDFs. (2) The MLP representation of a neural UDF tends to have significant approximation errors around the target surface (see detailed error characteristics of neural UDFs in Sec. 3). This makes it even more challenging to extract a high-quality approximation of the target surface.

Several methods, MeshUDF [7], CAP-UDF [21] and Neural Dual Contouring (NDC) [3], have recently been proposed for extracting a mesh surface from a UDF. MeshUDF and CAP-UDF attempt to infer the gradients of a UDF on the grids and determine the sign changes of the estimated gradients, invoking the Marching Cubes method for mesh extraction. When applied to a neural UDF, the sign-change inference step of this method suffers from instability due to the non-negligible error introduced by the approximate MLP representation near the surface where the gradients of the ideal UDF are undefined. As a result, the extracted meshes are less accurate and often have topological errors (e.g., holes). The NDC method proposes a data-driven Dual Contouring approach to predict the position of mesh vertices and dual faces directly from the UDF data. When applied to a neural UDF, this method often produces meshes with considerable artifacts such as holes, zig-zags, etc.

We develop a new strategy, consisting of novel sampling and efficient optimization techniques to address the difficulties in surface extraction from the neural UDF. Suppose the input is a neural UDF $F(p)$ encoding the target surface $S$ to be recovered. Our strategy has two key steps: (1) comput- ing approximate tangent planes of the target surface; and (2) local minimization for generating final surface points.

In Step (1), we first generate sample points $p_i$, around but not too close to, the target surface $S$, because the UDF values and gradients at locations too close to $S$ are relatively unreliable. Thus, $p_i$ are called off-surface sample points. For each $p_i$, we use the UDF value $F(p_i)$ and its gradient $\nabla F(p_i)$ to project $p_i$ towards the target surface to obtain point $q_i = p_i - F(p_i)n_i(p_i)$ where $n_i(p_i) = \nabla F(p_i)/\|\nabla F(p_i)\|$ [5]. These points $q_i$ are called projection points. Although the points $q_i$ are very close to the target surface $S$, as we will show later, the noisy error in the neural UDF makes these points a poor approximation to the target surface.

To further improve surface accuracy, for each projection point $q_i$, we generate an estimated tangent plane $T_i$ of $S$ such that $T_i$ passes through $q_i$ and has the unit normal vector $n_i$. Note that the normal vector $n_i$ of $T_i$ is set to be $n_i(p_i)$ rather than $n_i(q_i)$ since the former is a more reliable estimation. This is because the initial sample point $p_i$ is not too close to $S$, so the gradient $\nabla F(p_i)$ is less contaminated by the pronounced errors of the neural UDF close to $S$.

In Step (2), the estimated tangent planes are organized into clusters, which may overlap. For each cluster of tangent planes $T_i$, we solve a linear least squares problem to produce a final surface point $s_i$ that minimizes the sum of its squared distances to the tangent planes $T_i$. This minimization step based on tangent planes not only provides an accurate surface point but also allows us to faithfully reconstruct the sharp edges of the target surface. Finally, all the surface points $s_i$ from all the clusters are connected to form the output mesh surface to approximate the target surface $S$.

To efficiently implement the above strategy, our DualMesh-UDF method adopts an adaptive Octree structure to partition the space containing the target surface to regular cells. We developed efficient procedures to determine those cells that contain part of the target surface and perform the sampling and minimization procedures in each occupied cell. To connect the surface points to create the output mesh, we follow the Dual Contouring approach, connecting surface points residing in adjacent grid cells to create polygons dual to octree edges.

Extensive experiments demonstrate that our DualMesh-UDF significantly outperforms existing methods in terms of surface reconstruction accuracy and sharp feature preservation.

The main contribution of this work is a new algorithm to robustly and accurately extract a surface from a neural UDF. To overcome the inevitable approximation errors near the target surface and cut locus, we obtain robust estimation of surface tangent planes by leveraging off-surface sample points, use least square minimization to better predict the surface points, and achieve high-quality surface extraction results with sharp features better preserved compared to the state of the art. The code is available at https://github.com/cong-yi/DualMesh-UDF.

2. Related Work

The Marching Cubes method [13] and its variants have been established as the de facto standard of converting distance fields to boundary mesh representations. Using the gradient information of the distance function, Extended Marching Cubes [11] and Dual Contouring [10] are both capable of producing meshes with faithfully preserved sharp features (e.g., corners and edges).

However, all of these methods require inside/outside labeling on the sampling grid, which is either a regular grid of cubes or an adaptive octree grid of cells, to determine whether any zero-level set surface of the signed distance field crosses a particular grid cell. This requirement for sign changes limits the application of these methods to only SDFs or its variants that have sign changes across the underlying surface. Hence, they cannot be directly used to extract a mesh surface from a UDF, which is now often used to represent arbitrary surfaces such as open surfaces or non-orientable surfaces [5].

Our method is similar to [10] only in the sense that we
follow [10] to solve a quadratic error function (QEF) to estimate a surface point per grid cell. But unlike [10], where sign changes are available and the gradients at the zero-crossings are reliable, unsigned distance fields do not have sign changes and are non-differentiable at the zero level set. Additionally, we will show that the information around the zero-level set of a neural UDF is unreliable. Due to these two reasons, we alternatively make use of the spatial sample points that are off the target surface in the neural UDF to find reliably estimated tangent planes and formulate a QEF for estimating the final surface points.

Three relevant methods, MeshUDF [7], CAP-UDF [21] and Neural Dual Contouring (NDC) [3], have recently been proposed to extract meshes from UDFs. MeshUDF [7] and CAP-UDF [21] use the gradient information of the UDF to assign different signs (+ or −) to grid points on different sides of the underlying surface, thus invoking Marching Cubes to extract the surface according to the sign labels. However, the quality of the estimated sign labels can be significantly affected by the accuracy of the neural UDF. As we will show later, the neural UDF becomes less accurate around the target surface it represents, contaminating the inferred sign labels and explaining the poor performance of these two methods. Their use of Marching Cubes is incapable of preserving sharp corners or edges, while the use of a regular grid incurs high memory overhead when a high grid resolution is needed to resolve shape details. In contrast, our DualMesh-UDF method preserves sharp features and uses an adaptive octree grid to reduce computational expense even with a high grid resolution. Furthermore, estimating the surface point location is done by solving a least square problem (QEF) which is more robust than the gradient-based sign-labeling strategy used in MeshUDF in terms of the topology of the resulting meshes.

The NDC method [3] uses a data-driven approach to train a neural network that predicts vertex position per regular grid cell and the overall dual faces directly from a UDF. However, as a data-driven method, NDC’s performance is largely influenced by how accurately the network is trained. We show in the extensive experiments that our explicit geometry-based method consistently outperforms NDC in terms of reconstruction accuracy, preserving sharp features, and preserving smooth boundaries in the original shapes.

3. Error Characteristics of Neural UDF

We first analyze the characteristics of the errors introduced by the MLP representation used to approximate a UDF, which explains why extracting a mesh surface from a neural UDF is difficult. This analysis also provides a foundation for justifying design choices in our method to overcome these challenging characteristics and achieve robust surface extraction.

We begin with an ideal UDF \( \bar{F}(x) \) that represents a target surface \( \bar{S} \) defined by the zero-level set of \( \bar{F}(x) \), that is, \( \bar{S} = \{x | \bar{F}(x) = 0\} \subset \mathbb{R}^3 \). Note that the UDF \( \bar{F}(x) \) is non-negative, so it does not have sign changes across the surface \( \bar{S} \). Furthermore, \( \bar{F}(x) \) is not differentiable at the surface \( \bar{S} \) or the surface’s cut locus \(^1\) [9].

Now suppose that the ideal UDF \( \bar{F}(x) \) is approximated by an MLP, denoted by \( F(x) \). The approximation errors of \( F(x) \) to \( \bar{F}(x) \) are significant in the narrow region around the surface and a narrow region around the cut locus of \( \bar{S} \), because the neural UDF \( F(x) \) is inherently smooth and thus poorly approximates the ideal \( \bar{F}(x) \), which is non-smooth at the surface \( \bar{S} \) and its cut locus. Due to these errors, in general, we have \( F(x) > 0 \) for any \( x \in \bar{S} \) when the MLP is differentiable.

For ease of visualization, we will use a 2D example (see Fig. 1) to illustrate the characteristic behaviors of the approximation errors of \( F(x) \), whose zero-level set defines an open curve (red in Fig. 1(a)). The characteristics of the approximation errors of \( F(x) \) in 3D space are similar. Detailed visualization and analysis of the error behaviors in 3D space are provided in the supplementary materials. Note

---

\(^1\)The cut locus of a surface is a set of points such that each point of the set has two or more distinct closest points on the surface.
that the cut locus (Fig. 1(a)) of \(\bar{S}\) touches the sharp features of \(\bar{S}\). Therefore the neural UDF \(F(x)\) has even more significant errors in terms of both the distance value and the gradient direction around the sharp features (see the error maps of the distance values and the gradient directions in Fig. 1(e,f), respectively). Consequently, in 3D space, the resulting inaccurate distance values and unreliable gradient vectors of the neural UDF \(F(x)\) make it hard to faithfully reconstruct the target surface, especially to preserve the sharp edges in the extracted surface.

To recap, given a neural UDF \(F(x)\), there are mainly two reasons for the difficulty in extracting a surface \(\bar{S}_M\) to approximate the target surface \(\bar{S}\) defined by the ideal UDF \(\bar{F}(x)\): (1) the given neural UDF \(F(x)\) is usually a poor approximation of the ideal UDF \(\bar{F}(x)\) around the target surface \(\bar{S}\) and its cut locus, where \(\bar{F}(x)\) is non-differentiable; and (2) the neural UDF \(F(x)\) does not, in general, have a zero-crossing around the target surface \(\bar{S}\), thus no well-defined surface is associated with it. These issues, faced by current deep neural networks, make it hard to estimate the location of the surface in a numerically stable manner and thus motivate us to develop the two filtering criteria described in Sec. 4.3.

4. Method

Our method consists of three major designs: (1) a quadratic error function (QEF) that solves for a surface point within a cell for mesh extraction (Sec. 4.1); (2) an adaptive octree data structure that produces high-resolution grid cells while reducing the number of QEF solves required (Sec. 4.2); and (3) a point filtering strategy (Sec. 4.3) for a neural UDF, whose distance values and gradient directions are considerably more noisy compared to the GT as discussed in Sec. 3.

The proposed DualMesh-UDF pipeline has three main steps. Firstly, it employs an adaptive tree to partition the domain of the UDF containing the target surface \(\bar{S}\). Then, it detects non-empty cells (defined in Sec. 4.2) based on a two-step cell-shape intersection detection method, and solves one surface point per non-empty leaf cell in a least squares manner. For a neural network, two distance bounds are introduced to filter out sample points that may introduce noisy information and contaminate the localization of the target surface. Lastly, since there are no sign changes in the UDFs, we construct the mesh faces by checking each edge shared by four incident non-empty cells in the octree.

4.1. QEF to locate surface points

Given a cell that contains part of the target surface, we present a procedure to estimate a surface point in this non-empty cell, as illustrated in Fig. 2 for 2D demonstrations. For a neural UDF, we aim to compute a reconstructed point per non-empty cell. For a set of sample points \(\{p_i\}_{i=1}^m\) in the given cell, we query the neural UDF \(F(x)\) to get the approximate distance value and the normalized gradient vector\(^1\) at \(p_i\), i.e., \(d_i = F(p_i)\) and \(n_i = \nabla F(p_i)/\|\nabla F(p_i)\|\), respectively (see Fig. 2a).

Each of the points contributes to an estimated tangent plane of the target surface \(\bar{S}: n_i \cdot (p_i - x) - d_i = 0\). Considering the inaccurate nature of the neural UDF, we define the estimated surface point to be the point that is the closest to all these tangent planes and compute it by minimizing a quadratic error function (a least squares problem):

\[
\nu = \arg \min_x \sum_{i=1}^m (n_i \cdot (p_i - x) - d_i)^2, \tag{1}
\]

where \(m\) is the number of points \(\{p_i\}\) and \(x\) is a surface point to be solved. In our implementation, we solve the linear least squares problem in Eqn. 1 of the form \(\min \|Ax - b\|^2\). For a non-empty cell, this procedure can yield a reconstructed point \(\nu\), which the target surface \(\bar{S}\) approximately crosses.

However, when \(A\) is nearly singular, the solution to Eqn. 1 can be located near the boundary of the cell or even outside of it, which would compromise the quality of extracted mesh surfaces. Note that while [10] proposed an approach with an additional regularization point to stabilize such cases, their approach relies on the intersection points at the zero-level set of an SDF with the cell edges. However, in our setting, the zero-level set of the UDF is elusive or unavailable, therefore we do not have such intersections. We propose an alternative approach based on singular value analysis of the matrix \(A\). We denote the three singular val-

\(^1\)While \(\|\nabla F\|\) should be 1 for a distance function, note that since \(\nabla F\) is drawn from a neural representation, it may differ slightly, and thus need normalization.
ues of the matrix $A$ as $\sigma_0 \geq \sigma_1 \geq \sigma_2 \geq 0$ and consider three cases:

1) In the non-singular case, where all three singular values are much larger than 0, $v$ corresponds to a *sharp feature point* that can be solved directly from Eqn. 1.

2) If only $\sigma_2 \approx 0$, the solution space of $v$ corresponds to a linear edge within the cell. This line has the same direction with the singular vector corresponding to $\sigma_2$ and passes the solution of Eqn. 1. We then compute the intersections between this line and all 6 faces of the cell, yielding two intersecting points. We set $v$ as the midpoint of these two points.

3) In the case where both $\sigma_{1,2} \approx 0$, the part of the target surface enclosed in this cell is approximately planar. Similar to the edge case, we can formulate a plane function from the corresponding singular vectors. By computing the intersecting points between all 12 edges of the cell and this plane, we obtain multiple intersecting points. We use the centroid of these points as $v$.

This way, in the degenerate cases (2 and 3), the point $v$ is still a good approximate solution to Eqn. 1 and well positioned inside the cell.

In a neural UDF (e.g., Fig. 2a), the UDF values $\{d_i\}$ and the gradient directions $\{n_i\}$ are approximated and often unreliable. Consequently, the tangent plane computed from each sample point $p_i$ may not exactly align with the target surface. Solving Eqn. 1 yields a least squares solution that is numerically robust to the approximation errors in neural UDF and can lead to an accurate estimation of the surface point in this non-empty cell. For an ideal UDF case, our method is also applicable to yield an accurate result as shown in Fig. 2b.

Furthermore, to enhance the robustness of our method, we also design a point filtering strategy to remove unreliable sample points, especially those near the target surface and its cut locus, where larger approximation errors exist in the MLP-encoded UDF. The details of this filtering strategy are explained in Sec. 4.3.

**Differences to DC [10].** Note that the dual contouring (DC) method [10] designs a QEF using intersection points between the surface and cell edges, along with the gradient directions at these intersection points. This approach is not applicable to our objective of extracting surface meshes from a neural UDF. This is because (1) it is difficult to find a reliable intersection point, as an exact zero-level set that crosses an edge may not exist, and (2) the gradient directions in the region with lower UDF values (*i.e.*, closer to the target surface) are highly unstable and thus unreliable for estimating the tangent plane. Hence, our method leverages *off-surface* points that are sufficiently far from the target surface, as indicated by their UDF values (see Sec. 4.3). Our approach thus has less stringent requirements than the DC method (as presented in [10]), yet produces satisfactory results even on noisy neural UDFs as demonstrated by our results.

### 4.2. Octree Design and Subdivision

To efficiently process high-resolution data and speed up the mesh extraction process, we employ an octree data structure. As shown in Fig. 3, we recursively subdivide the cells unless they are categorized as *empty* according to the following criterion or they reach the maximal depth. After the subdivision, all leaf nodes that are non-empty will invoke the QEF procedure to solve for the dual points.

**Checking obviously empty cells.** Firstly, we design a checking condition to help quickly prune obviously empty cells. Given an octree cell $C$, let $c_0$ denote its center and $d_0$ denote the UDF value at $c_0$, that is, $d_0 = F(c_0)$. Then we have the following sufficient condition for cell $C$ to be empty:

$$d_0 > \text{Diag}(C)/2,$$

where $\text{Diag}(C)$ is the diagonal length of cell $C$. Since the UDF distance indicates the distance between a spatial point and the target surface, this criterion determines if the target surface lies outside the sphere centered at $c_0$ with radius $\text{Diag}(C)/2$. Given that this sphere contains the entire octree cell, the cell will not contain any portion of the surface if Eqn. 2 is satisfied. Otherwise, the cell will first be categorized as an unsolved cell. Since neural UDF is a reasonable approximation of the ideal UDF, we tailor Eqn. 2 to $d_0 > \text{Diag}(C)/2 + \epsilon$, where $\epsilon > 0$ is a tolerance for the approximation error and set to $\epsilon = 2 \times 10^{-3}$ throughout all experiments. Empirically, we observed this pruning strategy did not affect the quality of the results.

**Maximum octree depth.** Having a predefined maximal depth of the octree is critical, not only in terms of limiting the amount of computation but also because of the inaccuracy of the neural UDF near the target surface. Specifically, if the octree keeps subdividing the space that contains the target surface, after a certain depth the cells will become too small and the sample points $p$ within these cells will lie in the unreliable region of the neural UDF, which will eventually lead to unstable estimation.
4.3. Point filtering for neural UDFs

Since our method makes use of the gradient directions and the distance values to estimate a surface point in each non-empty grid cell (see Eqn. 1), the quality of the extracted mesh heavily depends on the approximation accuracy of the neural UDF $F(x)$. Consider a neighborhood of a sharp feature of the target surface, as shown in the zoom-in views of Fig. 1(e,f). The distance errors and gradient direction errors are more significant around the target surface and its cut locus.

Hence, motivated by our observation and analysis in Sec. 3, to avoid building the QEF using points from these regions, we use the following criteria to filter out unreliable points to enhance the robustness of the QEF solution to the noisy characteristics of the MLP-encoded UDFs.

**Criterion 1: Removing sample points potentially near the surface.** A candidate point $p_i$ is considered too close to the target surface if $F(p_i) < \delta_1$, where $\delta_1$ is a preset filtering threshold; such points are discarded. This criterion ensures the sample points are from a region where the distance errors are expected to be relatively small.

**Criterion 2: Removing sample points whose projections have large UDF values.** While the previous criterion rejects the sample points that are too close to the target surface and thus avoids the erroneous approximation at those regions, some sample points may still have larger UDF errors even far from the target surface (e.g. near the cut locus). We observe that these sample points will not be projected to regions near the target surface. Hence, if the UDF of the projected point $\{q_i\}$ is large, we consider it to be an incorrect estimate and thus reject the corresponding sample point $\{p_i\}$ as unreliable. To this end, we introduce the second filtering criterion as follows, $F(q_i) > \delta_2$, where $\delta_2$ is another preset filtering bound; again, the corresponding points $p_i$ are discarded.

4.4. Creating Output Mesh Surfaces

We consider the regular grid of cubic cells at the maximum depth of the octree. Then each edge of the grid is shared by four cubic cells surrounding the edge. To build the initial mesh connectivity, for each edge of the grid, we examine each of its four incident cells. Similar to [10], the mesh connection rule is designed as follows: if all four incident cells are non-empty, a quad-face candidate that connects the four dual points in these cells will be constructed. To ensure the correct connectivity, we validate and triangulate the quad faces. We further examine if the normals of the face candidates can consistently reflect their geometric property, being a sharp corner, part of a sharp edge, or part of a plane as classified by the SVD shape analysis (Sec. 4.1). For example, all correct triangulated faces incident to a surface point classified as part of a plane would have all their normals parallel to the singular vector corresponding to the largest singular value from SVD shape analysis, while the normals of triangulated faces incident to a surface point being part of a sharp edge should be orthogonal to its singular vector direction. This way, we reject all inconsistent face candidates.

We also provide a practical approach to make sure the output mesh is manifold when the desired target surface is manifold: With the help of the octree cells and the definition of reconstructed dual points, it is feasible to generate an auxiliary *blocky* model by moving all mesh vertices to the centroids of the corresponding cells. By extracting the outer envelope of this auxiliary model, we obtain a manifold structure. Finally, we tessellate the surface points using the connectivity from that manifold structure to produce a manifold mesh. This approach is crucial for applications that require manifold meshes (e.g. parameterization, remeshing, or shape analysis).

5. Experiments

5.1. Experimental details and metrics

**Experimental setting.** We rescaled all the shapes to a bounding cube with a side length of 2, centered at the origin. To improve efficiency, we share sample points between cells, i.e. $m = 27$ points in each non-empty leaf cell, including 8 corner points, 12 edge midpoints, 6 face midpoints, and 1 centroid point. We set the filtering parameters $\delta_1 = 2 \times 10^{-3}$ and $\delta_2 = 2 \times 10^{-3}$ to filter the sampled points as described in Sec. 4.3. We notice that more sample points may bring marginal performance improvement but incur computational overhead in our ablation study.

Given that the grid resolution of $128^3$ and $256^3$ are commonly used in related works, we compare our method with these prior techniques using a maximal octree resolution of $128^3$ and $256^3$. We also discuss how the resolution will affect our results by varying the maximum depth of the octree to have a resolution from $64^3$ to $512^3$ in our supplementary material.

**MLP architecture.** To overfit single shapes using individual MLPs, we employed the MLP implementation provided by [18]. The activation functions are the Sine activation, except for the last one which is a SoftPlus ($\beta = 100$) activation to ensure the output value is non-negative. All neural UDFs in the experiments were trained with the described MLP implementation. All MLP networks were trained for $3k$ iterations with the ADAM optimizer to minimize the difference between the predicted and the GT unsigned distance fields. For more general tasks, we also test an MLP with latent codes that represent a shape space, following the network and training settings in [5]. We report the timing performance on a Linux desktop with an Intel Core™ i7-10870H CPU and an NVIDIA GeForce RTX 3090 graphics card. We elaborate on the loss function as
Figure 4: Meshes extracted from neural UDFs. We show 6 examples and compare our results to the mesh extraction results in MeshUDF [7], CAP-UDF [21] and NDC [3]. Our method preserves sharp geometric features better, yields results without undesirable holes, and reconstructs the original smooth boundaries of open surfaces.
metrics. To evaluate the performance of our method and the other methods, we adopt the following metrics, i.e., the double-sided Chamfer distance (CD), the F-score based on CD, and the Hausdorff distance (HD). The Chamfer distance reflects the overall quality of the extracted surface mesh as compared to the GT shape. The F-score indicates the percentage of points that are reconstructed correctly under a threshold (set to 0.001 for shape-overfitting neural UDFs, and 0.01 for shared UDF network with latent codes). The Hausdorff distance can reveal if there is anything missing or redundant in the reconstructed geometry (e.g., a hole or a floating piece).

5.2. Comparison with SOTA methods

We first compare our results to those obtained by these three methods on a shape collection consisting of shapes from four public sources: 1) 100 from the Thingi10K Dataset [22] containing 3D printing models; 2) 134 from the MGN Dataset [2] containing garments with open boundaries; 3) 100 from the ABC Dataset [12] containing CAD models; and 4) 20 commonly used shapes in geometric processing research. In addition, we also compare different methods on a Möbius strip, which is a non-orientable surface. We fit each shape with an independent neural UDF and apply mesh extraction to each of these neural UDFs.

Table 1 reports the performance of different methods on this shape collection. Our method outperforms the other three competing methods in terms of all three quantitative metrics. While all of these methods use uniform grids, our octree structure results in increased computing efficiency. MeshUDF and CAP-UDF would likely be accelerated by adopting an octree structure, however, it would be non-trivial to adopt an octree structure for NDC. For a fair comparison, we also test our method without using the octree acceleration approach for the resolution of 128$^3$; by doing so, the T1 time increases to 2.70s, and the T2 time increases to 5.54s.

To qualitatively compare results obtained by different methods, we show the mesh surfaces extracted from neural UDFs for several shapes in Fig. 4. We can see that our results are higher quality than those produced by MeshUDF and CAP-UDF. These two methods cannot preserve sharp geometric features due to the use of the Marching Cubes method. Compared to NDC — a data-driven method — our method also produces consistently better results both quantitatively and qualitatively. Specifically, some mesh surfaces extracted by NDC are less smooth than ours. One example (the Pants) is shown in the fourth row in Fig. 4. The staircase artifact observed may be attributed to NDC having been trained on the ABC dataset [12] containing only mechanical components.

Our method is the only method that recovers open boundaries faithfully, while the other compared methods
show staircase artifacts or even redundant pieces near the open boundaries; see the first two rows of Fig. 4. Also, unexpected holes can be observed on the mesh surface extracted by the competing methods from the neural UDFs. Empirically, our method produces quality results without redundant pieces or unexpected holes, generating clear-cut boundaries of the open surfaces as shown in the top four rows of Fig. 4, and reproducing the sharp features as shown in the bottom two rows of the same figure.

More results and comparisons (also on the GT UDF) are presented in our Supplementary Materials, which further validate the superior performance of our method.

**Shared, pre-trained UDF network with latent codes.**
We consider a single neural network with latent codes trained to represent the entire shape space, where each shape is associated with a unique latent code. Specifically, we test our method and the competing methods on the pre-trained UDF network provided by [5] representing 300 ShapeNet car models. Although this UDF network is less accurate than the other overfitting-based network settings considered earlier, our method still outperforms the other methods by a significant margin as reported in Tab. 2, showing the versatility and robustness of our method.

### 5.3. Ablation study

We conducted an ablation study on our method using sampling numbers (the number \( m \) in Eqn. 1) of 27 (\( 3^3 \)) and 125 (\( 5^3 \)) to justify our design choice to use 27 as our sampling number. In Tab. 3, we demonstrate that using 125 sample points per cell results in a marginal improvement but incurs a significantly higher computing cost.

### 5.4. Limitations

While DualMesh-UDF demonstrates the ability to accurately extract surfaces from neural UDFs, some limitations remain. First, the extracted surface cannot be adaptively subdivided with respect to sharp features or fine geometry details. To faithfully reconstruct the fine geometry details, we need to pre-define a sufficient depth of the octree and solve the QEF problem in each of its non-empty leaf nodes. Second, to share the UDF and gradient values at sampled points between adjacent octree cells, we adopt a regular grid sampling strategy. A more flexible and adaptive sampling strategy may bring further improvement to our method.

### 6. Conclusion

We have presented a method for extracting high-quality surface meshes from unsigned distance fields. In order to attain robust performance on MLP-encoded neural UDFs, we discuss the characteristics of the approximation errors of the neural UDF and develop an adaptive octree-based method to effectively localize the target surface embedded in the given UDF. Extensive experiments show that our method outperforms the SOTA methods and produces high-quality results with sharp geometry features, with clear open boundaries, and free of undesirable holes.

### 7. Acknowledgement

This research is partly supported by the Innovation and Technology Commission of the HKSAR Government through the InnoHK initiative. The authors would like to thank Yanhong Lin and Ruixing Jia for their generous help on the project.

### References


