MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks

Benoît Guillard, Federico Stella, and Pascal Fua

CVLab, EPFL,
benoit.guillard@epfl.ch federico.stella@epfl.ch pascal.fua@epfl.ch

Abstract. Unsigned Distance Fields (UDFs) can be used to represent non-watertight surfaces. However, current approaches to converting them into explicit meshes tend to either be expensive or to degrade the accuracy. Here, we extend the marching cube algorithm to handle UDFs, both fast and accurately. Moreover, our approach to surface extraction is differentiable, which is key to using pretrained UDF networks to fit sparse data.

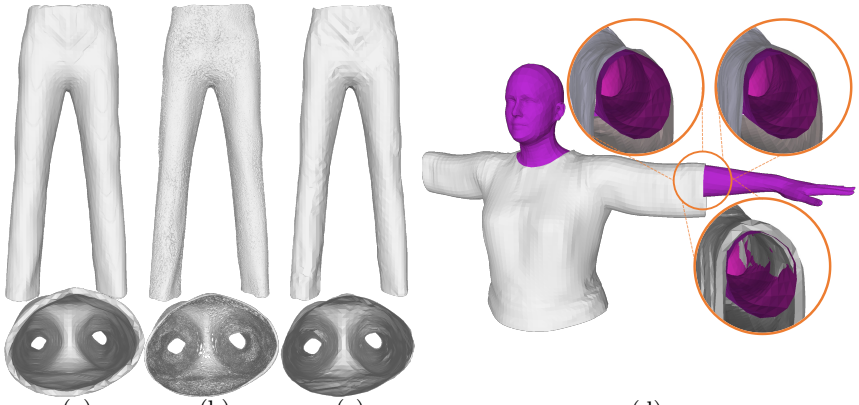


Fig. 1: **Meshing the UDF of a garment.** We present front and top views. **(a)** Inflating shapes to turn open surfaces into watertight ones [33,13,16] inherently reduces accuracy by making the surface thicker, as shown in top view. **(b)** Triangulating a cloud of 3D points collapsed on the 0-levelset [11] is time-consuming and tends to produce rough surfaces. **(c)** Directly meshing the UDF using our approach is more accurate and less likely to produce artifacts. In addition, it makes the iso-surface extraction process differentiable. **(d)** We mesh the UDF of a shirt and display it on a human body. The three insets represent the ground truth shirt, the reconstruction with our method, and the inflation approach, respectively. Our approach—in the upper right inset—produces fewer artifacts and no penetrations with the body.

1 Introduction

In recent years, deep implicit surfaces [29,27,8] have emerged as a powerful tool to represent and manipulate watertight surfaces. Furthermore, for applications that require an explicit 3D mesh, such as sophisticated rendering including complex physical properties [28] or optimizing physical performance [4], they can be used to parameterize explicit 3D meshes whose topology can change while preserving differentiability [1,31,16]. However, these approaches can only handle watertight surfaces. Because common 3D datasets such as ShapeNet [7] contain non-watertight meshes, one needs to preprocess them to create a watertight outer shell [29,34]. This is time consuming and ignores potentially useful inner components, such as seats in a car. An alternative is to rely on network initialization or regularization techniques to directly learn from raw data [2,3] but this significantly slows down the training procedure.

This therefore leaves open the problem of modeling non-watertight surfaces implictly. It has been shown in [11,35,33,13] that occupancy fields and signed distance functions (SDFs) could be replaced by unsigned ones (UDFs) for this purpose. However, unlike for SDFs, there are no fast algorithms to directly mesh UDFs. Hence, these methods rely on a two-step process that first extracts a dense point cloud that can then be triangulated using slow standard techniques [5]. Alternatively, non-watertight surfaces can be represented as watertight thin ones surrounding them [13,16,33]. This amounts to meshing the ϵ iso-surface of an UDF using marching cubes [26], for ϵ being a small strictly positive scalar. Unfortunately, that degrades reconstruction accuracy because the thin surfaces cannot be infinitely so, as ϵ cannot be arbitrarily small. Furthermore, some applications such as draping simulation [21,32,17] require surfaces to be single-faced and cannot be used in conjunction with this approach.

In this paper, we first show that marching cubes can be extended to UDFs by reasoning on their gradients. When neighboring gradients face in opposite directions, this is evidence that a surface element should be inserted between them. We rely on this to replace the sign flips on which the traditional marching cube algorithm depends and introduce a new approach that exploits the gradients instead. This yields vertices and facets. When the UDF is parameterized by latent vectors, we then show that the 3D position of these vertices can be differentiated with respect to the latent vectors. This enables us to fit the output of pre-trained networks to sparse observations, such as 3D points on the surface of a target object or silhouettes of that object.

In short, our contribution is a new approach to meshing UDFs and parameterizing 3D meshes to model non-watertight surfaces whose topology can change while preserving differentiability, which is something that had only been achieved for watertight surfaces before. We use it in conjunction with a learned shape prior to optimize fitting to partial observations via gradient descent. We demonstrate it achieves better reconstruction accuracy than current deep-learning based approaches to handling non-watertight surfaces, in a fraction of the computation time, as illustrated by Fig. 1. Our code is publicly available at https://github.com/cvlab-epfl/MeshUDF/ .

2 Related Work

Deep implicit surfaces [29,27,8] have proved extremely useful to model watertight surfaces using occupancy grids and SDFs, while non-watertight surfaces can be handled either using UDFs or inflating an SDF around them. As meshing almost always involves using a version of the classic marching cube algorithm [26], we discuss these first. We then review recent approaches to representing non-watertight surfaces using implicit surfaces.

Triangulating an Implicit Field. Marching cubes was originally proposed in [26] and refined in [10,25,22] to triangulate the 0-isosurface of a 3D scalar field. It marches sequentially across cubic grid cells and if field values at neighboring corners have opposing signs, triangular facets are created according to a manually defined lookup table. Vertices of these triangle facets are adjusted by linear interpolation over the field values. Since then, newer methods have been developed such as dual methods [19]. They are better at triangulating surfaces with sharp edges at the expense of increased complexity and requiring a richer input. Hence, due to its simplicity and flexibility, along with the availability of efficient implementations, the original algorithm of [22] remains in wide use [27,29,30,18,34]. More recently, [9] proposed a data driven approach at improving sharp features reconstructed by marching cubes. Even though marching cubes is not differentiable, it has been shown that gradients can be estimated for surface points, thus allowing backpropagation [1,31]. However, this approach requires surface normals whose orientation is unambiguous, which makes it impractical when dealing with non-watertight surfaces.

Triangulating Implicit Non-Watertight Surfaces. Unfortunately, neither the original marching cubes algorithm nor any of its recent improvements are designed to handle non-watertight surfaces. One way around this is to surround the target surface with a thin watertight one [13,16,33], as shown in Fig. 1(a). One can think of the process as *inflating* a watertight surface around the original one. Marching cubes can then be used to triangulate the inflated surface, but the result will be some distance away from the target surface, resulting in a loss of accuracy. Another approach is to sample the minimum level set of an UDF field, as in NDF [11] and AnchorUDF [35]. This is done by projecting randomly initialized points on the surface using gradient descent. To ensure full coverage, points are re-sampled and perturbed during the process. This produces a cloud, but not a triangulated mesh with information about the connectivity of neighboring points. Then the ball-pivoting method [5], which connects neighboring points one triplet at a time, is used to mesh the cloud, as shown in Fig. 1(b). It is slow and inherently sequential.

3 Method

We now present our core contribution, a fast and differentiable approach to extracting triangulated isosurfaces from unsigned distance fields produced by a

neural network. Let us consider a network that implements a function

$$\begin{aligned} \phi : \mathbb{R}^C \times \mathbb{R}^3 &\to \mathbb{R}^+ , \\ \mathbf{z}, \mathbf{x} &\mapsto s , \end{aligned} \tag{1}$$

where $\mathbf{z} \in \mathbb{R}^C$ is a parameter vector; $\mathbf{x}$ is a 3D point; s is the Euclidean distance to a surface. Depending on the application, $\mathbf{z}$ can either represent only a latent code that parameterizes the surface or be the concatenation of such a code and the network parameters. In Sec. 3.1, we propose an approach to creating a triangulated mesh $M = (V, F)$ with vertices V and facets F from the 0-levelset of the scalar field $\phi(\mathbf{z}, \cdot)$. Note that it could also apply to non-learned UDFs, as shown in the supplementary material. In Sec. 3.2, we show how to make the vertex coordinates differentiable with respect to $\mathbf{z}$. This allows refinement of shape codes or network parameters with losses directly defined on the mesh.

3.1 From UDF to Triangulated Mesh

Surface Detection within Cells As in standard marching cubes [26], we first sample a discrete regular grid G in the region of interest, typically $[-1, 1]^3$. At each location $\mathbf{x}_i \in G$ we compute

$$u_i = \phi(\mathbf{z}, \mathbf{x}_i) , \qquad \mathbf{g}_i = \nabla_{\mathbf{x}} \phi(\mathbf{z}, \mathbf{x}_i) ,$$

where u_i is the unsigned distance to the implicit surface at location $\mathbf{x}_i$, and $\mathbf{g}_i \in \mathbb{R}^3$ is the gradient computed using backpropagation. Given a cubic cell and its 8 corners, let $(u_1, ..., u_8)$, $(\mathbf{x}_1, ..., \mathbf{x}_8)$, and $(\mathbf{g}_1, ..., \mathbf{g}_8)$ be the above values in each one. Since all u_i are positive, a surface traversing a cell does not produce a sign flip as it does when using an SDF. However, when corners $\mathbf{x}_i$ and $\mathbf{x}_j$ lie on opposite sides of the 0-levelset surface, their corresponding vectors $\mathbf{g}_i$ and $\mathbf{g}_j$ should have opposite orientations, provided the surface is sufficiently smooth within the cell. Hence, we define a *pseudo-signed distance*

$$s_i = \text{sgn}(\mathbf{g}_1 \cdot \mathbf{g}_i) u_i , \tag{2}$$

where $\mathbf{x}_1$ is one of the cell corners that we refer to as the *anchor*. $\mathbf{x}_1$ is assigned a positive pseudo-signed distance and corners where the gradient direction is opposite to that at $\mathbf{x}_1$ a negative one. When there is at least one negative s_i, we use marching cubes' disjunction cases and vertex interpolation to reconstruct a triangulated surface in the cell. Computing pseudo-signs in this way is simple but has two shortcomings. First, it treats each cell independently, which may cause inconsistencies in the reconstructed facets orientations. Second, especially when using learned UDF fields that can be noisy [33], the above smoothness assumption may not hold within the cells. This typically results in holes in the reconstructed meshes.

To mitigate the first problem, our algorithm starts by exploring the 3D grid until it finds a cell with at least one negative pseudo-sign. It then uses it as the starting point for a breadth-first exploration of the surface. Values computed at

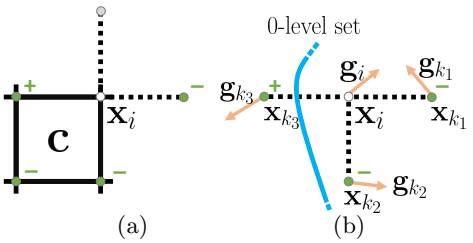


Fig. 2: **Voting. (a)** Corner $\mathbf{x}_i$ of cell $\mathbf{c}$ has 3 neighbors that already have a pseudo-sign and vote. **(b)** The projections of $\mathbf{g}_i$ and $\mathbf{g}_{k_1}$ on the edge connecting the two neighbors face each other. Thus $\mathbf{x}_{k_1}$ votes for $\mathbf{x}_i$ having the same sign as itself (-). The other two neighbors vote for $-$ as well given the result of computing Eq. 3.

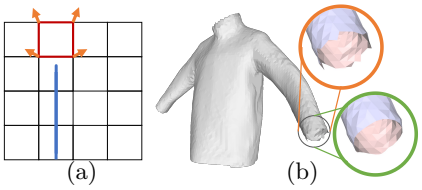


Fig. 3: **Removing artifacts. (a)** Given the blue 0-level surface, the red cell has gradients in opposing directions and yields an undesirable face. We prune these by evaluating the UDF on reconstructed faces. **(b)** Initially reconstructed borders are uneven (top). We smooth them during post-processing (bottom).

any cell corner are stored and never recomputed, which ensures that the normal directions and interpolated vertices are consistent in adjacent cells. The process is repeated to find other non-connected surfaces, if any. To mitigate the second problem we developed a more sophisticated method to assign a sign to each cell corner. We do so as described above for the root cell of our breadth-first search, but we use the voting scheme depicted by Fig. 2 for the subsequent ones. Voting is used to aggregate information from neighboring nodes to estimate pseudo-sign more robustly. Each corner $\mathbf{x}_i$ of a cell under consideration receives votes from all adjacent grid points $\mathbf{x}_k$ that have already been assigned a pseudo-sign, based on the relative directions of their gradients and the pseudo-sign of $\mathbf{x}_k$. Since gradients locally point towards the greatest ascent direction, if the projections of $\mathbf{g}_i$ and $\mathbf{g}_k$ along the edge connecting $\mathbf{x}_i$ and $\mathbf{x}_k$ face each other, there is no surface between them and the vote is in favor of them having the same sign: $v_{ik} = \text{sgn}(s_k)$. Otherwise, the vote depends on gradient directions and we take it to be

$$v_{ik} = (\mathbf{g}_i \cdot \mathbf{g}_k)\text{sgn}(s_k) \tag{3}$$

because the more the gradients are aligned, the more confident we are about the two points being on the same side of the surface or not, depending on the sign of the dot product. The sign of the sum of the votes is assigned to the corner.

If one of the $\mathbf{x}_k$ is zero-valued its vote does not contribute to the scheme, but it means that there is a clear surface crossing. This can happen when meshing learned UDF fields at higher resolutions, because their 0-level set can have a non-negligible volume. Thus, the first non-zero grid point along its direction takes its place in the voting scheme, provided that it has already been explored. To further increase the reliability of these estimates, grid points with many disagreeing votes are put into a lower priority queue to be considered later, when more nearby grid points have been evaluated and can help produce a more consistent sign estimate.

In practice we only perform these computations within cells whose average UDF values of $(u_1, ..., u_8)$ are small. Others can be ignored, thus saving computation time and filtering bad cell candidates which have opposing gradients but lie far from the surface.

Global Surface Triangulation The facets that the above approach yields are experimentally consistent almost everywhere, except for a small number of them, which we describe below and can easily remove in a post-processing stage. Note that the gradients we derive in Sec. 3.2 do not require backpropagation through the iso-surface extraction. Hence, this post-processing step does not compromise differentiability.

Removing Spurious Facets and Smoothing Borders. As shown in Fig. 3 **(a)**, facets that do not correspond to any part of the surfaces can be created in cells with gradients pointing in opposite directions without intersecting the 0-levelset. This typically happens near surface borders because our approach tends to slightly extend them, or around areas with poorly approximated gradients far from the surface in the case of learned UDF fields. Such facets can be detected by re-evaluating the distance field on all vertices. If the distance field for one vertex of a face is greater than half the side-length of a cubic cell, it is then eliminated. Moreover, since marching cubes was designed to reconstruct watertight surfaces, it cannot handle surface borders. As a result, they appear slightly jagged on initial reconstructions. To mitigate this, we apply Laplacian smoothing on the edges belonging to a single triangle. This smoothes borders and qualitatively improves reconstructions, as shown in Fig. 3 **(b)**.

3.2 Differentiating through Iso-Surface Extraction

Let $\mathbf{v} \in \mathbb{R}^3$ be a point on a facet reconstructed using the method of Sec. 3.1. Even though differentiating $\mathbf{v}$ directly through marching cubes is not possible [23,31], it was shown that if ϕ were an SDF instead of an UDF, derivatives could be obtained by reasoning about surface inflation and deflation [1,31]. Unfortunately, for an UDF, there is no "in" or "out" and its derivative is undefined on the surface itself. Hence, this method does not directly apply. Here we extend it so that it does, first for points strictly within the surface, and then for points along its boundary.

Derivatives within the Surface. Let us assume that $\mathbf{v} \in \mathbb{R}^3$ lies within a facet where the surface normal $\mathbf{n}$ is unambiguously defined up to its orientation. Let us pick a small scalar value $\alpha > 0$ and consider

$$\mathbf{v}_+ = \mathbf{v} + \alpha\mathbf{n} \quad \text{and} \quad \mathbf{v}_- = \mathbf{v} - \alpha\mathbf{n} \;,$$

the two closest points to $\mathbf{v}$ on the α-levelset on both sides of the 0-levelset. For α small enough, the outward oriented normals at these two points are close to being $\mathbf{n}$ and $-\mathbf{n}$. We can therefore use the formulation of [1,31] to write

$$\frac{\partial \mathbf{v}_+}{\partial \mathbf{z}} \approx -\mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_+) \qquad \text{and} \qquad \frac{\partial \mathbf{v}_-}{\partial \mathbf{z}} \approx \mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_-) \;. \tag{4}$$

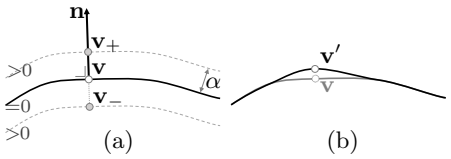


Fig. 4: **Iso-surface deformation**: **(a)** $\mathbf{v}$ on the 0-levelset, $\mathbf{v}_+$ and $\mathbf{v}_-$ at distance α ; **(b)** $\mathbf{v}$ moves to $\mathbf{v}'$ if the UDF decreases at $\mathbf{v}_+$ and increases at $\mathbf{v}_-$.

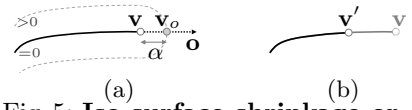


Fig. 5: **Iso-surface shrinkage or extension**: **(a)** with $\mathbf{v}$ on the border of the 0-levelset, we place $\mathbf{v}_o$ at a distance α in the direction of $\mathbf{o}$; **(b)** If the UDF increases at $\mathbf{v}_o$, $\mathbf{v}$ moves to $\mathbf{v}'$.

Since $\mathbf{v} = \frac{1}{2}(\mathbf{v}_- + \mathbf{v}_+)$, Eq. 4, yields

$$\frac{\partial \mathbf{v}}{\partial \mathbf{z}} \approx \frac{\mathbf{n}}{2}\left[\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} - \alpha\mathbf{n}) - \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} + \alpha\mathbf{n})\right] . \tag{5}$$

We provide a more formal proof and discuss the validity of the approximation in appendix. Note that using $\mathbf{n}' = -\mathbf{n}$ instead of $\mathbf{n}$ yields the same result. Intuitively, this amounts to surrounding the 0-levelset with α-margins where UDF values can be increased on one side and decreased on the other, which allows local deformations perpendicular to the surface. Fig. 4 **(a)** depicts the arrangement of $\mathbf{v}$, $\mathbf{v}_+$ and $\mathbf{v}_-$ around the surface. The derivative of Eq. 5 implies that infinitesimally increasing the UDF value at $\mathbf{v}_-$ and decreasing it at $\mathbf{v}_+$ would push $\mathbf{v}$ in the direction of $\mathbf{n}$, as shown in Fig. 4 **(b)**, and conversely. In practice, we use $\alpha = 10^{-2}$ in all our experiments.

Derivatives at the Surface Boundaries. Let us now assume that $\mathbf{v}$ sits on the edge of a boundary facet. Mapping it to $\mathbf{v}_+$ and $\mathbf{v}_-$ and using the derivatives of Eq. 5 would mean that all deformations are perpendicular to that facet. Thus, it does not permit shrinking or expanding of the surface during shape optimization. In this setting, there is a whole family of closest points to $\mathbf{v}$ in the α-levelset; they lay on a semicircle with radius α. To allow for shrinkage and expansion, we map $\mathbf{v}$ to the semicircle point along $\mathbf{o}$, a vector perpendicular to the edge, pointing outwards, and within the plane defined by the facet. Hence, we consider the point $\mathbf{v}_o$ which is the closest to $\mathbf{v}$ on the α-levelset in the direction of $\mathbf{o}$:

$$\mathbf{v}_o = \mathbf{v} + \alpha\mathbf{o} \tag{6}$$

For α small enough, the outward oriented normal at $\mathbf{v}_o$ is $\mathbf{o}$ and we again use the formulation of [1,31] and Eq. 6 to write

$$\frac{\partial \mathbf{v}_o}{\partial \mathbf{z}} = -\mathbf{o}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_o) \quad \text{and} \quad \frac{\partial \mathbf{v}}{\partial \mathbf{z}} = -\mathbf{o}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} + \alpha\mathbf{o}) , \tag{7}$$

which we use for all points $\mathbf{v}$ on border edges. As shown in Fig. 5, this implies that increasing the UDF value at $\mathbf{v}_o$ would push $\mathbf{v}$ inwards and make the surface shrink. Conversely, decreasing it extends the surface in the direction of $\mathbf{o}$.

4 Experiments

We demonstrate our ability to mesh UDFs created by deep neural networks. To this end, we first train a deep network to map latent vectors to UDFs representing different garments, that is, complex open surfaces with many different topologies. We then show that, given this network, our approach can be used to effectively triangulate these garments and to model previously unseen ones. Next, we plug our triangulation scheme into existing UDF networks and show that it is a straightforward operation. Finally, the benefit of the border gradients of Sec. 3.2 is evaluated. The voting scheme proposed in Sec. 3.1 is ablated in appendix, where more qualitative results are also shown.

4.1 Network and Metrics

Our approach is designed to triangulate the output of networks that have been trained to produce UDF fields. To demonstrate this, we use an auto-encoding approach [29] with direct supervision on UDF samples on the MGN dataset [6] to train a network ϕ_θ that maps latent vectors of dimension 128 to UDFs that represent garments. These UDFs can in turn be triangulated using our algorithm to produce meshes such as those of Fig. 1. We provide details of this training procedure in the supplementary material. The MGN dataset comprises 328 meshes. We use 300 to train ϕ_θ and the remaining 28 for testing. For comparison purposes, we also use the publicly available pre-trained network of NDF [11] that regresses UDF from sparse input point clouds. It was trained on raw ShapeNet [7] meshes, without pre-processing to remove inner components make them watertight or consistently orient facets.

To compare the meshes we obtain to the ground-truth ones, we evaluate the following three metrics (details in the supplementary material):

- The **Chamfer distance** (CHD) measures the proximity of 3D points sampled from the surfaces, the lower the better.
- The **Image consistency** (IC) is the product of IoU and cosine similarity of 2D renderings of normal maps from 8 viewpoints, the higher the better.
- The **Normal consistency** (NC) quantifies the agreement of surface normals in 3D space, the higher the better.

4.2 Mesh Quality and Triangulation Speed

Fig. 1 was created by triangulating a UDF produced by ϕ_θ using either our meshing procedure (*Ours*) or one of two baselines:

- *BP*. It applies the ball-pivoting method [5] implemented in [12] on a dense surface sampling of $900k$ points, as originally proposed in [11] and also used in [35]. Surface points are obtained by gradient descent on the UDF field.
- *Inflation* [16,33,13]. It uses standard marching cubes to mesh the ϵ-isolevel of the field, with $\epsilon > 0$.

Table 1: **Comparing UDF meshing methods.** Average Chamfer (CHD), image consistency (IC), normal consistency (NC) and processing time for 300 garments (left) and 300 ShapeNet cars (right). We use a single UDF network in each case and only change the meshing procedure. For BP, we decompose the time into sampling and meshing times.

	Garments, ϕ_θ network			**Cars, NDF network [11]**		
	BP	*Inflation*	*Ours*	*BP*	*Inflation*	*Ours*
CHD (↓)	1.62	3.00	**1.51**	6.84	11.24	**6.63**
IC (%, ↑)	92.51	88.48	**92.80**	90.50	87.09	**90.87**
NC (%, ↑)	89.50	94.16	**95.50**	61.50	**73.19**	70.38
Time (↓)	16.5s + 3000s	**1.0 sec.**	1.2 sec.	24.7s + 8400s	**4.8 sec.**	7.1 sec.

In Tab. 1 (left), we report metrics on the 300 UDF fields $\phi_\theta(\mathbf{z}_i, \cdot)$ for which we have latent codes resulting from the above-mentioned training. *Inflation* and *Ours* both use a grid size of 128^3 over the $[-1, 1]^3$ bounding box, and we set *Inflation*'s ϵ to be 55% of marching cubes' step size. In Tab. 1 (right) we also report metrics for the pretrained NDF network [11] tested on 300 ShapeNet cars, in which case we increase *Inflation* and *Ours* resolution to 192^3 to account for more detailed shapes. An example is shown in Fig. 9 The experiments were run on a NVidia V100 GPU with an Intel Xeon 6240 CPU.

As shown on the left of Table. 1, *Ours* is slightly more accurate than *NDF* in terms of all three metrics, while being orders of magnitude faster. *Inflation* is even faster—this reflects the overhead our modified marching cube algorithm imposes—but far less accurate. To show that this result is not specific to garments, we repeated the same experiment on 300 cars from the ShapeNet dataset and report the results on the right side of Table. 1. The pattern is the same except for NC, which is slightly better for *Inflation*. We conjecture this to be a byproduct of the smoothing provided by *Inflation*, which is clearly visible in Fig. 1**(a,c)**. To demonstrate that these results do not depend on the specific marching cube grid resolution we chose, we repeated the experiment for grid resolutions ranging from 64 to 512 and plot the average CHD as a function of resolution in Fig. 6. It remains stable over the whole range. For comparison purposes, we also repeated the experiment with *Inflation*. Each time we increase the resolution, we take the ϵ value that defines the iso-surface to be triangulated to be 10% greater than half the grid-size, as shown in Fig. 7. At very high resolution, the accuracy of *Inflation* approaches *Ours* but that also comes at a high-computational cost because operating on $512 \times 512 \times 512$ cubes instead of $128 \times 128 \times 128$ ones is much slower, even when using multi-resolution techniques.

4.3 Using Differentiability to Fit Sparse Data

Given the trained network ϕ_θ and latent codes for training shapes from Sec. 4.2, we now turn to recovering latent codes for the remaining 28 test garments. For

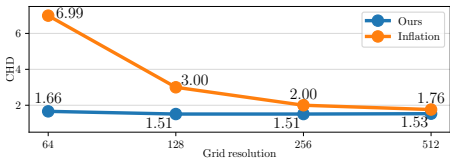


Fig. 6: **CHD as a function of grid resolution.** between reconstructed and ground truth meshes, averaged over the 300 training garments of MGN. *Ours* yields constantly accurate meshes, *Inflation* deforms the shapes at low resolutions.

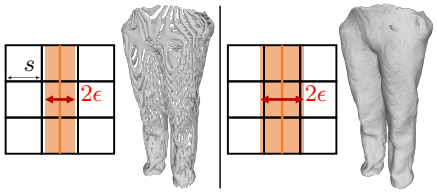
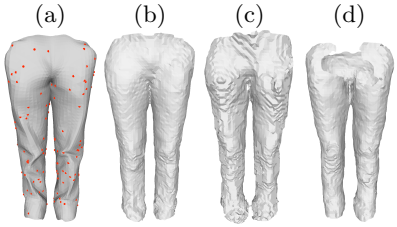


Fig. 7: **Choosing ϵ for *Inflation*.** when meshing a UDF's ϵ iso-level with standard marching cubes, the value of ϵ is lower bounded by half the step size s. **Left**: $2\epsilon < s$ yields many large holes. **Right**: $2\epsilon \geq s$ yields a watertight mesh.

Table 2: **Fitting to sparse point clouds.** The table shows average Chamfer (CHD), image consistency (IC), and normal consistency (NC) wrt. ground truth test garments. We report metrics for un-optimized latent codes (*Init.*), after optimizing ($\mathcal{L}_{PC,mesh}$) using our method, and optimizing either $\mathcal{L}_{PC,UDF}$ or $\widetilde{\mathcal{L}}_{PC,UDF}$ in the implicit domain. **(a)** A sparsely sampled ground truth mesh. **(b)** Mesh reconstructed by mimimizing $\mathcal{L}_{PC,mesh}$, **(c)** $\mathcal{L}_{PC,UDF}$, **(d)** $\widetilde{\mathcal{L}}_{PC,UDF}$.

	Init.	$\mathcal{L}_{PC,mesh}$	$\mathcal{L}_{PC,UDF}$	$\widetilde{\mathcal{L}}_{PC,UDF}$
CHD (↓)	20.45	**3.54**	4.54	4.69
IC (%,↑)	69.54	**84.84**	82.80	82.31
NC (%,↑)	74.54	**86.85**	80.68	86.35



each test garment G_j, given the UDF representing it, this would be a simple matter of minimizing the mean square error between it and the field $\phi_\theta(\mathbf{z}, \cdot)$ with respect to $\mathbf{z}$, which does not require triangulating. We therefore consider the more challenging and more realistic cases where we are only given eiter small set of 3D points P_j—in practice we use 200 points—or silhouettes and have to find a latent vector that generates the UDF that best approximates them.

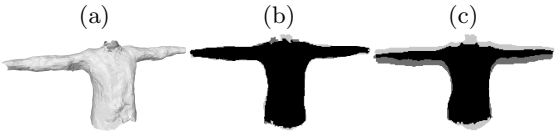
Fitting to 3D points. One way to do this is to remain in the implicit domain and to minimize one of the two loss functions

$$\mathcal{L}_{PC,UDF}(P_j, \mathbf{z}) = \frac{1}{|P_j|} \sum_{p \in P_j} |\phi_\theta(\mathbf{z}, p)| \,, \tag{8}$$

$$\widetilde{\mathcal{L}}_{PC,UDF}(P_j, \mathbf{z}) = \mathcal{L}_{PC,UDF}(P_j, \mathbf{z}) + \frac{1}{|A|} \sum_{a \in A} |\phi_\theta(\mathbf{z}, a) - \min_{p \in P_j} \|a - p\|_2 | \,,$$

Table 3: **Fitting to silhouettes.** Average Chamfer (CHD), image consistency (IC), and normal consistency (NC) wrt. ground truth test garments. We report metrics for un-optimized latent codes (*Init.*), using our method to minimize ($\mathcal{L}_{silh,mesh}$), and by minimizing ($\mathcal{L}_{silh,UDF}$) in the implicit domain. **(a)** Mesh reconstructed by minimizing $\mathcal{L}_{silh,mesh}$. **(b,c)** Superposition of a target silhouette (light gray) and of the reconstructions (dark gray) by minimizing $\mathcal{L}_{silh,UDF}$ or $\mathcal{L}_{silh,mesh}$. Black denotes perfect alignment and shows that the $\mathcal{L}_{silh,UDF}$ mesh is much better aligned.

	Init.	$\mathcal{L}_{silh,mesh}$	$\mathcal{L}_{silh,UDF}$
CHD	20.45	**9.68**	12.74
IC	69.54	**79.90**	74.46
NC	74.54	**81.37**	80.70



where A is a set of randomly sampled points. Minimizing $\mathcal{L}_{PC,UDF}$ means that the given P_j points must be on the zero-level surface of the UDF. Minimizing $\widetilde{\mathcal{L}}_{PC,UDF}$ means that, in addition, the predicted UDF evaluated at points of A must match the approximated UDF computed from P_j. Since the latter is sparse, $\widetilde{\mathcal{L}}_{PC,UDF}$ only provides an approximate supervision.

An alternative is to use our approach to triangulate the UDFs and minimize the loss function

$$\mathcal{L}_{PC,mesh}(P_j, \mathbf{z}) = \frac{1}{|P_j|} \sum_{p \in P_j} \min_{a \in M_\mathbf{z}} \|a - p\|_2 \ , \tag{9}$$

where $a \in M_\mathbf{z}$ means sampling 10k points a on the triangulate surface of $M_\mathbf{z}$. Minimizing $\mathcal{L}_{PC,mesh}$ means that the chamfer distance between the triangulated surfaces and the sample points should be small. Crucially, the results of Sec. 3.2 guarantee that $\mathcal{L}_{PC,mesh}$ is differentiable with respect to $\mathbf{z}$, which makes minimization practical. We tried minimizing the three loss functions defined above. In each case we started the minimization from a randomly chosen latent vector for a garment of the same type as the one we are trying to model, which corresponds to a realistic scenario if the initial estimate is provided by an upstream network. We report our results in Tab. 2. Minimizing $\mathcal{L}_{PC,mesh}$ clearly yields the best results, which highlights the usefulness of being able to triangulate and to differentiate the result.

Fitting to Silhouettes. We now turn to the problem of fitting garments to rasterized binary silhouettes. Each test garment j is rendered into a front-facing binary silhouette $S_j \in \{0,1\}^{256\times256}$. Given S_j only, our goal is to find the latent code $\mathbf{z}_j$ that best encodes j. To this end, we minimize

$$\mathcal{L}_{silh,mesh}(S_j, \mathbf{z}) = L_1(rend(M_\mathbf{z}), S_j) \ , \tag{10}$$

where *rend* is a differentiable renderer [20] that produces a binary image of the UDF triangulation $M_\mathbf{z}$ and $L_1(\cdot)$ is the L_1 distance. Once again, the differentiability of $M_\mathbf{z}$ with respect to $\mathbf{z}$ is key to making this minimization practical.

Table 4: **Ablation Study.** Average Chamfer (CHD), image consistency (IC), and normal consistency (NC) for test garments using either our full approach to computing gradients (*normals + border*) vs. computing the gradients everywhere using only the formula of Eq. 5. (*normals*).

Fitting	Metric	Gradients: *normals*	Gradients: *normals + border*
Point cloud, $\mathcal{L}_{PC,mesh}$	CHD	3.75	**3.54**
	NC	84.28	**84.84**
	IC	86.71	**86.76**
Silhouette, $\mathcal{L}_{silh,mesh}$	CHD	10.45	**9.68**
	IC	78.84	**79.90**
	NC	80.86	**81.37**

In theory, instead of rendering a triangulation, we could have used an UDF differential renderer. Unfortunately, we are not aware of any. Approaches such as that of [24] rely on finding sign changes and only work with SDFs. In contrast, CSP-Net [33] can render UDFs without meshing them but is not differentiable. To provide a baseline, we re-implemented SMPLicit's strategy [13] for fitting a binary silhouette by directly supervising UDF values. We sample a set of points $P \subset [-1, 1]^3$, and project each $p \in P$ to S_j using the front-facing camera $\mathbf{c}$ to get its projected value s_p. If $s_p = 1$, point p falls within the target silhouette, otherwise it falls into the background. SMPLicit's authors advocate optimizing $\mathbf{z}$ by summing

$$\mathcal{L}_{silh,UDF}(S_j, \mathbf{z}) = \begin{cases} |\phi_\theta(\mathbf{z}, p) - d_{max}| & \text{if } s_p = 0 \\ \min_{\bar{p} \text{ s.t. } \mathbf{c}(\bar{p})=\mathbf{c}(p)} |\phi_\theta(\mathbf{z}, \bar{p})| & \text{if } s_p = 1 \end{cases} . \tag{11}$$

on $p \in P$. That is, points projecting outside the silhouette ($s_p = 0$) should have a UDF value equal to the clamping value d_{max}. For points projecting inside the silhouette, along a camera ray we only consider $\bar{p}$, the closest point to the current garment surface estimate and its predicted UDF value should be close to 0. We report our results in Tab. 3. Minimizing $\mathcal{L}_{silh,mesh}$ yields the best results, which highlights the benefits of pairing our method with a differentiable mesh renderer.

Ablation Study. We re-ran the optimizations without the border derivative term of Eq. 7, that is, by computing the derivatives everywhere using the expression of Eq. 5. As can be seen in Tab. 4, this reduces performance and confirms the importance of allowing for shrinkage and expansion of the garments.

4.4 Differentiable Topology Change

A key feature of all implicit surface representations is that they can represent surfaces whose topology can change. As shown in Fig. 8, our approach allows us to take advantage of this while simultaneously creating a mesh whose vertices have associated spatial derivatives. To create this example, we started from a

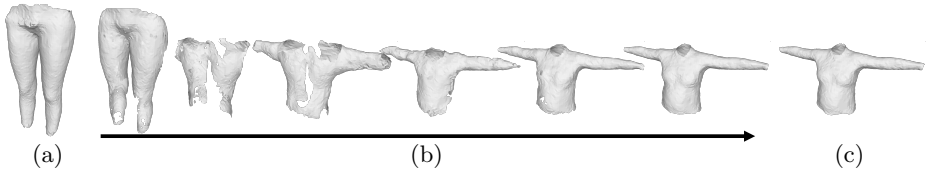


Fig. 8: **Optimization with a change in topology: (a)** Starting mesh associated to the initial latent code $\mathbf{z} = \mathbf{z}_{start}$; **(b)** Optimizing $\mathbf{z}$ with gradient descent by applying a 3D Chamfer loss between the reconstructed mesh and a target shape shown in **(c)**. During optimization, the latent code takes values that do not correspond to valid garments, hence the tears in our triangulations. Nevertheless, it eventually converges to the desired shape.

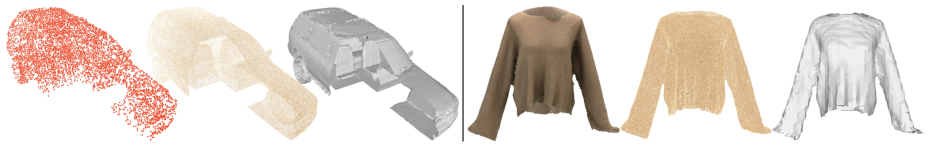


Fig. 9: **Using our approach to triangulate the outputs of NDF [11] (left) and AnchorUDF [35] (right).** In both cases, we display the input to the network, a point cloud in one case and a color image in the other, the dense cloud of points that is the final output of these methods, and a triangulation of the UDF they compute generated using our method.

latent code for a pair of pants and optimized with respect to it to create a new surface that approximates a sweater by minimizing the CHD loss of Eq. 9 over 10k 3D points on that sweater. The topology changes that occur on the mesh representing the deforming shape do not create any difficulties.

4.5 Generalization to other UDF Networks

To show that our meshing procedure is applicable as-is to other UDF-based methods, we use it downstream of publicly available pre-trained networks. In Fig. 9 (bottom) we mesh the outputs of the garment reconstruction network of AnchorUDF [35]. In Fig. 9 (top) we apply it to the point cloud completion pipeline of NDF [11]. Both these methods output dense point clouds surface, which must then be meshed using the time-costly ball pivoting algorithm. Instead, our method can directly mesh the UDF and does so in a fraction of the time while preserving differentiability. That makes the whole algorithm suitable for inclusion into an end-to-end differentiable pipeline.

4.6 Limitations

Reliance on learned UDF fields. The proposed method can mesh the zero-surface of an unsigned distance field. In practice however, UDF fields are approximated

with neural networks, and we find it difficult to learn a sharp 0-valued surface for networks with small capacities. It can for example happen that the approximate UDF field is not reaching zero, or that the zero surface thickens and becomes a volume, or that the gradients are not approximated well enough. In such cases, artifacts such as single-cell holes can appear when using our method at a high resolution. Note that applying our method to a real UDF would not exhibit such issues. By comparison however, applying marching cubes on an approximate and poorly learned SDF is more robust since it only requires the field to be continuous and to have a zero crossing to produce artifact-free surfaces. UDF networks could be made more accurate by using additional loss terms [15] or an adaptive training procedure [14], but this research direction is orthogonal to the method proposed in this paper. Moreover, similarly to [31] for SDFs, since the proposed gradients rely on the field being an UDF, they cannot be used to train a neural network from scratch. This would require network initialization or regularization strategies to ensure it regresses valid UDF fields, a topic we see as an interesting research direction.

Limitations of marching cubes. After locally detecting surface crossings via the pseudo-sign computation, we rely on standard marching cubes for meshing an open surface, which implies the need of a high resolution grid to detect high frequency details, and cubic scalability over grid resolution. Moreover, marching cubes was designed to handle watertight surfaces, and as a consequence some topological cases are missing, for example at surface borders or intersections. This could be remedied by detecting and handling such new cases with additional disjunctions. Finally, the breadth-first exploration of the surface makes the orientation of adjacent facets consistent with each other. However, non-orientable surfaces such as Möbius-strips would intrisically produce juncture points with inconsistent orientations when two different branches of the exploration reach each other. In such points, our method can produce holes. Similarly, marching cubes has geometric guarantees on the topology of reconstructed meshes, but this is not true for the proposed method since there is no concept of *inside* and *outside* in UDFs.

5 Conclusion

We have shown that deep-implicit non-watertight surfaces expressed in terms of unsigned distance functions could be effectively and differentiably triangulated. This provides an explicit parameterization of such surfaces that can be integrated in end-to-end differentiable pipelines, while retaining all the strengths of implicit representations, mainly that a network can accurately represent shapes with different topologies (jeans, sweater...) from the same latent space. In future work, we will explore how it can be used to jointly optimize the pose and clothes of people wearing loose attire.

Acknowledgement. This project was supported in part by the Swiss National Science Foundation.

References

1. Atzmon, M., Haim, N., Yariv, L., Israelov, O., Maron, H., Lipman, Y.: Controlling Neural Level Sets. In: Advances in Neural Information Processing Systems (2019) 2, 3, 6, 7
2. Atzmon, M., Lipman, Y.: SAL: Sign Agnostic Learning of Shapes from Raw Data. In: Conference on Computer Vision and Pattern Recognition (2020) 2
3. Atzmon, M., Lipman, Y.: SALD: Sign Agnostic Learning with Derivatives. In: International Conference on Learning Representations (2020) 2
4. Baqué, P., Remelli, E., Fleuret, F., Fua, P.: Geodesic Convolutional Shape Optimization. In: International Conference on Machine Learning (2018) 2
5. Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., Taubin, G.: The Ball-Pivoting Algorithm for Surface Reconstruction. IEEE Transactions on Visualization and Computer Graphics **5**(4), 349–359 (1999) 2, 3, 8
6. Bhatnagar, B.L., Tiwari, G., Theobalt, C., Pons-Moll, G.: Multi-Garment Net: Learning to Dress 3D People from Images. In: International Conference on Computer Vision (2019) 8
7. Chang, A., Funkhouser, T., G., L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: Shapenet: An Information-Rich 3D Model Repository. In: arXiv Preprint (2015) 2, 8
8. Chen, Z., Zhang, H.: Learning Implicit Fields for Generative Shape Modeling. In: Conference on Computer Vision and Pattern Recognition (2019) 2, 3
9. Chen, Z., Zhang, H.: Neural Marching Cubes. In: ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia) (2021) 3
10. Chernyaev, E.V.: Marching Cubes 33: Construction of Topologically Correct Isosurfaces. In: Institute for High Energy Physics, Moscow, Russia, Report CN/95-17 (1995) 3
11. Chibane, J., Mir, A., Pons-Moll, G.: Neural Unsigned Distance Fields for Implicit Function Learning. In: Advances in Neural Information Processing Systems (2020) 1, 2, 3, 8, 9, 13
12. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: An Open-Source Mesh Processing Tool. In: Eurographics Italian Chapter Conference (2008) 8
13. Corona, E., Pumarola, A., Alenya, G., Pons-Moll, G., Moreno-Noguer, F.: SMPLicit: Topology-Aware Generative Model for Clothed people. In: Conference on Computer Vision and Pattern Recognition (2021) 1, 2, 3, 8, 12
14. Duan, Y., Zhu, H., Wang, H., Yi, L., Nevatia, R., Guibas, L.J.: Curriculum DeepSDF. In: European Conference on Computer Vision (2020) 14
15. Gropp, A., Yariv, L., Haim, N., Atzmon, M., Lipman, Y.: Implicit Geometric Regularization for Learning Shapes. In: International Conference on Machine Learning (2020) 14
16. Guillard, B., Remelli, E., Lukoianov, A., Richter, S., Bagautdinov, T., Baque, P., Fua, P.: Deepmesh: Differentiable Iso-Surface Extraction. In: arXiv Preprint (2021) 1, 2, 3, 8
17. Gundogdu, E., Constantin, V., Parashar, S., Seifoddini, A., Dang, M., Salzmann, M., Fua, P.: Garnet++: Improving Fast and Accurate Static 3D Cloth Draping by Curvature Loss. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(1), 181–195 (2022) 2
18. Hao, Z., Averbuch-Elor, H., Snavely, N., Belongie, S.: Dualsdf: Semantic Shape Manipulation Using a Two-Level Representation. In: Conference on Computer Vision and Pattern Recognition. pp. 7631–7641 (2020) 3

19. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual Contouring of Hermite Data. In: ACM SIGGRAPH (2002) 3
20. Kato, H., Ushiku, Y., Harada, T.: Neural 3D Mesh Renderer. In: Conference on Computer Vision and Pattern Recognition (2018) 11
21. Lahner, Z., Cremers, D., Tung, T.: Deepwrinkles: Accurate and Realistic Clothing Modeling. In: European Conference on Computer Vision (September 2018) 2
22. Lewiner, T., Lopes, H., Vieira, A.W., Tavares, G.: Efficient Implementation of Marching Cubes' Cases with Topological Guarantees. In: Journal of Graphics Tools (2003) 3
23. Liao, Y., Donné, S., Geiger, A.: Deep Marching Cubes: Learning Explicit Surface Representations. In: Conference on Computer Vision and Pattern Recognition. pp. 2916–2925 (2018) 6
24. Liu, S., Zhang, Y., Peng, S., Shi, B., Pollefeys, M., Cui, Z.: Dist: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing. In: Conference on Computer Vision and Pattern Recognition. pp. 2019–2028 (2020) 12
25. Lopes, A., Brodlie, K.: Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. In: IEEE Transactions on Visualization and Computer Graphics (2003) 3
26. Lorensen, W., Cline, H.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: ACM SIGGRAPH. pp. 163–169 (1987) 2, 3, 4
27. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy Networks: Learning 3D Reconstruction in Function Space. In: Conference on Computer Vision and Pattern Recognition. pp. 4460–4470 (2019) 2, 3
28. Nimier-David, M., Vicini, D., Zeltner, T., Jakob, W.: Mitsuba 2: A Retargetable Forward and Inverse Renderer. ACM Transactions on Graphics **38**(6), 1–17 (2019) 2
29. Park, J.J., Florence, P., Straub, J., Newcombe, R.A., Lovegrove, S.: Deepsdf: Learning Continuous Signed Distance Functions for Shape Representation. In: Conference on Computer Vision and Pattern Recognition (2019) 2, 3, 8
30. Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., Geiger, A.: Convolutional Occupancy Networks. In: European Conference on Computer Vision. pp. 523–540 (2020) 3
31. Remelli, E., Lukoianov, A., Richter, S., Guillard, B., Bagautdinov, T., Baque, P., Fua, P.: Meshsdf: Differentiable Iso-Surface Extraction. In: Advances in Neural Information Processing Systems (2020) 2, 3, 6, 7, 14
32. Tang, M., Wang, T., Liu, Z., Tong, R., Manocha, D.: I-Cloth: Incremental Collision Handling for Gpu-Based Interactive Cloth Simulation. In: ACM Transactions on Graphics (2018) 2
33. Venkatesh, R., Karmali, T., Sharma, S., Ghosh, A., Babu, R.V., Jeni, L.A., Singh, M.: Deep Implicit Surface Point Prediction Networks. In: International Conference on Computer Vision (2021) 1, 2, 3, 4, 8, 12
34. Xu, Q., Wang, W., Ceylan, D., Mech, R., Neumann, U.: DISN: Deep Implicit Surface Network for High-Quality Single-View 3D Reconstruction. In: Advances in Neural Information Processing Systems (2019) 2, 3
35. Zhao, F., Wang, W., Liao, S., Shao, L.: Learning Anchored Unsigned Distance Functions with Gradient Direction Alignment for Single-View Garment Reconstruction. In: Conference on Computer Vision and Pattern Recognition (2021) 2, 3, 8, 13