DeepMesh: Differentiable Iso-Surface Extraction

Benoît Guillard, Edoardo Remelli, Artem Lukoianov, Pierre Yvernay, Stephan R. Richter, Timur Bagautdinov, Pierre Baque, and Pascal Fua, Fellow, IEEE

Abstract—Geometric Deep Learning has recently made striking progress with the advent of continuous deep implicit fields. They allow for detailed modeling of watertight surfaces of arbitrary topology while not relying on a 3D euclidean grid, resulting in a learnable parameterization that is unlimited in resolution. Unfortunately, these methods are often unsuitable for applications that require an explicit mesh-based surface representation because converting an implicit field to such a representation relies on the Marching Cubes algorithm, which cannot be differentiated with respect to the underlying implicit field. In this work, we remove this limitation and introduce a differentiable way to produce explicit surface mesh representations from Deep Implicit Fields. Our key insight is that by reasoning on how implicit field perturbations impact local surface geometry, one can ultimately differentiate the 3D location of surface samples with respect to the underlying deep implicit field. We exploit this to define DeepMesh — an end-to-end differentiable mesh representation that can vary its topology. We validate our theoretical insight through several applications: Single view 3D reconstruction, Shape Optimization, Full Scene 3D Reconstruction from Scans and End-to-End Training. In all cases our end-to-end differentiable parameterization gives us an edge over state-of-the-art algorithms.

Index Terms—Computational and artificial intelligence, neural networks electronic design automation and methodology, design methodology, graphics, shape computers and information processing, image processing, machine vision, object recognition mathematics, geometry, computational geometry, geometric modeling mathematics, numerical analysis, surface fitting systems engineering and theory, systems, man, and cybernetics, user interfaces, data visualization, isosurfaces.

I. INTRODUCTION

GEO METRIC Deep Learning has recently witnessed a breakthrough with the advent of Deep Implicit Fields (DIFs) [1], [2], [3]. These enable detailed modeling of watertight surfaces without relying on a 3D euclidean grid or meshes with fixed topology, resulting in a learnable surface parameterization that is not limited in resolution.

However, a number of important applications require explicit surface representations, such as triangulated meshes or 3D point clouds. Computational Fluid Dynamics (CFD) simulations and the associated learning-based surrogate methods used for shape design in many engineering fields [4], [5] are a good example of this where 3D meshes serve as boundary conditions for the Navier-Stokes equations. Similarly, many advanced physically-based rendering engines require surface meshes to model the complex interactions of light and physical surfaces efficiently [6], [7].

Making explicit representations benefit from the power of deep implicit fields requires converting the implicit surface parameterization to an explicit one, which typically relies on one of the many variants of the Marching Cubes algorithm [8], [9]. However, these approaches are not fully differentiable [10]. This makes it difficult to use continuous deep implicit fields to parameterize explicit surface meshes.

The non-differentiability of Marching Cubes has been addressed by learning differentiable approximations of it [10], [13]. These techniques, however, remain limited to low-resolution meshes [10] or fixed topologies [13]. An alternative approach is to reformulate downstream tasks, such as differentiable rendering [14], [15] or surface reconstruction [16], directly in terms of implicit functions, so that explicit surface representations are no longer needed. However, doing so is not easy and may even not be possible for more complex tasks, such as solving CFD optimization problems.

By contrast, we show that it is possible to use implicit functions, be they signed distance functions or occupancy maps, to produce explicit surface representations while preserving differentiability. Our key insight is that 3D surface samples can be differentiated with respect to the underlying deep implicit field, which is in the spirit of implicit differentiation [17]. We prove this formally by reasoning about how implicit field perturbations impact 3D surface geometry locally. Specifically, we derive a closed-form expression for the derivative of a surface sample with respect to the underlying implicit field, which is independent of the method used to compute the iso-surface. This lets us extract the explicit surface using a non-differentiable algorithm, such as Marching Cubes, and then perform the backward pass through the extracted surface samples. This yields an end-to-end differentiable surface parameterization that can describe arbitrary topology and is not limited in resolution. We will refer to our approach as DeepMesh. We first introduced it...
Fig. 1. DeepMesh. (a) We condition our representation on an input image and output an initial 3D mesh, which we refine via differentiable rasterization [11], thereby exploiting DeepMesh’s end-to-end differentiability. (b) We use our parameterization as a powerful regularizer for aerodynamic optimization tasks. Here, we start from an initial car shape and refine it to minimize pressure drag. (c) We use the end-to-end differentiability of iso-surface extraction to improve the occupancy field fitted to a sparse point cloud of a whole scene by an off-the-shelf network, Convolutional Occupancy Network (CON) [12]. In these two examples, the raw output of [12] is shown on the left and the refined version on the right. The errors are shown in red and are smaller after refinement.

in a conference paper [18] that focused on the 0-iso-surface of signed distance functions. We extend it here to iso-surface of generic implicit functions, such as occupancy fields by harnessing simple multivariate calculus tools.

We showcase the power and versatility of DeepMesh in several applications.

1) Given a model trained to map latent vectors to SDFs, we use our approach to triangulate the SDF fields and write image-based losses that yield improved 3D reconstructions from single images, as shown in Fig. 1(a).

2) Similarly, we use the surface triangulations to compute the aerodynamic properties of 3D shapes and refine them, as shown in Fig. 1(b).

3) We use our paradigm in conjunction with DIF-based methods to improve their performance in a plug-and-play fashion by adding loss terms that can be computed on the meshes. This highlights the importance to be able to handle both SDFs and occupancy grids.

4) We demonstrate that we can use our approach not only to better exploit the results of pre-trained networks but to actually train them better.

In all these cases, our end-to-end differentiable parameterization gives us an edge over state-of-the-art algorithms. Note, however, that our approach relies on latent variable models to capture priors applicable to entire object categories, unlike some of the recent multi-view approaches [19], [20] that return extremely detailed models but at the cost of overfitting for a single 3D scene. In a way, we trade off extreme reconstruction accuracy for generality.

In short, our core contribution is a theoretically well-grounded and computationally efficient way to differentiate through isosurface extraction. This enables us to harness the full power of neural implicit fields to define an end-to-end differentiable surface mesh parameterization that allows topology changes.

II. RELATED WORK

A. From Discrete to Continuous Implicit Surfaces.

Level sets of a 3D function can represent watertight surfaces whose topology can change [21], [22]. Being representable as 3D grids and thus easily processable by standard deep learning architectures, they have been used extensively [23], [24], [25], [26], [27], [28], [29], [30]. However, methods operating on dense grids have been limited to low resolution volumes due to excessive memory requirements. Methods operating on sparse representations of the grid tend to trade off the need for memory for a limited representation of fine details and lack of generalization [27], [28], [31], [32].

This has changed recently with the introduction of continuous deep implicit fields, which represent 3D shapes as level sets of dense grids that map 3D coordinates to a signed distance function [1] or occupancy field [2], [3]. This yields a continuous shape representation with respect to 3D coordinates that is lightweight but not limited in resolution. This representation has been successfully used for single view 3D reconstruction [2], [3], [33], and 3D shape completion [34], [35].

Signed distance and occupancy fields have case-specific benefits, and our methods applies to both representations. For the applications we consider in Section IV-B, SDFs appear to represent more accurate surfaces. Occupancy fields are however more suited to union operations in the implicit domain, since the minimum of 2 occupancy fields yields a valid occupancy. This property can be useful for combining shape primitives in Constructive Solid Geometry (CSG) applications [36], but does not always hold for SDFs. Similarly, computing ground truth SDF values of a mesh with internal surface elements yields a false zero-levelset with no change of sign, and this source of inaccuracy is removed when using occupancy.
However, for applications requiring explicit surface parameterizations, the non-differentiability of iso-surface extraction so far largely prevented the exploitation of implicit representations. Exceptions are [37], [38] that propose a solution to differentiate through iso-surface extraction specifically tailored to differentiable rasterization. By contrast, our method for implicit differentiation is agnostic to the downstream task. Our expression is similar to the one of [39], which formulates surface derivative with respect to time instead of latent vectors. However, our derivation clarifies the underlying assumptions, namely that the vertices move towards their closest neighbors when the surface deforms infinitesimally. Following the submission of our conference paper [18] and after the original submission of this extended journal version, it has been proposed [40] to use the gradients that we derive by breaking down the descent step in two separate stages, acting first on the implicit field and then on the network parameters, which improves the results.

B. Converting Implicit Functions to Surface Meshes

The Marching Cube (MC) algorithm [8], [9], [41] is a popular way to convert implicit functions to surface meshes. The algorithm proceeds by sampling the field on a discrete 3D grid, detecting zero-crossing of the field along grid edges, and building a surface mesh using a lookup table. Unfortunately, the process of determining the position of vertices on grid edges involves linear interpolation, which does not allow for topology changes through backpropagation [10], as illustrated in Fig. 2(a). Because this is a central motivation for this work, we provide a more detailed analysis of this shortcoming in the supplementary material. In what follows, we discuss two classes of methods that tackle the non-differentiability issue. The first one emulates iso-surface extraction with deep neural networks, while the second one avoids the need for mesh representations by formulating objectives directly in the implicit domain.

1) Emulating Iso-Surface Extraction: In [10] Deep Marching Cubes maps voxelized point clouds to a probabilistic topology distribution and vertex locations defined over a discrete 3D euclidean grid through a 3D CNN. While this allows changes to surface topology through backpropagation, the probabilistic modeling requires keeping track of all possible topologies at the same time, which, in practice, limits resulting surfaces to low resolutions. Voxel2mesh [13] deforms a mesh primitive and adaptively increases its resolution. While this makes it possible to represent high resolution meshes, it prevents changes of topology.

2) Writing Objective Functions in Terms of Implicit Fields: In [16], variational analysis is used to re-formulate standard surface mesh priors, such as those that enforce smoothness, in terms of implicit fields. Although elegant, this technique requires carrying out complex derivations for each new loss function and can only operate on an euclidean grid of fixed resolution. The differentiable renderers of [14], [42], [43] rely on sphere tracing and operate directly in terms of implicit fields. Unfortunately, since it is computationally intractable to densely sample the underlying volume, these approaches either define implicit fields over a low-resolution euclidean grid [14] or rely on heuristics to accelerate ray-tracing [42], while reducing accuracy. 3D volume sampling efficiency can be improved by introducing a sparse set of anchor points when performing ray-tracing [15]. However, this requires reformulating standard surface mesh regularizers in terms of implicit fields using computationally intensive finite differences. Furthermore, these approaches are tailored to differentiable rendering, and are not directly applicable to different settings that require explicit surface modeling, such as computational fluid dynamics. This also applies to [37], [38] that use implicit differentiation for implicit surface rendering. Both can be seen as special cases of the gradients we derive where surface points only move along the viewing direction.

III. Method

Tasks such as Single view 3D Reconstruction (SVR) [44], [45] or shape design in the context of CFD [4] are commonly performed by deforming the shape of a 3D surface mesh \( M = (V,F) \), where \( V = \{v_1,v_2,\ldots\} \) denotes vertex positions in \( \mathbb{R}^3 \) and \( F \) facets, to minimize a task-specific loss function \( L_{\text{task}}(M) \). \( L_{\text{task}} \) can be, e.g., an image-based loss defined on the output of a differentiable renderer for SVR or a measure of aerodynamic performance for CFD.

To perform surface mesh optimization robustly, a common practice is to rely on low-dimensional parameterizations that are either learned [1], [46], [47] or hand-crafted [4], [5], [48]. In that setting, a differentiable function maps a low-dimensional set of parameters \( x \) to vertex coordinates \( V \), implying a fixed topology. Allowing changes of topology, an implicit surface representation would pose a compelling alternative but conversely require a differentiable conversion to explicit representations in order to backpropagate gradients of \( L_{\text{task}} \).

In the remainder of this section, we first recapitulate neural implicit surface representations that underpin our approach. We
then introduce our main contribution, a differentiable approach to computing surface samples and updating their 3D coordinates to optimize $L_{\mathrm{task}}$. Finally, we present DeepMesh, a fully differentiable surface mesh parameterization that can represent arbitrary topologies.

A. Deep Implicit Field Representation

In this work, we represent a generic watertight surface $S$ implicitly by a function $s : \mathbb{R}^3 \to \mathbb{R}$. Typical choices for $s$ include the Signed Distance Function (SDF) where $s(x)$ is $d(x, S)$ if $x$ is outside $S$ and $-d(x, S)$ if it is inside, where $d$ is the euclidean distance; and Occupancy Maps with $s(x) = 1$ inside and $s(x) = 0$ outside.

Given a dataset of watertight surfaces $D$, such as ShapeNet [49], we train a Multi-Layer Perceptron (MLP) $f_\Theta$ as in [50] to approximate $s$ over such set of surfaces $D$ by minimizing

$$
\mathcal{L}_{\mathrm{imp}}(\{z_S\}_{S \in D}, \Theta) = \mathcal{L}_{\mathrm{data}}(\{z_S\}_{S \in D}, \Theta) + \lambda_{\mathrm{reg}} \sum_{S \in D} \|z_S\|^2,
$$

where $z_S \in \mathbb{R}^Z$ is the $Z$-dimensional encoding of surface $S$, $\Theta$ denotes network parameters, $\mathcal{L}_{\mathrm{data}}$ is a data term that measures how similar $f_\Theta$ is to the ground-truth function $s$ corresponding to each sample surface, and $\lambda_{\mathrm{reg}}$ is a weight term balancing the contribution of reconstruction and regularization in the overall loss.

In practice when $s$ is a signed distance, we take $\mathcal{L}_{\mathrm{data}}$ to be the $L_1$ loss

$$
\mathcal{L}_{\mathrm{data}} = \sum_{S \in D} \frac{1}{|X_S|} \sum_{x \in X_S} |f_\Theta(z_S, x) - s(x)|,
$$

where $X_S$ denotes sample 3D points on the surface $S$ and around it. When $s$ is an occupancy map, we take $\mathcal{L}_{\mathrm{data}}$ to be the binary cross entropy loss

$$
\mathcal{L}_{\mathrm{data}} = -\sum_{S \in D} \frac{1}{|X_S|} \sum_{x \in X_S} s(x) \log(f_\Theta(z_S, x)) + (1 - s(x)) \log(1 - f_\Theta(z_S, x)).
$$

Once trained, $s$ is approximated by $f_\Theta$ which is by construction continuous and differentiable almost everywhere for all standard activation functions (ReLU, sigmoid, tanh...). Consequently, $s$ can be taken to be a level-set of $\{x \in \mathbb{R}^3, f_\Theta(z_S, x) = \alpha\}$, where $\alpha$ is zero for SDFs and typically 0.5 for occupancy grids. Since $f_\Theta$ is defined up to a constant, we will refer to zero-crossings in the rest of the paper for simplicity.

B. Differentiable Iso-Surface Extraction

Once the weights $\Theta$ of (1) have been learned, $f_\Theta$ maps a latent vector $z$ to a signed distance or occupancy field and the surface of interest is its zero level set. Recall that our goal is to minimize the objective function $L_{\mathrm{task}}$ introduced at the beginning of this section. As it takes as input a mesh defined in terms of its vertices and facets, evaluating it and its derivatives requires a differentiable conversion from an implicit field to a set of vertices and facets, something that Marching Cubes does not provide, as depicted by Fig. 2(a). More formally, we need to evaluate

$$
\frac{\partial L_{\mathrm{task}}}{\partial c} = \sum_{x \in V} \frac{\partial L_{\mathrm{task}}}{\partial x} \frac{\partial x}{\partial c},
$$

where the $x$ are mesh vertices and therefore the surface. $c$ stands for either the latent $z$ vector if we wish to optimize $L_{\mathrm{task}}$ with respect to $z$ only or for the concatenation of the latent vector and the network weights $[z|\Theta]$ if we wish to optimize with respect to both the latent vectors and the network weights. Note that we compute $\partial L_{\mathrm{task}}/\partial c$ by summing over the mesh vertices but we could use any other sampling of the surface.

1) Differentiating the Loss Function: In this work, we take inspiration from classical functional analysis [51] and reason about the continuous zero-crossing of the implicit function $s$ rather than focusing on how vertex coordinates depend on the implicit field $f_\Theta$ when sampled by the marching cubes algorithm. To this end, we prove below the following theorem.

Theorem 1. If the gradient of $f_\Theta$ at point $x$ located on the surface does not vanish, then $\frac{\partial s}{\partial c} = -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \frac{\partial f_\Theta(z, x)}{\partial c}$ where $\mathbf{n} = \nabla f_\Theta(x)$ is the normal to the surface at $x$.

Injecting this expression of $\partial s/\partial c$ into (4) yields

$$
\frac{\partial L_{\mathrm{task}}}{\partial c} = -\sum_{x \in V} \frac{\partial L_{\mathrm{task}}}{\partial x} \frac{\nabla f_\Theta}{\|\nabla f_\Theta\|^2} \frac{\partial f_\Theta}{\partial c}.
$$

Note that when $s$ is an SDF, $\|\nabla s\| = 1$ and therefore $\|\nabla f_\Theta\| \approx 1$. The $\|\nabla f_\Theta\|^2$ numerator from (5) can then be ignored, which is consistent with the result we presented in [18]. This scaling factor is not I for other implicit fields. However, it does not affect the direction of the gradients only their magnitude, which often gets rescaled by optimizers [52], [53] anyway.

Proof of Theorem 1: We start by stating the Implicit Function Theorem (IFT), which we later use in our proof.

Theorem 2 (Implicit Function Theorem - IFT): Let $F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ and $c_0 \in \mathbb{R}^m$, $p_0 \in \mathbb{R}^n$ such that:

1) $F(c_0, p_0) = 0$;

2) $F$ is continuously differentiable in a neighborhood of $(c_0, p_0)$;

3) the partial Jacobian $\partial_p F(c_0, p_0) \in \mathbb{R}^{n \times n}$ is non-singular.

Then there exists a unique differentiable function $p^* : \mathbb{R}^n \to \mathbb{R}^n$ such that:

1) $p_0 = p^*(c_0)$;

2) $F(c, p^*(c)) = 0$ for all $c$ in the above mentioned neighborhood of $c_0$;

3) $\partial p^*(c) = -[\partial_p F(c_0, p_0)]^{-1} \partial_c F(c_0, p_0)$, that is, a matrix in $\mathbb{R}^{n \times m}$.

Intuitively, $p^*$ returns the solutions of a system of $n$ equations—the $n$ output values of $F$—with $m$ unknowns. For our purposes, $c \in \mathbb{R}^m$ can be either the shape code and the network weights jointly or the shape code only, as discussed above.

To apply the IFT to our problem, let us rewrite $f_\Theta$ as a function $M : \mathbb{R}^m \times \mathbb{R}^3 \to \mathbb{R}$ that maps $c \in \mathbb{R}^m$ and a point in $p \in \mathbb{R}^3$ to a scalar value $M(c, p) \in \mathbb{R}$. The IFT does not directly apply to $M$ because it operates from $\mathbb{R}^m \times \mathbb{R}^3$ into $\mathbb{R}$ instead of into $\mathbb{R}^3$.

Hence, we must add two more dimensions to the output space of $M$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
To this end, let \( c_0 \in \mathbb{R}^m; p_0 \in \mathbb{R}^3 \) such that \( M(c_0, p_0) = 0 \), meaning that \( p_0 \) is on the implicit surface defined by parameter \( c_0 \); and \( u \in \mathbb{R}^3 \) and \( v \in \mathbb{R}^3 \) such that \((u, v)\) is a basis of the tangent plane to the surface \( \{ M(c_0, \cdot) = 0 \} \) at \( p_0 \). Let \( n = \partial_p M(c_0, p_0) \) be the normal vector to the surface at \( p_0 \). This lets us define the function \( F : \mathbb{R}^m \times \mathbb{R}^3 \rightarrow \mathbb{R}^3 \) as

\[
F(c, p) \mapsto \begin{pmatrix} M(c, p) \\ (p - p_0) \cdot u \\ (p - p_0) \cdot v \end{pmatrix},
\]

By construction, we have \( n \cdot u = n \cdot v = 0 \) and \( F(c_0, p_0) = 0 \).

Note that the first value of the function \( F(c, p) \) vector is zero when the point \( p \) is on the surface defined by \( c \) while the other two are equal to zero when \((p - p_0)\) is perpendicular to the surface defined by \( c_0 \). By zeroing all three, \( p^* \) returns a point \( p \) that is on the surface for \( c \neq c_0 \) and such that \((p - p_0)\) is perpendicular to the surface. A geometric interpretation is that \( p_0 \) is the point on the surface defined by \( c_0 \) that is the closest to \( p^*(c) \). This is illustrated on Fig. 2(b).

Given the IFT applied to \( F \), there is a mapping \( p^* : \mathbb{R}^m \rightarrow \mathbb{R}^3 \) such that

1) \( p_0 = p^*(c_0) \);
2) \( F(c, p^*(c)) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \) for all \( c \) in a neighborhood of \( c_0 \).
3) \( \partial p^*(c_0) = -[\partial_p F(c_0, p_0)]^{-1} \partial_c F(c_0, p_0) \).

We have

\[
\partial_c F(c_0, p_0) = \begin{pmatrix} \partial_c M(c_0, p_0) \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{3 \times m},
\]

\[
\partial_p F(c_0, p_0) = \begin{pmatrix} n \\ u \\ v \end{pmatrix} \in \mathbb{R}^{3 \times 3}
\]

Given that the last two rows of \( \partial_p F(c, p_0) \) are zero, to compute \( \partial p^*(c_0) \) according to the IFT, we only need to evaluate the first column of \([\partial_p F(c_0, p_0)]^{-1}\). As the two last rows of \( \partial_p F(c_0, p_0) \) are \( u \) and \( v \) that are unit vectors such that \( u \cdot v = n \cdot u = n \cdot v = 0 \), that first column has to be \( n/||n||^2 \). Hence, we have

\[
\partial p^*(c_0) = -[\partial_p F(c_0, p_0)]^{-1} \partial_c F(c_0, p_0) = -\frac{n}{||n||^2} \partial_c F(c_0, p_0) \in \mathbb{R}^{3 \times m}.
\]

Recall that \( p^* \) maps a code \( c \) in the neighborhood of \( c_0 \) to a 3D point such that \( M(c, p^*(c)) = f_\Theta(p^*(c), z) = 0 \). In other words, \( p_0 = p^*(c_0) \) is a point on the implicit surface defined by \( f_\Theta \) when \( c = c_0 \) and we have

\[
\frac{\partial p_0}{\partial c} = \frac{n}{||n||^2} \frac{\partial}{\partial c} F(c_0, p_0),
\]

\[
\frac{\partial f_\Theta}{\partial c} = \frac{n}{||n||^2} \frac{\partial}{\partial c} F(z, p_0)
\]

where \( c \) either stands for the latent vector \( z \) or the concatenation of the latent vector and the network weights \( |z|_\Theta \).

In the above proof, the Implicit Function Theorem requires additional constraints to be introduced for the gradients to be well defined. Enforcing those of \((6)\) results in points being mapped to their closest neighbor on the infinitesimally deformed surface. Our gradients stem from this choice.

2) Forward and Backward Passes: Recall that the goal of our forward pass is to extract surface mesh \( M = (V, F) \) from an underlying neural implicit field \( f_\Theta \). Because sampling a DIF on a dense euclidean Grid is computationally intensive, we use a hierarchical approach to reduce the total number of evaluations during the forward and backward passes summarized by Algorithms 1 and 2.

We start by evaluating \( f_\Theta \) on a low resolution grid, and then iteratively subdivide each voxel and re-evaluate the DIF only where needed until we reach a desired grid resolution, as in [2], [54]. When our DIF is a signed distance function, we subdivide voxels if the field absolute value on any of the voxel corners \( \{|f_\Theta(x_i)|\}_{i=1}^3 \) is smaller than the voxel diagonal \( \sqrt{2} \) \( \Delta x \), where \( \Delta x \) denotes voxel size. When it is an occupancy map, we only split voxels when the occupancy map does not have the same value at all corners. For this to work well, we have to start from a grid that roughly captures the object topology to make hierarchical iso-surface extraction converge. In practice, we have found that starting with a \( 32^3 \) grid is enough.

In this way, we can quickly obtain a high resolution DIF grid without needless computation far away from the surface. Once the grid has been assembled, we use a GPU-accelerated marching cubes algorithm [55] to extract the vertices \( v \) and vertex normals \( n \) needed to perform the backward pass. The backward pass then performs the computation of \((5)\). This requires computing the values of \( f_\Theta(z, v) \) and its derivatives \( \frac{\partial f_\Theta}{\partial c}(z, v) \) at the newly found vertices \( v \). We show that the resulting overhead is small in the results section.

In Algorithm 2, we use the mesh normals \( n \) instead of the normalized field gradients \( \sum f_\Theta \) of \((5)\). Preliminary experiments revealed that computing mesh normals is more computationally efficient compared to backpropagating through the network to obtain \( \nabla f_\Theta \) using automatic differentiation. We observed an average angle difference of less than 1.5° between \( n \) and \( \nabla f_\Theta \), and no discernible difference in behavior when using the former as a substitute for the latter.

IV. EXPERIMENTS

We first use synthetic examples to show that, unlike marching cubes, our approach allows for differentiable topology changes.
is a deep signed distance function, we first compute unsigned distances to the surfaces, subtract a small $\epsilon = 0.01$ value and treat the result as a signed distance function. This amounts to representing the garments as watertight thin surfaces of thickness $2\epsilon$. This approximation allows us to use signed distances to represent garments, instead of having to resort to more advanced techniques to model them as single layer meshes with unsigned distance fields [56], [57], [58], [59]. We visualize additional watertight reconstructions in the supplementary material.

In short, $f_{\Theta_1}$ associates to a latent vector $z$ an implicit field $f_{\Theta_1}(z)$ that represent a cow, a duck, or a mix of the two, while $f_{\Theta_2}$ associates to $z$ a garment representation that can be a mixture of the four it was trained on.

1) End-to-End Differentiability: In Fig. 3, we start from a shape $S$ and find a vector $z$ so that $f_{\Theta_1}(z)$ with $x \in \{1, 2\}$ approximates $S$ as well as possible. We then use the pipeline of Section III-B to minimize a differentiable objective function of $z$ so that $f_{\Theta_2}(z)$ becomes an approximation of a different surface $T$. In the following experiments we only optimize the latent vector $z$, while the network parameters $\Theta_x$ are frozen and act as learned shape parameterization.

When using $f_{\Theta_1}$, we take the differentiable objective function to be minimized to be the chamfer distance between the current surface $C$ and the target surface $T$

$$L_{\text{task1}}(C, T) = \min_{c \in C} d(c, T) + \min_{t \in T} d(C, t). \quad (12)$$

where $d$ is the point-to-surface distance in 3D. When using $f_{\Theta_2}$, we take it to be

$$L_{\text{task2}}(C, T) = ||\text{DR}(C) - \text{DR}(T)||_1, \quad (13)$$

where $\text{DR}$ is the output of a differentiable rasterizer [11] rendering binary silhouettes. In other words, $L_{\text{task1}}$ is the surface-to-surface distance while $L_{\text{task2}}$ is the image-to-image $L_1$ distance between the two rendered surfaces.

In both cases, the left shape smoothly turns into the right one, and changes its genus to do so. Note that even though we rely on a deep implicit field to represent our topology-changing surfaces, unlike in [14], [15], [16], [42], we did not have to reformulate the loss functions in terms of implicit surfaces.

2) Comparison to Implicit Field Differentiable Rendering: Recent advances in differentiable rendering [14], [42], [43] have shown that is possible to render continuous SDFs differentiably by carefully designing a differentiable version of the sphere tracing algorithm. By contrast, we simply use DeepMesh’s end-to-end differentiability to exploit an off-the-shelf differentiable rasterizer of meshes to achieve the same result.

To highlight the advantages of doing so, we take $f_{\Theta_1}$, initialize the latent code $z$ to the one of the cow, and then minimize the silhouette distance $L_{\text{task2}}$ with respect to the duck. In Table I we compare our approach to [42]. Sphere tracing requires to query the network along each camera ray in a sequential fashion, resulting in longer computational time with respect to our

---

**Algorithm 2. DeepMesh Backward**

1: **input:** upstream gradient $\frac{\partial C}{\partial v}$ for $v \in V$
2: **output:** downstream gradient $\frac{\partial C}{\partial \Theta}$
3: $\frac{\partial C}{\partial \Theta}(v) = -\frac{\partial C}{\partial v} \frac{\partial v}{\partial \Theta}$ for $v \in V$
4: extra pass on samples $\frac{\partial f_{\Theta}}{\partial \Theta}(z, v)$
5: Return $\frac{\partial C}{\partial \Theta} = \sum_{v \in V} \frac{\partial f_{\Theta}}{\partial \Theta}(v) \frac{\partial f_{\Theta}}{\partial \Theta}(v)$

We then demonstrate that we can exploit surface mesh differentiability to outperform state-of-the-art approaches on three very different tasks, Single view 3D Reconstruction, Aerodynamic Shape Optimization, Structural Shape Optimization, and Full Scene 3D Reconstruction from Scans. In these experiments, we use Theorem 1 with $c = z$, that is, we only optimize with respect to shape codes while keeping the network weights fixed. In the final subsection, we discuss an application in which we take $c = \Theta$, that is, we optimize with respect to the network weights.

---

**A. Differentiable Topology Changes**

In the experiment depicted by Fig. 3 we train two separate networks $f_{\Theta_1}$ and $f_{\Theta_2}$ that implement the approximate implicit field of (1). $f_{\Theta_1}$ is a deep occupancy network trained to minimize the loss of (3) on two models of a cow and a rubber duck. They are of genus 0 and 1, respectively. $f_{\Theta_2}$ is a deep signed distance function network trained to minimize the loss of (2) on four different articles of clothing, a t-shirt, a pair of pants, a dress, and a sweater. Note that the clothes are represented as open surface meshes without inside/outside regions. Hence, they are not watertight surfaces. To nevertheless represent them using a signed distance function, we first compute unsigned distances to the surfaces, subtract a small $\epsilon = 0.01$ value and treat the result as a signed distance function. This amounts to representing the garments as watertight thin surfaces of thickness $2\epsilon$. This approximation allows us to use signed distances to represent garments, instead of having to resort to more advanced techniques to model them as single layer meshes with unsigned distance fields [56], [57], [58], [59]. We visualize additional watertight reconstructions in the supplementary material.

In short, $f_{\Theta_1}$ associates to a latent vector $z$ an implicit field $f_{\Theta_1}(z)$ that represent a cow, a duck, or a mix of the two, while $f_{\Theta_2}$ associates to $z$ a garment representation that can be a mixture of the four it was trained on.

---

Fig. 3. Topology-Variant Parameterization. We minimize (a) a surface-to-surface or (b) an image-to-image distance with respect to the latent vector $z$ to transform an initial shape into a target one that has a different genus. This demonstrates that we can backpropagate gradient information from mesh vertices to latent vector while modifying surface mesh topology.
approach, which projects surface triangles to image space and rasterizes them in parallel. Furthermore, our approach requires fewer function evaluation, as we do not need to sample densely the volume around the zero-crossing of the field.

3) Iso-Surface Extraction Method: Our gradients are independent of the meshing procedure and mesh structure to which we apply them. This is demonstrated in Fig. 4 by repeating the optimization of the latent code \( z \) to minimize the surface-to-surface distance \( L_{\text{task1}} \) of (13) using three different approaches to extracting 3D meshes from iso-surfaces: Marching cubes [41], marching tetrahedra [60] and dual contouring [61]. These methods yield different meshes, but the underlying 3D surfaces they represent after optimization are almost identical. For practical purposes and for all other experiments, we use marching cubes due to the availability of efficient implementations [55] that can easily interface with PyTorch [62].

B. Single View 3D Reconstruction

Single view 3D Reconstruction (SVR) has emerged as a standardized benchmark to evaluate 3D shape representations [2], [3], [24], [31], [32], [33], [63], [64], [65], [66], [67]. We demonstrate that it is straightforward to apply our approach to this task on two standard datasets, ShapeNet [49] and Pix3D [68].

1) Differentiable Meshes for SVR.: As in [2], [3], we condition our deep implicit field architecture on the input images via a residual image encoder [69], which maps input images to latent code vectors \( z \). These latent codes are then used to condition the architecture of Section III-A and compute the value of deep implicit function \( f_{\Theta} \). Finally, we minimize \( L_{\text{imp}} \) (1) wrt. \( \Theta \) on a training set of image-surface pairs generated on the ShapeNet Core [49] dataset for the cars and chairs object classes. Each object class is split into 1210 training and 112 testing shapes, each of which is paired with the renderings provided in [33]. 3D supervision points are generated according to the procedure of [1]. To showcase that our differentiability results work with any implicit representation, we train networks that output either signed distance fields or occupancy fields. To this end, we minimize the loss functions (2) and (3), respectively.

We begin by using the differentiable nature of our mesh representation to refine the output of an encoder, as depicted by the top row of Fig. 1. As in many standard methods, we use our encoder to predict an initial latent code \( z \). Then, unlike in standard methods, we refine the predicted shape \( M \), that is, given the camera pose associated to the image and the current value of \( z \), we project the reconstructed mesh back to the image plane so that the projection matches the object silhouette \( S \) in the image as well as possible. To this end, we define the task-specific loss function \( L_{\text{task}} \) to be minimized, as discussed in Section III, in one of two ways:

\[
L_{\text{task3}} = \| \text{DR}_{\text{silhouette}}(M(z)) - T \|_1, \tag{14}
\]

\[
L_{\text{task4}} = \sum_{a \in A} \min_{b \in B} \| a - b \|^2 + \sum_{b \in B} \min_{a \in A} \| a - b \|^2. \tag{15}
\]

In (14), \( T \) denotes the silhouette of the target surface and \( \text{DR}_{\text{silhouette}} \) is the differentiable rasterizer of [11] that produces a binary mask from the mesh generated by the latent vector \( z \). In (15), \( A \subset [-1, 1]^2 \) denotes the 2D coordinates of \( T \)'s external contour while \( B \subset [-1, 1]^2 \) denotes those of the external contour of \( M(z) \). We refer the interested reader to [70] for more details on this objective function. Note that, unlike that of \( L_{\text{task3}} \), the computation of \( L_{\text{task4}} \) does not require a differentiable rasterizer.

Recall that we can use either signed distance functions or occupancy fields to model objects. To compare these two approaches, we ran 400 gradient descent iterations using Adam [52] to minimize either \( L_{\text{task3}} \) or \( L_{\text{task4}} \) with respect to \( z \). This yields four possible combinations of model and loss function and we report their respective performance in Table II. They are expressed in terms of two metrics:

- The 3D Chamfer distance for 10000 points on the reconstructed and ground truth surfaces, in the original ShapeNet Core scaling. The lower, the better.
- A normal consistency score in image space computed by averaging cosine similarities between reconstructed and

---

Table I: Comparison to Implicit Field Differentiable Rendering

| Method                     | \( 10^3 \cdot l_2 \) silhouette distance ↓ | # network queries ↓ | run time [s] ↓ |
|----------------------------|-------------------------------------------|---------------------|----------------|
| Sphere Tracing [42]        | 5.97                                      | 898k                | 1.24           |
| DeepMesh [isosurface at 256^3, 512^2 pixel renderings] | 4.63 | 266k | 0.29 |

To fit a 2D silhouette, rendering the implicit field with sphere tracing [42] is slower and less effective than extracting an explicit mesh with our method and rendering it with an off-the-shelf mesh rasterizer [11].
TABLE II
SVR ABLATION STUDY ON SHAPENET CORE

| Metric | Model | Refine | car | chair |
|--------|-------|--------|-----|-------|
| CHD $10^4$ ↓ | Occ. DR | 3.02 | 11.18 |
| | CHD | 2.86 (↓ 5.3%) | 10.92 (↓ 2.3%) |
| | SDF DR | 2.65 (↓ 12.3%) | 10.35 (↓ 7.4%) |
| | None | 2.96 | 9.07 |
| | SDF | 2.73 (↓ 7.8%) | 8.83 (↓ 2.6%) |
| | None | 2.56 (↓ 13.5%) | 8.22 (↓ 9.4%) |

NC % ↑

| Metric | Model | Refine | car | chair |
|--------|-------|--------|-----|-------|
| CHD $10^4$ ↓ | Occ. DR | 92.17 | 77.26 |
| | CHD | 92.07 (↓ 0.1%) | 78.98 (↑ 2.2%) |
| | SDF DR | 92.36 (↑ 0.2%) | 78.49 (↑ 1.6%) |
| | None | 92.29 | 78.74 |
| | SDF | 92.22 (↑ 0.1%) | 80.02 (↑ 1.6%) |
| | None | 92.56 (↑ 0.3%) | 80.17 (↑ 1.8%) |

We exploit end-to-end differentiability to perform image-based refinement using either occupancy maps (Model=Occ.) or signed distance functions (Model=SDF). We report 3D Chamfer distance (Metric=CHD) and normal consistency (Metric=NC) for raw reconstructions (Refine=None), refinement via differentiable rendering (Refine=DR) and contour matching (Refine=CHD).

Fig. 5. CHD improvement over the 400 refinement iterations of DeepMesh for cars (top) and chairs (bottom), grouped by quartile in the initial CHD value (from orange = worse 25% of initial shapes, to blue = top 25%). Here DeepMesh uses an SDF and refines the contour matching. Vertical axes are in log scale, and transparent areas show ± the standard deviation for each quartile.

ground truth rendered normal maps from 8 regularly spaced viewpoints. The higher, the better.

All four configurations deliver an improvement in terms of both metrics. However, the combination of using a signed distance field and minimizing the 2D chamfer distance of $L_{\text{task}}$ delivers the largest one. We will therefore refer to it as DeepMesh and use it in the remainder of this section, unless otherwise specified. We hypothesize that signed distance networks perform better due to the 3D supervision points being generated according to the procedure of [1], which might favor SDF networks over Occupancies.

In Fig. 5 we show the Chamfer distance changing over the 400 refinement iterations of DeepMesh on both car and chair categories. We group the test shapes into quartiles according to their initial Chamfer distance with their corresponding ground truth mesh, and compute the average of each quartile. The Chamfer distance is mostly improved for shapes that have a high initial reconstruction error. For the 3 quartiles that have the best initial reconstruction accuracy, the CHD decrease is smaller and mostly takes place during the first iterations. Although the decrease is small for the first quartile, there still is an improvement from 1.27 to 1.24 for cars, and from 2.65 to 2.35 for chairs.

2) Comparative Results on ShapeNet: In Table III, we compare our approach against state-of-the-art reconstruction approaches of watertight meshes: Generating surface meshes with fixed topology [65], generating meshes from voxelized intermediate representations [67], and representing surface meshes using signed distance functions [33]. We used the standard train/test splits and renderings described above for all benchmarked methods.

DeepMesh (raw) refers to reconstructions obtained using our encoder-decoder architecture based on signed distance fields but without refinement, which is similar to those of [2], [3], without any further refinement, whereas DeepMesh incorporates the final refinement that the differentiability of our approach allows. DeepMesh (raw) performs comparably to the other methods whereas DeepMesh does consistently better. In other words, the improvement can be ascribed to the refinement stage as opposed to differences in network architecture. We provide additional results and describe failure modes in the supplementary material.

3) Comparative Results on Pix3D: Whereas ShapeNet contains only rendered images, Pix3D [68] is a test dataset that comprises real images paired to 3D models. Here, we focus on the chair object category and discard truncated images to create a test set of 2530 images. We use it to compare our method with our best competitor [33] according to Table III. To this end, we use the same networks as for ShapeNet, that is, we do not fine-tune the models on Pix3D images. Instead, we train them only on synthetic chair renderings so as to encourage the learning of stronger shape priors. Testing these networks on real images introduces a large domain gap because synthetic renderings do not account for complex lighting effects or variations in camera intrinsic parameters.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
We report our results in Table IV and in Fig. 6. Interestingly, in this more challenging setting using real-world images, our simple baseline DeepMesh (raw) already performs on par with more sophisticated methods that use camera information [33]. As for ShapeNet, our full model DeepMesh outperforms all other approaches.

### C. Aerodynamic Shape Optimization

Computational Fluid Dynamics (CFD) plays a central role in designing cars, airplanes and many other machines. It typically involves approximating the solution of the Navier-Stokes equations using numerical methods. Because this is
computationally demanding, surrogate methods [4], [5], [71], [72] have been developed to infer physically relevant quantities, such as pressure fields, drag, and lift directly from 3D surface meshes without performing actual physical simulations. This makes it possible to optimize these quantities with respect to the 3D shape using gradient-based methods and at a much lower computational cost.

In practice, the space of all possible shapes is immense, and directly optimizing the vertices of a template car would result in invalid meshes. Therefore, for the optimization to work well, one has to parameterize the space of possible shape deformations, which acts as a strong regularizer. In [4], [5] hand-crafted surface parameterizations were introduced. It was effective but not generic and had the potential to significantly restrict the space of possible designs. We show here that we can use DeepMesh to improve upon hand-crafted parameterizations.

1) Experimental Setup: We started with the ShapeNet car split by automatic deletion of all the internal car parts [73] and then manually selected $N = 1400$ shapes suitable for CFD simulation. For each surface $M_i$, we ran OpenFoam [74] to predict a pressure field $p_i$, exerted by air traveling at 15 meters per second towards the car. The resulting training set $\{M_i, p_i\}_{i=1}^N$ was then used to train a Mesh Convolutional Neural Network [75] $g_\beta$ to predict the pressure field $p = g_\beta(M)$, as in [4]. We use $\{M_i\}_{i=1}^N$ to also learn the representation of Section III-B and train the network that implements $f_\Theta$ of (1). As in Section IV-B, we train both an occupancy network and signed-distance network, which we dub DeepMesh-OCC and DeepMesh-SDF, respectively.

The shapes are deformed to minimize the aerodynamic objective function

$$L_{\text{task5}}(M) = \int_M g_\beta \mathbf{n}_x \, dM + \mathcal{L}_{\text{constraint}}(M) + \mathcal{L}_{\text{reg}}(M),$$

(16)

where $\mathbf{n}_x$ denotes the projection of surface normals along airflow direction, the integral term approximates drag given the predicted pressure field [76], $\mathcal{L}_{\text{constraint}}$ is a loss that forces the result to preserve space the engine and the passenger compartment, and $\mathcal{L}_{\text{reg}}$ is a regularization term that prevents $x$ from moving too far away from known shapes. $\mathcal{L}_{\text{constraint}}$ and $\mathcal{L}_{\text{reg}}$ are described in more detail in the supplementary material. $L_{\text{task5}}$ is formulated as a global optimization, and does not explicitly encourage the optimized shape to adhere to the initial one. However, because of the complex landscape of the latent space, different initializations converge to different shapes, as visualized in the supplementary material.

2) Comparative Results: We compare our surface parameterizations to several baselines: (1) vertex-wise optimization, that is, optimizing the objective with respect to each vertex; (2) scaling the surface along its 3 principal axis; (3) using the FreeForm parameterization of [4], which extends scaling to higher order terms as well as periodical ones and (4) the PolyCube parameterization of [5] that deforms a 3D surface by moving a pre-defined set of control points.

We report quantitative results for the minimization of the objective function of (16) for a subset of 8 randomly chosen cars in Table V, and show qualitative ones in Fig. 7. Not only does our method deliver lower drag values than the others but, unlike them, it allows for topology changes and produces semantically correct surfaces as shown in Fig. 7(c). We provide additional results in the supplementary material. As can be seen in Table II, DeepMesh-SDF slightly outperforms DeepMesh-OCC. We conjecture this is due to our sampling strategy for supervision points, which follows closely the one of [1] and might therefore favor SDF networks.

D. Structural Shape Optimization

We investigated another application of DeepMesh as a data-driven parameterization for optimizing physical attributes of 3D surfaces which can change their topology. More specifically, we considered the optimization of cantilever beams with respect to two conflicting objectives, minimizing the stress under load while minimizing their volume, and thus their weight.

We procedurally generated $N = 7000$ 3D beams of fixed length, but with variable width and height, and a random number—between 0 and 10—of circular holes, as shown in Fig. 8(a). For each resulting surface $M_i$, we used the finite element solver FINO [77] to compute its stress $s_i$ when a load of 1000 Newtons is attached to its right side. As for aerodynamic shape optimization, the resulting training set $\{M_i, s_i\}_{i=1}^N$ was then used to train a Mesh Convolutional Neural Network [75] $g_\beta$ to predict the stress field on the surface $s = g_\beta(M)$. We also use our training set to learn the latent vector representation of Section III-B and train the network that implements $f_\Theta$ of (1) and represents the beams’ shapes in terms of a signed distance function.

As in the case of aerodynamic shape optimization case, shapes can then be deformed to minimize the objective function

$$L_{\text{task6}}(M) = \int_M g_\beta \, dM + \lambda L_{\text{volume}}(M) + \mathcal{L}_{\text{reg}}(M),$$

(17)

with respect to the latent vector representing them. Here, the integral term approximates the mean stress on the surface given the predictions of the network $g_\beta$, $L_{\text{volume}}$ is a loss encouraging

| Parameterization | None | Scaling | FreeForm [4] | PolyCube [5] | DeepMesh-SDF | DeepMesh-OCC |
|------------------|------|---------|-------------|--------------|---------------|---------------|
| Degrees of Freedom | ~ 100k | 3 | 21 | ~ 332 | 256 | 256 |
| Simulated $L_{\text{task}}^\text{reg}$ | not converged | 0.931 ± 0.014 | 0.844 ± 0.171 | 0.841 ± 0.203 | **0.675 ± 0.167** | 0.721 ± 0.154 |

We minimize drag on car shapes comparing different surface parameterizations. Numbers in the table (avg ± std) denote relative improvement of the objective function $L_{\text{task}}^\text{reg}$ for the optimized shape, as obtained by CFD simulation in OpenFoam.
Fig. 7. Drag minimization. Starting from an initial shape (left column), $\mathcal{L}_{\text{task}}$ is minimized using three different parameterizations: FreeForm (top), PolyCube (middle), and our DeepMesh (bottom). The middle column depicts the optimization process and the relative improvements in terms of $\mathcal{L}_{\text{task}}$. The final result is shown in the right column. FreeForm and PolyCube lack a semantic prior, resulting in implausible details such as sheared wheels (orange inset). By contrast, DeepMesh not only enforces such priors but can also effect topology changes (blue inset).

Fig. 8. Stress minimization. (a) Beams are attached to a wall on their left side and bear a constant load on their right side. They are all of the same length, but have variable thickness, height, and number of randomly positioned holes. (b) and (c) Starting from two initial beams, we use our DeepMesh parameterization and minimize both mechanical stresses—shown as colors—and volume. This allows for changing their topology and results in plausible beams.

the beams to have a small volume, $\lambda$ is a scalar balancing the two objectives, and $\mathcal{L}_{\text{reg}}$ is the regularization term of (16). $\mathcal{L}_{\text{volume}}$ penalizes low SDF values on a regular 3D grid $G$ of query points. We write it as

$$
\mathcal{L}_{\text{volume}}(\mathcal{M}) = \sum_{x \in G} -f_\Theta(z, x),
$$

which decreases with the volume of $\mathcal{M}$.

Fig. 8(b) and (c) shows two such optimization, starting from two different initial latent codes. In both cases, holes that are weakening the beams are removed, and the beams undergo drastic topology changes. The optimized shapes have smaller volumes, with a single large opening between the upper and lower parts of the structure.

E. Scene Reconstruction

In the examples of Section IV-B and IV-C, we had access to code and training data that enabled us to compare the performance of SDFs and occupancy grids and found out experimentally that the former tend to perform better. However, there are situations in which we only have access to a network that produces occupancy fields without any easy way to transform it into one that produces SDFs. In this section, we show that our method can differentiate through any implicit representation and that the ability to handle not only SDFs but also occupancy fields is valuable.

We use the pretrained scene reconstruction network of [12] that regresses an occupancy field from sparse point clouds describing indoor scenes. We use 10 k points as in the original paper. A point cloud $P$ is first encoded into a set of feature vectors of size 32. These are stored over a 3D feature grid $G \in \mathbb{R}^{32 \times 32 \times 32}$ and in three projected 2D feature planes $P_1, P_2, P_3 \in \mathbb{R}^{128 \times 128 \times 32}$, all aligned with the input point cloud. These features are linearly interpolated in space and decoded into occupancy values. We transform the resulting occupancy field into a differentiable mesh $\mathcal{M}_z$ using our framework, where $z = [G | P_1 | P_2 | P_3]$ is the concatenation of the feature grids.

Using the differentiability of the mesh, we minimize with respect to $z$ the single-sided Chamfer distance

$$
\mathcal{L}_{\text{task7}} = \sum_{p \in P} \min_{a \in \mathcal{M}_z} \|a - b\|^2,
$$

between the reconstructed mesh and $P$, where $a \in \mathcal{M}_z$ represents 10 k points sampled over the mesh.

In Table VI we compute the average Chamfer distance and Intersection over Union (IoU) [12] with the ground truth meshes
for the 2 provided test scenes. The improvements are substantial, as can be seen in Figs. 1(c) and 9.

### F. End-to-End Training

In all previous examples, we considered a pre-trained network \( f_\Theta \) and optimized with respect to the latent variables it takes as input. We now demonstrate that our differentiable iso-surface extraction scheme can also be used to train \( f_\Theta \) and to backpropagate gradients to the network weights \( \Theta \). In other words, we consider the setting where \( c = \Theta \) in Theorem 1 and show how it can be used to improve the performance of a DeepSDF network.

Let us therefore assume that the network \( f_\Theta \) implements DeepSDF, as described in [1]. In the original paper, \( \Theta \) along with the latent representations are learned by minimizing the implicit loss function \( L_{\text{imp}} \) of (1) and its accuracy is assessed in terms of the chamfer distance between target shapes and reconstructed ones. It can be written as

\[
L_{\text{chamfer}} = \sum_{p \in P} \min_{q \in Q} \| p - q \|_2^2 + \sum_{q \in Q} \min_{p \in P} \| p - q \|_2^2 ,
\]

where \( P \) and \( Q \) denote surface samples, we use 10 K points in our implementation. Computing \( L_{\text{chamfer}} \) requires triangulating, which can be done differentially in our framework. This gives us the option to train \( f_\Theta \) not only by minimizing \( L_{\text{chamfer}} \), as in the original method, but by minimizing \( L_{\text{imp}} + L_{\text{chamfer}} \). In other words, we can optimize directly with respect to a relevant metric.

In practice, we first minimize \( L_{\text{imp}} \) to learn a first version of \( \Theta \) and of the latent vectors \( z \) of (1). As described in [1], this yields a network that we will refer to as DeepSDF. We then freeze the latent vectors and minimize \( L_{\text{imp}} + L_{\text{chamfer}} \) with respect to \( \Theta \). This yields a second network that we will refer to as DeepMesh. We do this for the chairs and lamps categories of ShapeNet [49]. For chairs, we use the same data split and samples as in Section IV-B. We apply the same pre-processing steps to lamps, remove duplicates from the original dataset, and use 1100 training shapes and 106 testing shapes.

We compare DeepSDF and DeepMesh qualitatively in Fig. 10 and quantitatively in Table VII, where we report metrics on the test sets by fitting latent codes to SDF samples of unseen shapes. Minimizing \( L_{\text{chamfer}} \) delivers a substantial boost. This is especially true for lamps because they feature thin structures for which even a small error in the predicted SDF values can result in a substantial surface misalignment.

One limitation of our approach is that we cannot use it to train a network from scratch because our gradient computation is only valid at the iso-surface. To ensure the field remains a valid implicit representation (e.g., signed distance) throughout the entire volume, regularization is necessary - an issue we explore further in the next section. In this specific scenario, where volumetric supervision is available, we simply initialize the network using \( L_{\text{imp}} \) and retain this term during the fine-tuning.
phase. An alternative approach involves initializing the signed distance function with that of a sphere [40].

G. Failure Case: Vanishing Surface
As explained above, end-to-end training cannot be done from scratch, because our gradients require the implicit field to already represent a valid surface. Here, we examine another failure case that arises when training end-to-end with unsufficient regularization.

We start with the network $f_{\Theta_1}$ of Section IV-A, which was initially trained to represent a toy cow and a rubber duck, and initialize the latent code $z$ to that of the cow. We then jointly optimize the code $z$ and network weights $\Theta_1$ to conform to a new shape, the Stanford bunny $B$. We do so by using our gradients and applying a surface to surface distance directly on the output mesh $\mathcal{M}$, with no other form of regularization or supervision of the implicit field. We minimize

$$L_{\text{task1}}(\mathcal{M}_{\Theta_1}(z)) = \min_{m \in \mathcal{M}} d(m, B) + \min_{b \in B} d(\mathcal{M}, b).$$

with respect to $z$ and $\Theta_1$ with Adam [52] for 800 steps, where $d$ is the point-to-surface distance in 3D.

Fig. 11 illustrates that we achieve a reasonable fit of the bunny within 500 iterations. However, following this point, degenerate surfaces emerge and eventually disappear entirely. In this case, fitting a useen shape with a weak prior turned the field into an invalid signed distance, and failed. To avoid this issue, regularization techniques can be employed, including:

- early stopping (Fig. 11 for $i = 500$);
- initializing the network weights to some valid implicit field by pretrained on a dataset and keeping them frozen (as in Section IV-A to IV-E), or initializing the network weights to match a 3D sphere (as in [40]);
- simultaneously supervising both the implicit field and the mesh surface (as in Section IV-F);
- adding consistency terms for the implicit field, such an eikonal regularization on the gradients (as in [78]).

H. Execution Speed
We now turn to measuring the execution speed of our method and the overhead it incurs over a simple supervision of implicit fields values. In Table VIII, we compare forward and backward times for losses either on the field’s values ($L_{\text{data}}$ of (2)) or through iso-surface extraction ($L_{\text{chamfer}}$ of (19)). The network is a DeepSDF with 8 layers of size 512, and we report average times over the testing chairs of ShapeNet. For $L_{\text{data}}$, we apply it on the default amout of 8192 points per batch. For $L_{\text{chamfer}}$, we run iso-surface extraction at resolution 128³. The machine we use is an NVidia V100 GPU with an Intel Xeon Gold 6240 CPU.

A naive iso-surface extraction is 2 orders of magnitude slower than simply computing SDF values. However, with the coarse-to-fine strategy presented in Section III-B2, the overhead is reasonable and allows for efficient training. Note also that the backwards pass of our method is slightly faster than with direct supervision of SDF values. This is because we backpropagate from the surface points only, instead of samples over the entire volume.

V. Conclusion
We have introduced DeepMesh, a new approach to extracting 3D surface meshes from continuous deep implicit fields while preserving end-to-end differentiability. This makes it possible to combine powerful implicit models with objective functions requiring explicit representations such as surface meshes.

DeepMesh has the potential to become a powerful Computer Assisted Design tool because allowing differential topology changes of explicit surface parameterizations opens the door to new applications. In future work, we will further extend our paradigm to Unsigned Distance Functions [57] to handle open surfaces without having to thicken them, as we did here. We also plan to exploit Generative Adversarial Networks operating
on surface meshes [79] to increase the level of realism of the surfaces we generate. Furthermore, our method still requires 3D supervision on the field at training time. In the future, we plan to address this with recent approaches that allow learning implicit representations from raw data [78].

REFERENCES

[1] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove, “DeepSDF: Learning continuous signed distance functions for shape representation,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2019, pp. 165–174.

[2] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupy networks: Learning 3D reconstruction in function space,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2019, pp. 4460–4470.

[3] Z. Chen and H. Zhang, “Learning implicit fields for generative shape modeling,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2019, pp. 5932–5941.

[4] P. Baquè, E. Remelli, F. Fleuriet, and P. Fua, “Geodesic convolutional shape optimization,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 472–481.

[5] N. Umetani and B. Bickel, “Learning three-dimensional flow for interactive aerodynamic design,” ACM Trans. Graph., vol. 37, no. 4, 2018, Art. no. 89.

[6] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, “Mitsuba 2: A retablegable forward and inverse renderer,” ACM Trans. Graph., vol. 38, no. 6, pp. 1–17, 2019.

[7] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove, “Decomposing implicit functions,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 523–540.

[8] W. Lorensen and H. Cline, “Marching cubes: A high resolution 3D surface construction algorithm,” in Proc. ACM SIGGRAPH, 1987, pp. 163–169.

[9] T. Newman and H. Yi, “A survey of the marching cubes algorithm,” Comput. Graph., vol. 30, no. 5, pp. 854–879, 2006.

[10] Y. Liao, S. Domé, and A. Geiger, “Deep marching cubes: Learning explicit surface representations,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2916–2925.

[11] H. Kato, Y. Ushiku, and T. Harada, “Neural 3D mesh renderer,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 3907–3917.

[12] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, “Convolutional occupancy networks,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 523–540.

[13] U. Wickramasinghe, E. Remelli, G. Knott, and P. Fua, “Voxel2mesh: 3D mesh model generation from volumetric data,” in Proc. Conf. Med. Image Comput. Comput. Assist. Interv., 2020, pp. 299–308.

[14] Y. Jiang, D. Ji, Z. Han, and M. Zwick, “Sdfdiff: Differentiable rendering of signed distance fields for 3D shape optimization,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1251–1261.

[15] S. Liu, S. A. W. Chen, and H. Li, “Learning to infer implicit surfaces without 3D supervision,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8295–8306.

[16] M. Michalkiewicz, J. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksdotter, “Implicit surface representations as layers in neural networks,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2019, pp. 711–729.

[17] Z. Wu et al., “3D shapenets: A deep representation for volumetric shapes,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1912–1920.

[18] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, “Pix2vox: Context-aware 3D reconstruction from single and multi-view images,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2019, pp. 5045–5054.

[19] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann, “DISN: Deep implicit surface network for high-quality single-view 3D reconstruction,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 492–502.

[20] J. Chibane, T. Alldieck, and G. Pons-Moll, “Implicit functions in feature space for 3D shape reconstruction and completion,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 6970–6981.

[21] S. Peng, W. Jiang, H. Pi, X. Li, H. Bao, and X. Zhou, “Deep snake for real-time instance segmentation,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 8533–8542.

[22] Y. D. Fougerolle, A. Gribok, S. Foufou, F. Truchetet, and M. A. Abidi, “Boolean operations with implicit and parametric representation of primitives using R-functions,” IEEE Trans. Vis. Comput. Graph., vol. 11, no. 5, pp. 529–539, Sep./Oct. 2005.

[23] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, “Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 3504–3515.

[24] L. Yariv et al., “Multiview neural surface reconstruction by disentangling geometry and appearance,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 2492–2502.

[25] J. Stam and R. Schmidt, “On the velocity of an implicit surface,” ACM Trans. Graph., vol. 30, no. 3, pp. 1–7, 2011.

[26] I. Mehta, M. Chandraker, and R. Ramamoorthi, “A level set theory for neural implicit evolution under explicit flows,” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 711–729.

[27] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, “Efficient implementation of marching cubes’ cases with topological guarantees,” in Proc. J. Graph. Tools, 2003, pp. 1–15.

[28] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, “Dist: Rendering deep implicit signed distance function with differentiable sphere tracing,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 2492–2502.

[29] D. Vicini, S. Speierer, and W. Jakob, “Differentiable signed distance function rendering,” ACM Trans. Graph., vol. 41, pp. 1–18, 2022.

[30] A. Kanazawa, S. Tulsiani, A. Efros, and J. Malik, “Learning category-specific mesh reconstruction from image collections,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 371–386.

[31] P. Henderson and V. F. Blanz, “Learning single-image 3D reconstruction by generative modelling of shape, pose and shading,” Int. J. Comput. Vis., vol. 128, no. 4, pp. 835–854, 2020.

[32] V. Blanz and T. Vetter, “A morphable model for the synthesis of 3D faces,” Neural Inf. Process. Syst., 1999, pp. 187–194.

[33] T. Bagautdinov, C. Wu, J. Saragih, P. Fua, and Y. Sheikh, “Modeling facial geometry using compositional VAEs,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 3877–3886.
Proc. Int. Conf. Comput. Vis.

[48] E. Remelli, A. Tkach, A. Tagliasacchi, and M. Pauly, “Low-dimensionality calibration through local anisotropic scaling for robust hand model personalization,” in Proc. Int. Conf. Comput. Vis., 2017, pp. 2535–2543.

[49] R. Chang et al., “ShapeNet: An information-rich 3D model repository,” 2015, arXiv:1512.03012.

[50] K. Park, T. Pattan, and M. Vincze, “Pix2pose: Pixel-wise coordinate regression of objects for 6D pose estimation,” in Proc. Int. Conf. Comput. Vis., 2019, pp. 7668–7677.

[51] G. Allaire, F. Jouve, and A.-M. Toader, “A level-set method for shape optimization,” Comptes Rendus Mathématiques, vol. 334, no. 12, pp. 1125–1130, 2002.

[52] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimisation,” in Proc. Int. Conf. Learn. Representations, 2015.

[53] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” J. Mach. Learn. Res., vol. 12, pp. 2121–2159, 2011.

[54] R. Venkatesh et al., “Deep implicit surface point prediction networks,” in Proc. Int. Conf. Comput. Vis., 2021, pp. 12633–12642.

[55] T. Yamagata, “mcubes_pytorch: PyTorch implementation for marching cubes,” 2020. [Online]. Available: https://github.com/tatsy/mcubes_pytorch/

[56] F. Zhao, W. Wang, S. Liao, and L. Shao, “Learning anchored un-signed distance functions with gradient direction alignment for single-view garment reconstruction,” in Proc. Int. Conf. Comput. Vis., 2021, pp. 12654–12663.

[57] B. Guillard, F. Stella, and P. Fua, “MeshUDF: Fast and differentiable meshing of unsigned distance field networks,” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 576–592.

[58] J. Chabane, A. Mir, and G. Fons-Moll, “Neural unsigned distance fields for implicit function learning,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 21638–21652.

[59] X. Long et al., “NeuralUDF: Learning unsigned distance fields for multi-view reconstruction of surfaces with arbitrary topologies,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2022, pp. 20834–20843.

[60] A. Dei and A. Koide, “An efficient method of triangulating equivalved surfaces by using tetrahedral cells,” Transactopms Inf. Syst., vol. 74, no. 1, pp. 214–224, 1991.

[61] T. Ju, F. Losasso, S. Schaefer, and J. Warren, “Dual contouring of hermite data,” in Proc. ACM SIGGRAPH, 2002, pp. 339–346.

[62] A. Paszke et al., “Automatic differentiation in PyTorch,” in Proc. Nat. Acad. Sci., 2017.

[63] H. Fan, H. Su, and L. Guibas, “A point set generation network for 3D object reconstruction from a single image,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2017, pp. 605–613.

[64] T. Groueix, M. Fisher, V. Kim, B. Russell, and M. Aubry, “Atlasnet: A Papier-Mâché approach to learning 3D surface generation,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 216–224.

[65] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y. Jiang, “Pixel2mesh: Generating 3D mesh models from single RGB images,” in Proc. Euro. Conf. Comput. Vis., 2018, pp. 52–67.

[66] J. K. Pontes, C. Kong, S. Sridharan, S. Lucey, A. Eriksson, and C. Fookes, “Image2mesh: A learning framework for single image 3D reconstruction,” in Proc. Asian Conf. Comput. Vis., 2018, pp. 365–381.

[67] G. Gkioxari, J. Malik, and J. Johnson, “Mesh R-CNN,” in Proc. Int. Conf. Comput. Vis., 2019, pp. 9785–9795.

[68] S. Xingyuan et al., “Pix3D: Dataset and methods for single-image 3D shape modeling,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2974–2983.

[69] H. Ke, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–777.

[70] B. Guillard, E. Remelli, P. Yvermay, and P. Fua, “Sketch2Mesh: Reconstructing and editing 3D shapes from sketches,” in Proc. Int. Conf. Comput. Vis., 2021, pp. 13003–13012.

[71] D. Toal and A. Keane, “Efficient multipoint aerodynamic design optimization via cokriging,” J. Aircr., vol. 48, no. 5, pp. 1685–1695, 2011.

[72] G. Xu, X. Liang, S. Yao, D. Chen, and Z. Li, “Multi-objective aerodynamic optimization of the streamlined shape of high-speed trains based on the kriging model,” PLoS One, vol. 12, no. 1, pp. 1–14, 2017.

[73] F. S. Sim, D. Schroeder, and J. Barbić, “Vega: Non-linear FEM deformable object simulator,” in Proc. Comput. Graph. Forum, 2013, pp. 36–48.

[74] H. Jasak et al., “OpenFOAM: A C library for complex physics simulations,” in Proc. Int. Workshop Coupled Methods Numer. Dyn., 2007, pp. 1–20.

[75] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, “SplineCNN: Fast geometric deep learning with continuous B-spline kernels,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 869–877.

[76] B. Munson, T. Okishi, W. Huebsch, and A. Rothmayer, Fluid Mechanics. New York, NY, USA: Wiley, 2013.

[77] Seamplex, “Fino, a free finite element solver,” 2016. [Online]. Available: https://github.com/seamplex/fino

[78] M. Atzmon and Y. Lipman, “SAL: Sign agnostic learning of shapes from raw data,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2020, pp. 2565–2574.

[79] S. Cheng et al., “MeshGAN: Non-linear 3D morphable models of faces,” 2019, arXiv:1903.10384.

[80] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” 2016, arXiv:1607.08022.

[81] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, “DeePSdf: Learning continuous signed distance functions for shape representation,” in Proc. Comput. Vis. Pattern Recognit., 2019, pp. 165–174.

[82] A. Mosińska, P. Marquez-Neila, M. Kozinski, and P. Fua, “Beyond the pixel-wise loss for topology-aware delineation,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 3136–3145.

[83] D. Oner, M. Kozik"{i}ski, L. Citraro, N. C. Dadap, A. G. Konings, and P. Fua, “Promoting connectivity of network-like structures by enforcing region separation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 9, pp. 5401–5413, Sep. 2022.

[84] D. Oner, A. Garin, M. Kozik"{i}ski, K. Hess, and P. Fua, “Persistent homology with improved locality information for more effective delineation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 8, pp. 10588–10595, Aug. 2023.

[85] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model CNNs,” in Proc. Conf. Comput. Vis. Pattern Recognit., 2017, pp. 5425–5434.

Benoît Guillard is currently working toward the PhD degree with EPFL CVLab with Prof. Pascal Fua, working on deep learning methods to represent and manipulate 3D objects.

Edoardo Remelli is a research scientist with Meta Reality Labs Research, working on egocentric action recognition. He holds a PhD in computer vision from EPFL, and his research there has focused on articulated pose estimation, geometric deep learning, and neural rendering.
Artem Lukoianov is currently working toward the PhD degree with MIT advised by Prof. Justin Solomon and Prof. Vincent Sitzmann, working on computer vision, computer graphics, neural signal representations, and equivariant learning. He previously was a researcher at neural concept and EPFL CVLab.

Pierre Yvernay received the master’s degrees in theoretical physics from ENS and computational mathematics from ENSTA, in France. He is a Product and Software Engineer with Neural Concept. He previously was a Research Engineer with EPFL CVLab.

Stephan R. Richter received the BSc and MSc degrees in IT systems engineering from the Hasso-Plattner-Institute in Potsdam, Germany, and the PhD degree in computer science from TU Darmstadt, Germany. He leads a research team with Apple. Previously, he was a research scientist with the Intelligent Systems Lab at Intel. His research interests span topics in computer vision and related areas.

Timur Bagautdinov is a research scientist with Meta Reality Labs Research, working on photorealistic telepresence and neural rendering. He holds a PhD from EPFL CVLab.

Pierre Baque is the CEO of Neural Concept, and holds a PhD in computer vision from EPFL with a focus on Bayesian reasoning, Graphical Models and 3D Shape Optimization.

Pascal Fua (Fellow, IEEE) is a professor of computer science with EPFL, Switzerland. His research interests include shape and motion reconstruction from images, analysis of microscopy images, and Augmented Reality. He is an IEEE Fellow and has been an associate editor of the IEEE journal Transactions for Pattern Analysis and Machine Intelligence.