

DeepMesh: Differentiable Iso-Surface Extraction

Benoît Guillard, Edoardo Remelli, Artem Lukoianov, Pierre Yvernay, Stephan R. Richter, Timur Bagautdinov, Pierre Baque, 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 via Differentiable Rendering, Physically-Driven 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.



1 INTRODUCTION

Geometric 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 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.

• B. Guillard, E. Remelli and P. Fua are with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne, Switzerland. E-mail: {benoit.guillard, edoardo.remelli, pascal.fua}@epfl.ch. A. Lukoianov is with the Group of Geometrical Data Processing at MIT. E-mail: arteml@mit.edu. P. Yvernay and P. Baque are with Neural Concept. E-mail: {pierre.yvernay,pierre.baque}@neuralconcept.com. S.R. Richter is with Intel Labs.. T. Bagautdinov is with Meta Reality Labs. E-mail: timurb@meta.com.

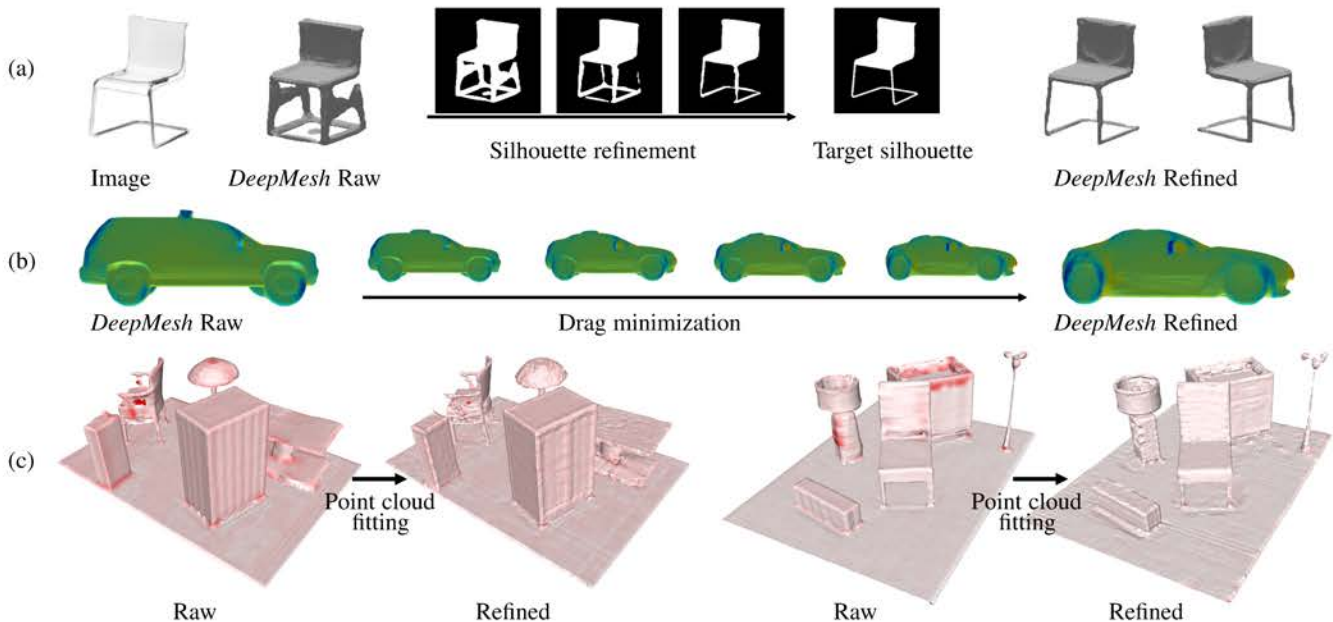


Fig. 1. **DeepMesh**. (a) We condition our representation on an input image and output an initial 3D mesh, which we refine via differentiable rasterization [11], thereby exploiting DeepMesh’s end-to-end differentiability. (b) We use our parameterization as a powerful regularizer for aerodynamic optimization 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.

- 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 iso-surface 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.

2 RELATED WORK

2.1 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 deep networks 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 Sec. 4.2, 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.

2.2 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.

2.2.1 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.

2.2.2 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.

3 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 $\mathcal{M} = (V, F)$, where

$V = \{\mathbf{v}_1, \mathbf{v}_2, \dots\}$ denotes vertex positions in \mathbb{R}^3 and F facets, to minimize a task-specific loss function $\mathcal{L}_{\text{task}}(\mathcal{M})$. $\mathcal{L}_{\text{task}}$ 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 \mathbf{z} 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 $\mathcal{L}_{\text{task}}$.

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 $\mathcal{L}_{\text{task}}$. Finally, we present *DeepMesh*, a fully differentiable surface mesh parameterization that can represent arbitrary topologies.

3.1 Deep Implicit Field Representation

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

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

$$\mathcal{L}_{\text{imp}}(\{\mathbf{z}_S\}_{S \in \mathcal{D}}, \Theta) = \mathcal{L}_{\text{data}}(\{\mathbf{z}_S\}_{S \in \mathcal{D}}, \Theta) + \lambda_{\text{reg}} \sum_{S \in \mathcal{D}} \|\mathbf{z}_S\|_2^2, \quad (1)$$

where $\mathbf{z}_S \in \mathbb{R}^Z$ is the Z -dimensional encoding of surface S , Θ denotes network parameters, $\mathcal{L}_{\text{data}}$ is a data term that measures how similar f_{Θ} is to the ground-truth function s corresponding to each sample surface, and λ_{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 $\mathcal{L}_{\text{data}}$ to be the L_1 loss

$$\mathcal{L}_{\text{data}} = \sum_{S \in \mathcal{D}} \frac{1}{|X_S|} \sum_{\mathbf{x} \in X_S} |f_{\Theta}(\mathbf{z}_S, \mathbf{x}) - s(\mathbf{x})|, \quad (2)$$

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

$$\mathcal{L}_{\text{data}} = - \sum_{S \in \mathcal{D}} \frac{1}{|X_S|} \sum_{\mathbf{x} \in X_S} s(\mathbf{x}) \log(f_{\Theta}(\mathbf{z}_S, \mathbf{x})) + (1 - s(\mathbf{x})) \log(1 - f_{\Theta}(\mathbf{z}_S, \mathbf{x})). \quad (3)$$

Once trained, s is approximated by f_{Θ} 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 $\{\mathbf{x} \in \mathbb{R}^3, f_{\Theta}(\mathbf{z}_S, \mathbf{x}) = \alpha\}$, when α is zero for SDFs and typically 0.5 for occupancy grids. Since f_{Θ} is defined up to a constant, we will refer to zero-crossings in the rest of the paper for simplicity.

3.2 Differentiable Iso-Surface Extraction

Once the weights Θ of Eq. 1 have been learned, f_Θ maps a latent vector \mathbf{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 $\mathcal{L}_{\text{task}}$ 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

$$\frac{\partial \mathcal{L}_{\text{task}}}{\partial \mathbf{c}} = \sum_{\mathbf{x} \in V} \frac{\partial \mathcal{L}_{\text{task}}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{c}}, \quad (4)$$

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

3.2.1 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_Θ when sampled by the marching cubes algorithm. Some of the same ideas have been explored for implicit differentiation in neural networks [17], [52]. To this end, we prove below the following theorem.

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

Injecting this expression of $\partial \mathbf{x}/\partial \mathbf{c}$ into Eq. 4 yields

$$\frac{\partial \mathcal{L}_{\text{task}}}{\partial \mathbf{c}} = - \sum_{\mathbf{x} \in V} \frac{\partial \mathcal{L}_{\text{task}}}{\partial \mathbf{x}} \frac{\nabla f_\Theta}{\|\nabla f_\Theta\|^2} \frac{\partial f_\Theta}{\partial \mathbf{c}}. \quad (5)$$

Note that when s is an SDF, $\|\nabla s\| = 1$ and therefore $\|\nabla f_\Theta\| \approx 1$. The $\|\nabla f_\Theta\|^2$ numerator from Eq. 5 can then be ignored, which is consistent with the result we presented in [18]. This scaling factor is not 1 for other implicit fields. However, it does not affect the direction of the gradients only their magnitude, which often gets rescaled by optimizers [53], [54] 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 $F : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $\mathbf{c}_0 \in \mathbb{R}^m, \mathbf{p}_0 \in \mathbb{R}^n$ such that:*

- 1) $F(\mathbf{c}_0, \mathbf{p}_0) = 0$;
- 2) F is continuously differentiable in a neighborhood of $(\mathbf{c}_0, \mathbf{p}_0)$;
- 3) the partial Jacobian $\partial_p F(\mathbf{c}_0, \mathbf{p}_0) \in \mathbb{R}^{n \times n}$ is non-singular.

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

- 1) $\mathbf{p}_0 = p^*(\mathbf{c}_0)$;
- 2) $F(\mathbf{c}, p^*(\mathbf{c})) = 0$ for all \mathbf{c} in the above mentioned neighborhood of \mathbf{c}_0 ;

- 3) $\partial p^*(\mathbf{c}_0) = -[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0)$, 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 n unknowns. For our purposes, $\mathbf{c} \in \mathbb{R}^m$ 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_Θ as a function $M : \mathbb{R}^m \times \mathbb{R}^3 \rightarrow \mathbb{R}$ that maps $\mathbf{c} \in \mathbb{R}^m$ and a point in $\mathbf{p} \in \mathbb{R}^3$ to a scalar value $M(\mathbf{c}, \mathbf{p}) \in \mathbb{R}$. 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 .

To this end, let $\mathbf{c}_0 \in \mathbb{R}^m; \mathbf{p}_0 \in \mathbb{R}^3$ such that $M(\mathbf{c}_0, \mathbf{p}_0) = 0$, meaning that \mathbf{p}_0 is on the implicit surface defined by parameter \mathbf{c}_0 ; and $\mathbf{u} \in \mathbb{R}^3$ and $\mathbf{v} \in \mathbb{R}^3$ such that (\mathbf{u}, \mathbf{v}) is a basis of the tangent plane to the surface $\{M(\mathbf{c}_0, \cdot) = 0\}$ at \mathbf{p}_0 . Let $\mathbf{n} = \partial_p M(\mathbf{c}_0, \mathbf{p}_0)$ be the normal vector to the surface at \mathbf{p}_0 . This lets us define the function $F : \mathbb{R}^m \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$ as

$$F(\mathbf{c}, \mathbf{p}) \mapsto \begin{pmatrix} M(\mathbf{c}, \mathbf{p}) \\ (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{u} \\ (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{v} \end{pmatrix}, \quad (6)$$

By construction, we have $\mathbf{n} \cdot \mathbf{u} = \mathbf{n} \cdot \mathbf{v} = 0$ and $F(\mathbf{c}_0, \mathbf{p}_0) = 0$.

Note that the first value of the $F(\mathbf{c}, \mathbf{p})$ vector is zero when the point \mathbf{p} is on the surface defined by \mathbf{c} while the other two are equal to zero when $(\mathbf{p} - \mathbf{p}_0)$ is perpendicular to the surface defined by \mathbf{c}_0 . By zeroing all three, p^* returns a point \mathbf{p} that is on the surface for $\mathbf{c} \neq \mathbf{c}_0$ and such that $(\mathbf{p} - \mathbf{p}_0)$ is perpendicular to the surface. A geometric interpretation is that \mathbf{p}_0 is the point on the surface defined by \mathbf{c}_0 that is the closest to $p^*(\mathbf{c})$. 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) $\mathbf{p}_0 = p^*(\mathbf{c}_0)$;
- 2) $F(\mathbf{c}, p^*(\mathbf{c})) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ for all \mathbf{c} in a neighborhood of \mathbf{c}_0 .
- 3) $\partial p^*(\mathbf{c}_0) = -[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0)$.

We have

$$\partial_c F(\mathbf{c}_0, \mathbf{p}_0) = \begin{pmatrix} \partial_c M(\mathbf{c}_0, \mathbf{p}_0) \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{3 \times m}, \quad (7)$$

$$\partial_p F(\mathbf{c}_0, \mathbf{p}_0) = \begin{pmatrix} \mathbf{n} \\ \mathbf{u} \\ \mathbf{v} \end{pmatrix} \in \mathbb{R}^{3 \times 3} \quad (8)$$

Given that the last two rows of $\partial_c F(\mathbf{c}_0, \mathbf{p}_0)$ are zero, to compute $\partial p^*(\mathbf{c}_0)$ according to the IFT, we only need to evaluate the first column of $[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)]^{-1}$. As the two last rows of $\partial_p F(\mathbf{c}_0, \mathbf{p}_0)$ are \mathbf{u} and \mathbf{v} that are unit vectors such that $\mathbf{u} \cdot \mathbf{v} = \mathbf{n} \cdot \mathbf{u} = \mathbf{n} \cdot \mathbf{v} = 0$, that first column has to be $\mathbf{n}/\|\mathbf{n}\|^2$. Hence, we have

$$\begin{aligned} \partial p^*(\mathbf{c}_0) &= -[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0), \quad (9) \\ &= -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \partial_c F(\mathbf{c}_0, \mathbf{p}_0) \in \mathbb{R}^{3 \times m}. \end{aligned}$$

Recall that p^* maps a code \mathbf{c} in the neighborhood of \mathbf{c}_0 to a 3D point such that $M(\mathbf{c}, p^*(\mathbf{c})) = f_\Theta(p^*(\mathbf{c}), \mathbf{z}) = 0$. In other words, $\mathbf{p}_0 = p^*(\mathbf{c}_0)$ is a point on the implicit surface defined by f_Θ when $\mathbf{c} = \mathbf{c}_0$ and we have

$$\frac{\partial \mathbf{p}_0}{\partial \mathbf{c}} = -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \partial_c F(\mathbf{c}_0, \mathbf{p}_0), \quad (10)$$

$$= -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \frac{\partial f_\Theta(\mathbf{z}, \mathbf{p}_0)}{\partial \mathbf{c}} \quad (11)$$

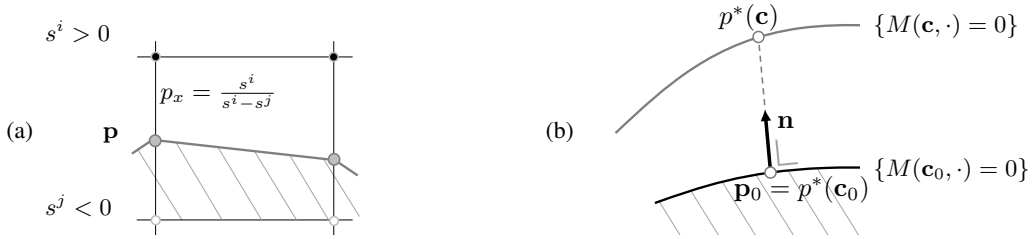


Fig. 2. **Marching Cubes differentiation vs Iso-surface differentiation.** (a) Marching Cubes determines the position p_x of a vertex \mathbf{p} along an edge via linear interpolation. This does not allow for effective back-propagation when topology changes because its behavior is degenerate when $s^i = s^j$ as shown in [10]. (b) Instead, we adopt a *continuous* model expressed in terms of how implicit function perturbations locally impact surface geometry. Here, we depict the geometric relation between implicit parameter perturbation $\mathbf{c}_0 \mapsto \mathbf{c}$ and local surface change $\mathbf{p}_0 \mapsto p^*(\mathbf{c})$, which we exploit to compute $\frac{\partial p^*(\mathbf{c})}{\partial \mathbf{c}}$ even when the topology changes.

where \mathbf{c} either stands for the latent vector \mathbf{z} or the concatenation of the latent vector and the network weights $[\mathbf{z}|\Theta]$. \square

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

3.2.2 Forward and Backward Passes

Recall that the goal of our forward pass is to extract surface mesh $\mathcal{M} = (V, F)$ from an underlying neural implicit field f_Θ . 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_Θ 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], [55]. When our DIF is a signed distance function, we subdivide voxels if the field absolute value on any of the voxel corners $\{|f_\Theta(\mathbf{x}_i)|\}_{i=1}^8$ is smaller than the voxel diagonal $\sqrt{2}\Delta x$, where Δx 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 [56] to extract the vertices \mathbf{v} and vertex normals \mathbf{n} needed to perform the backward pass. The backward pass then performs the computation of Eq. 5. This requires computing the values of $f_\Theta(\mathbf{z}, \mathbf{v})$ and its derivatives $\frac{\partial f_\Theta}{\partial \mathbf{c}}(\mathbf{z}, \mathbf{v})$ at the newly found vertices \mathbf{v} . We show that the resulting overhead is small in the results section.

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

Algorithm 1 DeepMesh Forward

- 1: **input:** latent code \mathbf{z} , DIF weights Θ
 - 2: **output:** surface mesh $\mathcal{M} = (V, F)$
 - 3: assemble coarse 3D grid G
 - 4: sample field on grid $S = f_\Theta(\mathbf{z}, G)$
 - 5: while G has not reached target resolution:
 - 6: $G_s = \text{split}(G)$
 - 7: $S = S + f_\Theta(\mathbf{z}, G_s)$
 - 8: $G = G + G_s$
 - 9: extract iso-surface $(V, F) = \text{MC}(S)$
 - 10: **Return** $\mathcal{M} = (V, F)$
-

Algorithm 2 DeepMesh Backward

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

4 EXPERIMENTS

We first use synthetic examples to show that, unlike marching cubes, our approach allows for differentiable topology changes. 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 $\mathbf{c} = \mathbf{z}$, 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 $\mathbf{c} = \Theta$, that is, we optimize with respect to the network weights.

4.1 Differentiable Topology Changes

In the experiment depicted by Fig. 3 we train two separate networks f_{Θ_1} and f_{Θ_2} that implement the approximate implicit field of Eq. 1. f_{Θ_1} is a deep occupancy network trained to minimize the loss of Eq. 3 on two models of a cow and a rubber duck. They are of genus 0 and 1, respectively. f_{Θ_2} is a deep signed distance function network trained to minimize the loss of Eq. 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

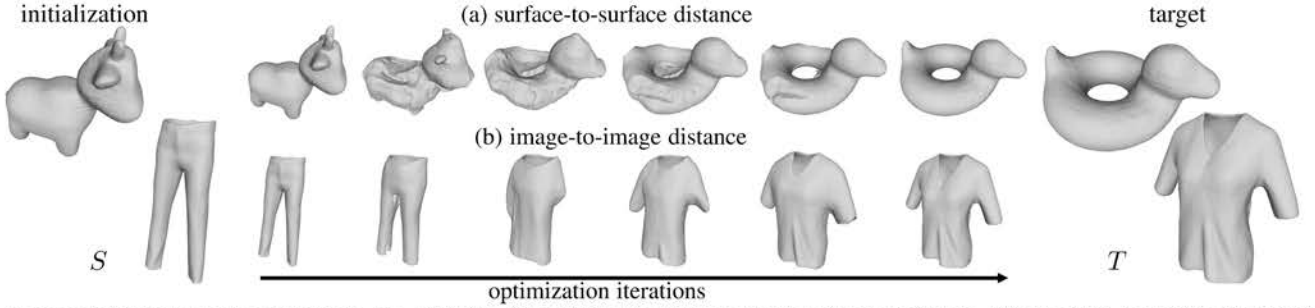


Fig. 3. **Topology-Variant Parameterization.** We minimize (a) a surface-to-surface or (b) an image-to-image distance with respect to the latent vector \mathbf{z} to transform an initial shape into a target one that has a different genus. This demonstrates that we can backpropagate gradient information from mesh vertices to latent vector while modifying surface mesh topology.

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 2ϵ . 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 [57], [58], [59], [60]. We visualize additional watertight reconstructions in the supplementary material.

In short, f_{Θ_1} associates to a latent vector \mathbf{z} an implicit field $f_{\Theta_1}(\mathbf{z})$ that represent a cow, a duck, or a mix of the two, while f_{Θ_2} associates to \mathbf{z} a garment representation that can be a mixture of the four it was trained on.

4.1.1 End-to-end Differentiability

In Fig. 3, we start from a shape S and find a vector \mathbf{z} so that $f_{\Theta_x}(\mathbf{z})$ with $x \in 1, 2$ approximates S as well as possible. We then use the pipeline of Sec. 3.2 to minimize a differentiable objective function of \mathbf{z} so that $f_{\Theta_x}(\mathbf{z})$ becomes an approximation of a different surface T . In the following experiments we only optimize the latent vector \mathbf{z} , while the network parameters Θ_x are frozen and act as a learned shape parameterization.

When using f_{Θ_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

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

where d is the point-to-surface distance in 3D. When using f_{Θ_2} , we take it to be

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

where DR is the output of a differentiable rasterizer [11] rendering binary silhouettes. In other words, $\mathcal{L}_{\text{task1}}$ is the surface-to-surface distance while $\mathcal{L}_{\text{task2}}$ 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.

4.1.2 Comparison to Implicit Field Differentiable Rendering

Recent advances in differentiable rendering [14], [42], [43] have shown that is possible to render continuous SDFs differentially by carefully designing a differentiable version of the sphere tracing

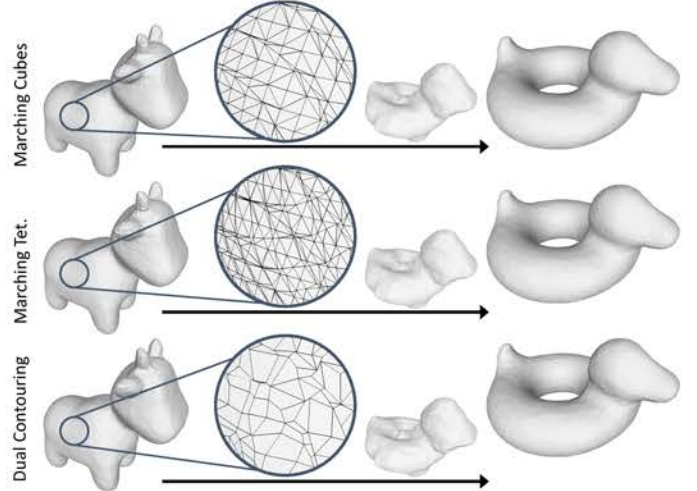


Fig. 4. **Different surface extraction methods** yield the same surface-to-surface optimization as in Fig. 3(a), and our gradients can be used equally well with Marching Cubes (top), Marching Tetrahedra (middle) or Dual Contouring (bottom).

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_{Θ_1} , initialize the latent code \mathbf{z} to the one of the cow, and then minimize the silhouette distance $\mathcal{L}_{\text{task2}}$ with respect to the duck. In Tab. 1 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 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.

4.1.3 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 \mathbf{z} to minimize the surface-to-surface distance $\mathcal{L}_{\text{task1}}$ of Eq. 13 using three different approaches to extracting 3D meshes from iso-surfaces: Marching cubes [41], marching tetrahedra [61] and dual contouring [62]. 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 [56] that can easily interface with PyTorch [63].

TABLE 1

Comparison to Implicit Field Differentiable Rendering. 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].

Method	$10^3 \cdot l_2$ silhouette distance ↓	# network queries ↓	run time [s] ↓
Sphere Tracing [42] [most efficient settings, 512^2 pixel renderings]	5.97	898k	1.24
<i>DeepMesh</i> [isosurface at 256^3 , 512^2 pixel renderings]	4.63	266k	0.29

4.2 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], [64], [65], [66], [67], [68]. We demonstrate that it is straightforward to apply our approach to this task on two standard datasets, ShapeNet [49] and Pix3D [69].

4.2.1 Differentiable Meshes for SVR.

As in [2], [3], we condition our deep implicit field architecture on the input images via a residual image encoder [70], which maps input images to latent code vectors \mathbf{z} . These latent codes are then used to condition the architecture of Sec. 3.1 and compute the value of deep implicit function f_Θ . Finally, we minimize \mathcal{L}_{imp} (Eq. 1) wrt. Θ 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 Eqs. 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 \mathbf{z} . Then, unlike in standard methods, we refine the predicted shape \mathcal{M} , that is, given the camera pose associated to the image and the current value of \mathbf{z} , we project the reconstructed mesh back to the image plane so that the projection matches the object silhouette \mathcal{S} in the image as well as possible. To this end, we define the task-specific loss function $\mathcal{L}_{\text{task}}$ to be minimized, as discussed in Section 3, in one of two ways:

$$\mathcal{L}_{\text{task3}} = \|\text{DR}_{\text{silhouette}}(\mathcal{M}(\mathbf{z})) - T\|_1, \quad (14)$$

$$\mathcal{L}_{\text{task4}} = \sum_{a \in A} \min_{b \in B} \|a - b\|^2 + \sum_{b \in B} \min_{a \in A} \|a - b\|^2. \quad (15)$$

In Eq. 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 \mathbf{z} . In Eq. 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 $\mathcal{M}(\mathbf{z})$. We refer the interested reader to [71] for more details on this objective function. Note that, unlike that of $\mathcal{L}_{\text{task3}}$, the computation of $\mathcal{L}_{\text{task4}}$ 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 [53] to minimize either $\mathcal{L}_{\text{task3}}$ or $\mathcal{L}_{\text{task4}}$ with respect to \mathbf{z} . This yields four possible combinations of model and loss function and we

TABLE 2

SVR ablation study on ShapeNet Core. 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).

Metric	Model	Refine	car	chair
CHD · 10 ⁴ ↓	Occ.	None	3.02	11.18
		DR	2.86 (↓ 5.3%)	10.92 (↓ 2.3%)
		CHD	2.65 (↓ 12.3%)	10.35 (↓ 7.4%)
	SDF	None	2.96	9.07
		DR	2.73 (↓ 7.8%)	8.83 (↓ 2.6%)
		CHD	2.56 (↓ 13.5%)	8.22 (↓ 9.4%)
NC % ↑	Occ.	None	92.17	77.26
		DR	92.07 (↓ 0.1%)	78.98 (↑ 2.2%)
		CHD	92.36 (↑ 0.2%)	78.49 (↑ 1.6%)
	SDF	None	92.29	78.74
		DR	92.22 (↓ 0.1%)	80.02 (↑ 1.6%)
		CHD	92.56 (↑ 0.3%)	80.17 (↑ 1.8%)

report their respective performance in Tab. 2. 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 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 $\mathcal{L}_{\text{task4}}$ 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.

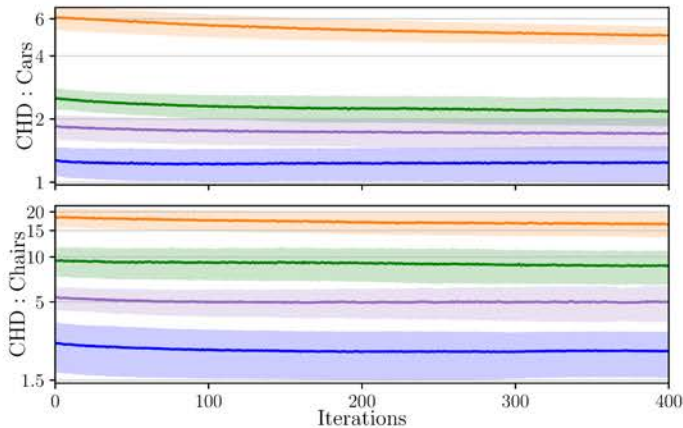


Fig. 5. **CHD improvement** over the 400 refinement iterations of *DeepMesh* for cars (top) and chairs (bottom), grouped by quartile in the initial CHD value (from orange = worse 25% of initial shapes, to blue = top 25%). Here *DeepMesh* uses an SDF and refines the contour matching. Vertical axes are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

4.2.2 Comparative Results on ShapeNet

In Tab. 3, we compare our approach against state-of-the-art reconstruction approaches of watertight meshes: Generating surface meshes with fixed topology [66], generating meshes from voxelized intermediate representations [68], and representing surface meshes using signed distance functions [33]. We used the standard train/test splits and renderings described above for all benchmarked methods.

TABLE 3

SVR comparative results on ShapeNet Core. Exploiting end-to-end differentiability to perform image-based refinement allows us to outperform state-of-the-art methods in terms of both 3D Chamfer distance (CHD) and normal consistency (NC).

Metric	Method	car	chair
CHD $\cdot 10^4 \downarrow$	Mesh R-CNN [68]	4.55	11.13
	Pixel2Mesh [66]	4.72	12.19
	DISN [33]	3.59	8.77
	DeepMesh (raw)	2.96	9.07
	DeepMesh	2.56	8.22
NC % \uparrow	Mesh R-CNN [68]	89.09	74.82
	Pixel2Mesh [66]	89.00	72.21
	DISN [33]	91.73	78.58
	DeepMesh (raw)	92.29	78.74
	DeepMesh	92.56	80.17

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.

4.2.3 Comparative results on Pix3D.

Whereas ShapeNet contains only rendered images, Pix3D [69] 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

TABLE 4
SVR comparative results on Pix3D Chairs. Our full approach outperforms our best competitor in all metrics on real images.

Metric	DISN [33]	DeepMesh (raw)	DeepMesh
CHD $\cdot 10^3 \downarrow$	5.150	4.850	4.063 (\downarrow 16.3%)
NC % \uparrow	56.94	62.76	64.28 (\uparrow 2.4 %)

our method with our best competitor [33] according to Tab. 3. 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.

We report our results in Tab. 4 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.

4.3 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], [72], [73] 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.

4.3.1 Experimental Setup.

We started with the ShapeNet car split by automatic deletion of all the internal car parts [74] and then manually selected $N = 1400$ shapes suitable for CFD simulation. For each surface \mathcal{M}_i we ran OpenFoam [75] to predict a pressure field p_i exerted by air traveling at 15 meters per second towards the car. The resulting training set $\{\mathcal{M}_i, p_i\}_{i=1}^N$ was then used to train a Mesh Convolutional Neural Network [76] g_β to predict the pressure field $p = g_\beta(\mathcal{M})$, as in [4]. We use $\{\mathcal{M}_i\}_{i=1}^N$ to also learn the representation of Sec. 3.2 and train the network that implements f_Θ of Eq. 1. As in Section 4.2, we train both an occupancy network and signed-distance network, which we dub *DeepMesh-OCC* and *DeepMesh-SDF*, respectively.



Fig. 6. **Comparative results for SVR on Pix3D.** We compare our refined predictions to runner-up approaches for the experiment in Tab. 4. *DeepMesh* can represent arbitrary topology as well as learn strong shape priors, resulting in reconstructions that are consistent even when observed from view-points different from the input one. For more results see Appendix.

The shapes are deformed to minimize the aerodynamic objective function

$$\mathcal{L}_{\text{task5}}(\mathcal{M}) = \iint_{\mathcal{M}} g_{\beta} \mathbf{n}_x \, d\mathcal{M} + \mathcal{L}_{\text{constraint}}(\mathcal{M}) + \mathcal{L}_{\text{reg}}(\mathcal{M}), \quad (16)$$

where \mathbf{n}_x denotes the projection of surface normals along airflow

direction, the integral term approximates drag given the predicted pressure field [77], $\mathcal{L}_{\text{constraint}}$ is a loss that forces the result to preserve space the engine and the passenger compartment, and \mathcal{L}_{reg} is a regularization term that prevents \mathbf{z} from moving too far away from known shapes. $\mathcal{L}_{\text{constraint}}$ and \mathcal{L}_{reg} are described in more detail in the supplementary material. $\mathcal{L}_{\text{task5}}$ is formulated

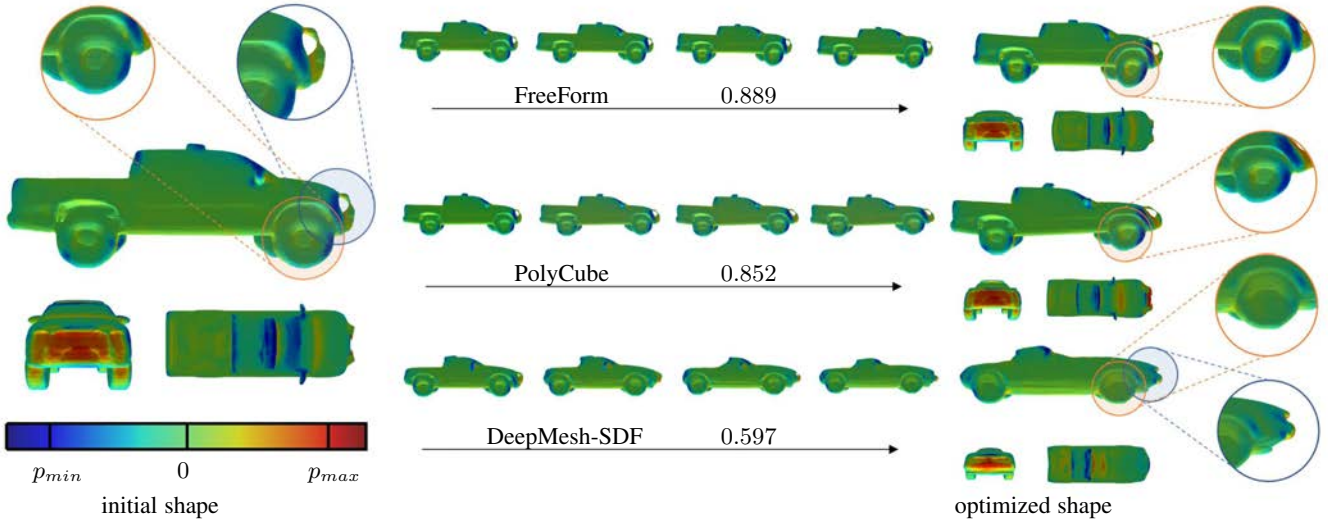


Fig. 7. **Drag minimization.** Starting from an initial shape (left column), \mathcal{L}_{task} is minimized using three different parameterizations: FreeForm (top), PolyCube (middle), and our *DeepMesh* (bottom). The middle column depicts the optimization process and the relative improvements in terms of \mathcal{L}_{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).

TABLE 5

CFD-driven optimization. We minimize drag on car shapes comparing different surface parameterizations. Numbers in the table (avg \pm std) denote relative improvement of the objective function $\mathcal{L}_{task}^{\%} = \mathcal{L}_{task} / \mathcal{L}_{task}^{t=0}$ for the optimized shape, as obtained by CFD simulation in OpenFoam.

Parameterization	None	Scaling	FreeForm [4]	PolyCube [5]	DeepMesh-SDF	DeepMesh-OCC
Degrees of Freedom	$\sim 100k$	3	21	~ 332	256	256
Simulated $\mathcal{L}_{task}^{\%} \downarrow$	not converged	0.931 ± 0.014	0.844 ± 0.171	0.841 ± 0.203	0.675 ± 0.167	0.721 ± 0.154

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.

4.3.2 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 Eq. 16 for a subset of 8 randomly chosen cars in Table 5, 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 2, *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.

4.4 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 \mathcal{M}_i , we used the finite element solver FINO [78] 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 $\{\mathcal{M}_i, s_i\}_{i=1}^N$ was then used to train a Mesh Convolutional Neural Network [76] g_{β} to predict the stress field on the surface $s = g_{\beta}(\mathcal{M})$. We also use our training set to learn the latent vector representation of Sec. 3.2 and train the network that implements f_{Θ} of Eq. 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

$$\mathcal{L}_{task6}(\mathcal{M}) = \iint_{\mathcal{M}} g_{\beta} d\mathcal{M} + \lambda \mathcal{L}_{volume}(\mathcal{M}) + \mathcal{L}_{reg}(\mathcal{M}), \quad (17)$$

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

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

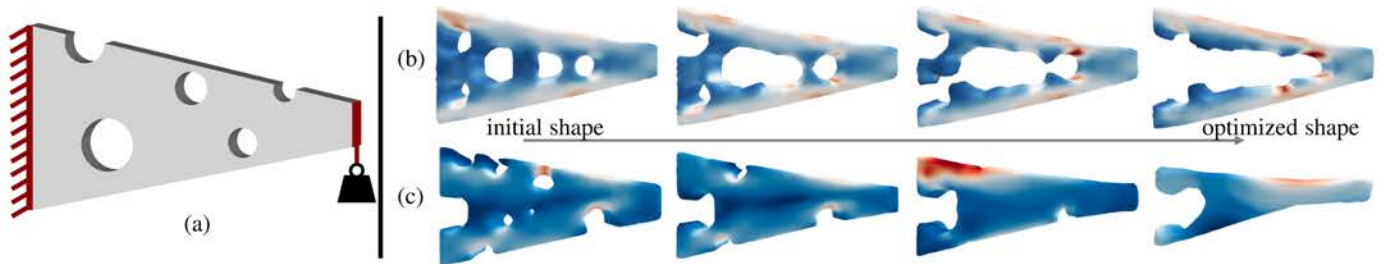


Fig. 8. **Stress minimization.** (a) Beams are attached to a wall on their left side and bear a constant load on their right side. They are all of the same length, but have variable thickness, height, and number of randomly positioned holes. (b,c) Starting from two initial beams, we use our *DeepMesh*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

TABLE 6
Refining reconstructed scenes from ConvOccNet.

	Original	Optimized
CHD	0.954	0.529
IoU	81.67 %	88.74 %

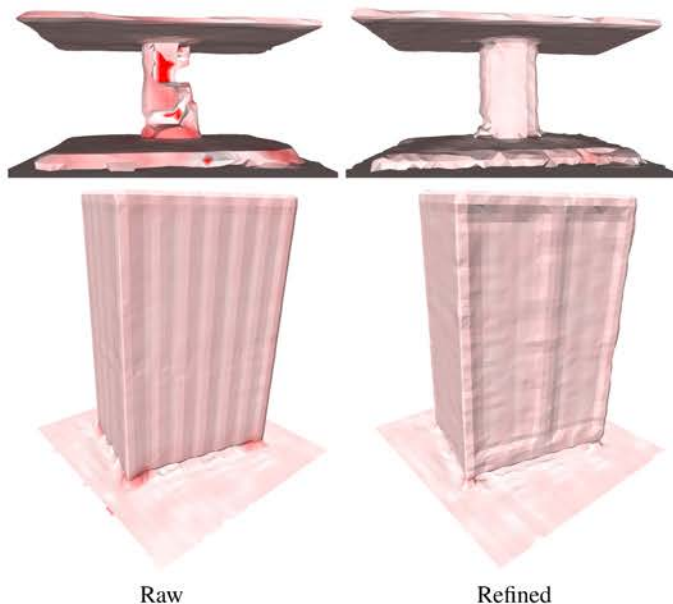


Fig. 9. **Refining CON [12] scenes.** Using our differentiable surface extraction, we can refine CON features so that the output mesh better matches the input scene point cloud. This fixes artefacts on the table, and reconstructs finer details on the wardrobe from the scene in Fig. 1(c - left). Chamfer error are shown in red, and reduced after refinement.

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

Fig. 8(b,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.

4.5 Scene Reconstruction

In the examples of Section 4.2 and 4.3, 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

TABLE 7

End-to-end training. We exploit end-to-end-differentiability to fine-tune pre-trained *DeepSDF* networks in order to minimize directly surface-to-surface (Chamfer) distance. This improves Chamfer distance (CHD) and normal consistency (NC) on the testset.

Metric	Category	<i>DeepSDF</i>	<i>DeepMesh</i>
CHD $\cdot 10^4$ (\downarrow)	<i>Chairs</i>	2.75	2.56 (\downarrow 6.9 %)
	<i>Lamps</i>	8.12	7.59 (\downarrow 6.5 %)
NC (\uparrow)	<i>Chairs</i>	80.9	81.9 (\uparrow 1.2 %)
	<i>Lamps</i>	73.9	75.1 (\uparrow 1.6 %)

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 10k 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 $G \in \mathbb{R}^{32 \times 32 \times 32 \times 32}$ 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 $M_{\mathbf{z}}$ using our framework, where $\mathbf{z} = [G|P_1|P_2|P_3]$ is the concatenation of the feature grids.

Using the differentiability of the mesh, we minimize with respect to \mathbf{z} the single-sided Chamfer distance

$$\mathcal{L}_{\text{task7}} = \sum_{p \in P} \min_{a \in M_{\mathbf{z}}} \|a - p\|^2, \quad (18)$$

between the reconstructed mesh and P , where $a \in M_{\mathbf{z}}$ represents 10k points sampled over the mesh.

In Tab. 6 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 Fig. 1(c) and Fig. 9.

4.6 End-to-End Training

In all previous examples, we considered a pre-trained network f_{Θ} 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_{Θ} and to backpropagate gradients to the network weights Θ . In other words, we consider

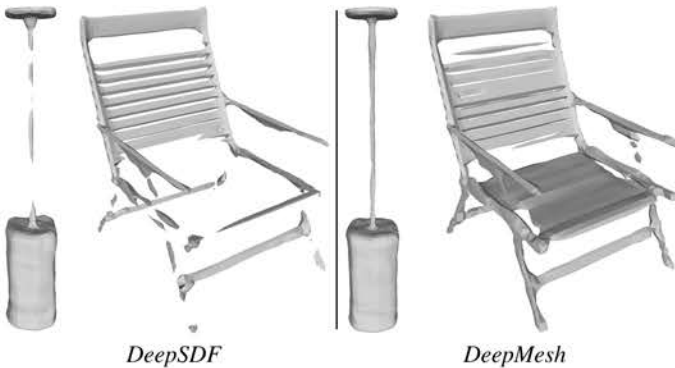


Fig. 10. **Learning network weights by minimizing the Chamfer loss.** *DeepSDF* is trained by minimizing errors on predicted SDF values, which can result in thin components being poorly reconstructed. *DeepMesh* is trained by also minimizing Chamfer distances, which penalizes such mistakes. The resulting network reconstructs thin structures better.

the setting where $\mathbf{c} = \Theta$ 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_Θ implements *DeepSDF*, as described in [1]. In the original paper, Θ along with the latent representations are learned by minimizing the implicit loss function \mathcal{L}_{imp} of Eq. 1 and its accuracy is assessed in terms of the chamfer distance between target shapes and reconstructed ones. It can be written as

$$\mathcal{L}_{\text{chamfer}} = \sum_{\mathbf{p} \in P} \min_{\mathbf{q} \in Q} \|\mathbf{p} - \mathbf{q}\|_2^2 + \sum_{\mathbf{q} \in Q} \min_{\mathbf{p} \in P} \|\mathbf{p} - \mathbf{q}\|_2^2, \quad (19)$$

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

In practice, we first minimize \mathcal{L}_{imp} to learn a first version of Θ and of the latent vectors \mathbf{z} of Eq. 1. As described in [1], this yields a network that we will refer to as *DeepSDF*. We then freeze the latent vectors and minimize $\mathcal{L}_{\text{imp}} + \mathcal{L}_{\text{chamfer}}$ with respect to Θ . 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 Sec. 4.2. 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 Tab. 7, where we report metrics on the test sets by fitting latent codes to SDF samples of unseen shapes. Minimizing $\mathcal{L}_{\text{chamfer}}$ 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 \mathcal{L}_{imp} and retain

TABLE 8

Execution speed for forward and backward passes, either directly in the implicit domain or with our method providing an explicit mesh. For using a mesh, we list runtimes of a naive and an optimized implementation of iso-surface extraction.

In the implicit domain		
Operation		Time
Forward:	<i>compute fields values for 8192 points</i>	4 ms.
Backward:	<i>from fields values to latent code</i>	8 ms.
With an explicit mesh		
Operation		Time
Forward, naive:	<i>create mesh with dense grid + CPU marching cubes</i>	785 ms.
Forward, optimized:	<i>create mesh with sparse grid + GPU marching cubes</i>	29 ms.
Backward:	<i>from mesh vertices to latent code</i>	6 ms.

this term during the fine-tuning phase. An alternative approach involves initializing the signed distance function with that of a sphere [40].

4.7 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 insufficient regularization.

We start with the network f_{Θ_1} of Sec 4.1, which was initially trained to represent a toy cow and a rubber duck, and initialize the latent code \mathbf{z} to that of the cow. We then jointly optimize the code \mathbf{z} and network weights Θ_1 to conform to a new shape, the Stanford bunny \mathcal{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

$$\mathcal{L}_{\text{task1}}(\mathcal{M}_{\Theta_1}(\mathbf{z})) = \min_{m \in \mathcal{M}} d(m, \mathcal{B}) + \min_{b \in \mathcal{B}} d(\mathcal{M}, b). \quad (20)$$

with respect to \mathbf{z} and Θ_1 with Adam [53] for 800 steps, where d is the point-to-surface distance in 3D.

Figure 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 an unseen 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 pretraining on a dataset and keeping them frozen (as in Sec. 4.1 to 4.5), 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 Sec. 4.6) ;
- adding consistency terms for the implicit field, such an eikonal regularization on the gradients (as in [79]).

4.8 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 Tab. 8, we compare forward and backward times for

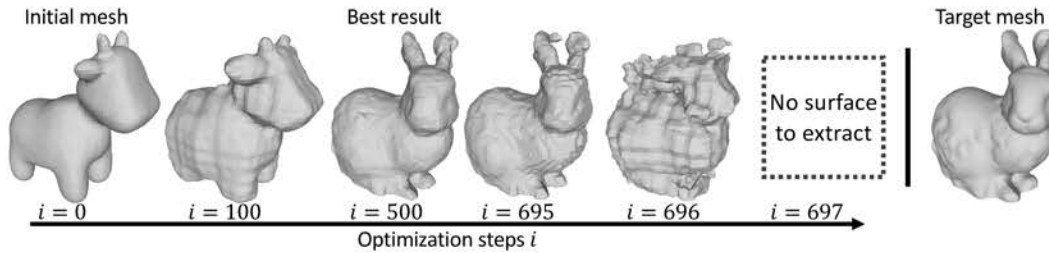


Fig. 11. **Limitation of end-to-end training:** In the absence of regularization, optimizing the network weights Θ and latent code z using our gradients for an extended number of iterations to fit an unseen shape can lead to divergence in the optimization process.

losses either on the field’s values (\mathcal{L}_{data} of Eq. 2) or through iso-surface extraction ($\mathcal{L}_{chamfer}$ of Eq. 19). The network is a DeepSDF with 8 layers of size 512, and we report average times over the testing chairs of ShapeNet. For \mathcal{L}_{data} we apply it on the default amount of 8192 points per batch. For $\mathcal{L}_{chamfer}$ we run iso-surface extraction at resolution 128^3 . The machine we run this test on uses 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 Sec. 3.2.2, 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.

5 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 [58] 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 [80] 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 [79].

ACKNOWLEDGMENTS

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

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 *Conference on Computer Vision and Pattern Recognition*, 2019.
- [2] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy Networks: Learning 3D Reconstruction in Function Space,” in *Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4460–4470.
- [3] Z. Chen and H. Zhang, “Learning Implicit Fields for Generative Shape Modeling,” in *Conference on Computer Vision and Pattern Recognition*, 2019.
- [4] P. Baqué, E. Remelli, F. Fleuret, and P. Fua, “Geodesic Convolutional Shape Optimization,” in *International Conference on Machine Learning*, 2018.
- [5] N. Umetani and B. Bickel, “Learning Three-Dimensional Flow for Interactive Aerodynamic Design,” *ACM Transactions on Graphics*, vol. 37, no. 4, p. 89, 2018.
- [6] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, “Mitsuba 2: A Retargetable Forward and Inverse Renderer,” *ACM Transactions on Graphics*, vol. 38, no. 6, pp. 1–17, 2019.
- [7] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: from Theory to Implementation*. Morgan Kaufmann, 2016.
- [8] W. Lorensen and H. Cline, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” in *ACM SIGGRAPH*, 1987, pp. 163–169.
- [9] T. Newman and H. Yi, “A Survey of the Marching Cubes Algorithm,” *Computers & Graphics*, vol. 30, no. 5, pp. 854–879, 2006.
- [10] Y. Liao, S. Donné, and A. Geiger, “Deep Marching Cubes: Learning Explicit Surface Representations,” in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2916–2925.
- [11] H. Kato, Y. Ushiku, and T. Harada, “Neural 3D Mesh Renderer,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [12] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, “Convolutional Occupancy Networks,” in *European Conference on Computer Vision*, 2020, pp. 523–540.
- [13] U. Wickramasinghe, E. Remelli, G. Knott, and P. Fua, “Voxel2mesh: 3D Mesh Model Generation from Volumetric Data,” in *Conference on Medical Image Computing and Computer Assisted Intervention*, 2020.
- [14] Y. Jiang, D. Ji, Z. Han, and M. Zwicker, “Sdfdiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization,” in *Conference on Computer Vision and Pattern Recognition*, 2020.
- [15] S. Liu, S. A. W.Chen, and H. Li, “Learning to Infer Implicit Surfaces Without 3D Supervision,” in *Advances in Neural Information Processing Systems*, 2019.
- [16] M. Michalkiewicz, J. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, “Implicit Surface Representations as Layers in Neural Networks,” in *International Conference on Computer Vision*, 2019.
- [17] S. Gould, R. Hartley, and D. Campbell, “Deep Declarative Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 3988–4004, 2021.
- [18] E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua, “MeshSdf: Differentiable Iso-Surface Extraction,” in *Advances in Neural Information Processing Systems*, 2020.
- [19] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, “Neus: Learning Neural Implicit Surfaces by Volume Rendering for Multi-View Reconstruction,” in *Advances in Neural Information Processing Systems*, 2021.
- [20] B. Mildenhall, S. P. P., M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” in *European Conference on Computer Vision*, 2020.
- [21] J. A. Sethian, *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.
- [22] S. Osher and N. Paragios, *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer, 2003.
- [23] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Generative and Discriminative Voxel Modeling with Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2016.

- [24] C. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3D-R2n2: A Unified Approach for Single and Multi-View 3D Object Reconstruction,” in *European Conference on Computer Vision*, 2016, pp. 628–644.
- [25] M. Gadelha, S. Maji, and R. Wang, “3D Shape Induction from 2D Views of Multiple Objects,” in *arXiv Preprint*, 2016.
- [26] D. Rezende, S. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, “Unsupervised Learning of 3D Structure from Images,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4996–5004.
- [27] G. Riegler, A. O. Ulusoy, and A. Geiger, “Octnet: Learning Deep 3D Representations at High Resolutions,” in *Conference on Computer Vision and Pattern Recognition*, 2017.
- [28] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs,” in *International Conference on Computer Vision*, 2017.
- [29] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3D Shapenets: A Deep Representation for Volumetric Shapes,” in *Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.
- [30] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, “Pix2vox: Context-Aware 3D Reconstruction from Single and Multi-View Images,” in *Conference on Computer Vision and Pattern Recognition*, 2019.
- [31] S. Richter and S. Roth, “Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [32] M. Tatarchenko, S. Richter, R. Ranftl, Z. Li, V. Koltun, and T. Brox, “What Do Single-View 3D Reconstruction Networks Learn?” in *Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3405–3414.
- [33] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann, “DISN: Deep Implicit Surface Network for High-Quality Single-View 3D Reconstruction,” in *Advances in Neural Information Processing Systems*, 2019.
- [34] J. Chibane, T. Alldieck, and G. Pons-Moll, “Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion,” in *Conference on Computer Vision and Pattern Recognition*, 2020.
- [35] S. Peng, W. Jiang, H. Pi, X. Li, H. Bao, and X. Zhou, “Deep Snake for Real-Time Instance Segmentation,” in *Conference on Computer Vision and Pattern Recognition*, 2020.
- [36] Y. D. Fougerolle, A. Gribok, S. Fofouf, F. Truchetet, and M. A. Abidi, “Boolean Operations with Implicit and Parametric Representation of Primitives using R-Functions,” *IEEE Transactions on Visualization and Computer Graphics*, 2005.
- [37] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, “Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision,” in *Conference on Computer Vision and Pattern Recognition*, 2020.
- [38] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, B. Ronen, and Y. Lipman, “Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance,” in *Advances in Neural Information Processing Systems*, 2020.
- [39] J. Stam and R. Schmidt, “On the Velocity of an Implicit Surface,” *ACM Transactions on Graphics*, vol. 30, no. 3, pp. 1–7, 2011.
- [40] I. Mehta, M. Chandraker, and R. Ramamoorthi, “A Level Set Theory for Neural Implicit Evolution under Explicit Flows,” in *European Conference on Computer Vision*, 2022.
- [41] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, “Efficient Implementation of Marching Cubes’ Cases with Topological Guarantees,” in *Journal of Graphics Tools*, 2003.
- [42] 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 *Conference on Computer Vision and Pattern Recognition*, 2020.
- [43] D. Vicini, S. Speierer, and W. Jakob, “Differentiable Signed Distance Function Rendering,” *ACM Transactions on Graphics*, 2022.
- [44] A. Kanazawa, S. Tulsiani, A. Efros, and J. Malik, “Learning Category-Specific Mesh Reconstruction from Image Collections,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [45] P. Henderson and V. Ferrari, “Learning Single-Image 3D Reconstruction by Generative Modelling of Shape, Pose and Shading,” *International Journal of Computer Vision*, vol. 128, no. 4, pp. 835–854, 2020.
- [46] V. Blanz and T. Vetter, “A Morphable Model for the Synthesis of 3D Faces,” in *ACM SIGGRAPH*, August 1999, pp. 187–194.
- [47] T. Bagautdinov, C. Wu, J. Saragih, P. Fua, and Y. Sheikh, “Modeling Facial Geometry Using Compositional VAEs,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [48] E. Remelli, A. Tkach, A. Tagliasacchi, and M. Pauly, “Low-Dimensionality Calibration through Local Anisotropic Scaling for Robust Hand Model Personalization,” in *International Conference on Computer Vision*, 2017.
- [49] A. Chang, T. Funkhouser, L. G., P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, “Shapenet: An Information-Rich 3D Model Repository,” in *arXiv Preprint*, 2015.
- [50] K. Park, T. Patten, and M. Vincze, “Pix2pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation,” in *International Conference on Computer Vision*, 2019.
- [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] Z. Huang, S. Bai, and J. Z. Kolter, “(Implicit)²: Implicit Layers for Implicit Representations,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 9639–9650, 2021.
- [53] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimisation,” in *International Conference on Learning Representations*, 2015.
- [54] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” in *Journal of Machine Learning Research*, 2011.
- [55] R. Venkatesh, T. Karmali, S. Sharma, A. Ghosh, R. V. Babu, L. A. Jeni, and M. Singh, “Deep Implicit Surface Point Prediction Networks,” in *International Conference on Computer Vision*, 2021.
- [56] T. Yatagawa, “mcubes_pytorch: PyTorch Implementation for Marching Cubes,” 2020. [Online]. Available: https://github.com/tatsy/mcubes_pytorch/
- [57] F. Zhao, W. Wang, S. Liao, and L. Shao, “Learning Anchored Unsigned Distance Functions with Gradient Direction Alignment for Single-View Garment Reconstruction,” in *International Conference on Computer Vision*, 2021.
- [58] B. Guillard, F. Stella, and P. Fua, “MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks,” in *European Conference on Computer Vision*, 2022.
- [59] J. Chibane, A. Mir, and G. Pons-Moll, “Neural Unsigned Distance Fields for Implicit Function Learning,” in *Advances in Neural Information Processing Systems*, 2020.
- [60] X. Long, C. Lin, L. Liu, Y. Liu, P. Wang, C. Theobalt, T. Komura, and W. Wang, “NeuralUDF: Learning Unsigned Distance Fields for Multi-view Reconstruction of Surfaces with Arbitrary Topologies,” in *Conference on Computer Vision and Pattern Recognition*, 2022.
- [61] A. Doi and A. Koide, “An Efficient Method of Triangulating Equivalued Surfaces by Using Tetrahedral Cells,” *Transactopms on Information and Systems*, vol. 74, no. 1, pp. 214–224, 1991.
- [62] T. Ju, F. Losasso, S. Schaefer, and J. Warren, “Dual Contouring of Hermite Data,” in *ACM SIGGRAPH*, 2002.
- [63] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. Devito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic Differentiation in Pytorch,” in *Advances in Neural Information Processing Systems*, 2017.
- [64] H. Fan, H. Su, and L. Guibas, “A Point Set Generation Network for 3D Object Reconstruction from a Single Image,” in *Conference on Computer Vision and Pattern Recognition*, 2017.
- [65] T. Groueix, M. Fisher, V. Kim, B. Russell, and M. Aubry, “Atlasnet: A Papier-Mâché Approach to Learning 3D Surface Generation,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [66] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y. Jiang, “Pixel2mesh: Generating 3D Mesh Models from Single RGB Images,” in *European Conference on Computer Vision*, 2018.
- [67] J. K. Pontes, C. Kong, S. Sridharan, S. Lucey, A. Eriksson, and C. Fookes, “Image2mesh: A Learning Framework for Single Image 3D Reconstruction,” in *Asian Conference on Computer Vision*, 2018.
- [68] G. Gkioxari, J. Malik, and J. Johnson, “Mesh R-CNN,” in *International Conference on Computer Vision*, 2019.
- [69] S. Xingyuan, W. Jiajun, Z. Xiuming, Z. Zhoutong, Z. Chengkai, X. Tianfan, J. Tenenbaum, and W. Freeman, “Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [70] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [71] B. Guillard, E. Remelli, P. Yvernay, and P. Fua, “Sketch2mesh: Reconstructing and Editing 3D Shapes from Sketches,” in *International Conference on Computer Vision*, 2021.
- [72] D. Toal and A. Keane, “Efficient Multipoint Aerodynamic Design Optimization via Cokriging,” *Journal of Aircraft*, vol. 48, no. 5, pp. 1685–1695, 2011.
- [73] 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, 01 2017.

- [74] F. S. Sin, D. Schroeder, and J. Barbič, “Vega: Non-Linear FEM Deformable Object Simulator,” in *Computer Graphics Forum*, 2013.
- [75] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, “OpenFOAM: A C++ Library for Complex Physics Simulations,” in *International workshop on coupled methods in numerical dynamics*, 2007.
- [76] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, “Splinecnn: Fast Geometric Deep Learning with Continuous B-Spline Kernels,” in *Conference on Computer Vision and Pattern Recognition*, 2018.
- [77] B. Munson, T. Okiishi, W. Huebsch, and A. Rothmayer, *Fluid Mechanics*. Wiley Singapore, 2013.
- [78] Seamplex, “Fino, a free finite element solver,” 2016. [Online]. Available: <https://github.com/seamplex/finno>
- [79] M. Atzmon and Y. Lipman, “SAL: Sign Agnostic Learning of Shapes from Raw Data,” in *Conference on Computer Vision and Pattern Recognition*, 2020.
- [80] S. Cheng, M. Bronstein, Y. Zhou, I. Kotsia, M. Pantic, and S. Zafeiriou, “Meshgan: Non-Linear 3D Morphable Models of Faces,” in *arXiv Preprint*, 2019.
- [81] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance Normalization: The Missing Ingredient for Fast Stylization,” in *arXiv Preprint*, 2016.
- [82] A. Mosińska, P. Marquez-Neila, M. Kozinski, and P. Fua, “Beyond the Pixel-Wise Loss for Topology-Aware Delineation,” in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3136–3145.
- [83] D. Oner, M. Koziński, L. Citraro, N. C. Dadap, A. G. Konings, and P. Fua, “Promoting Connectivity of Network-Like Structures by Enforcing Region Separation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5401–5413, 2021.
- [84] D. Oner, A. Garin, M. Koziński, K. Hess, and P. Fua, “Persistent Homology with Improved Locality Information for More Effective Delineation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 10 588–10 595, 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 *Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5425–5434.



Artem Lukoianov is a PhD student at 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 is a Product and Software Engineer at Neural Concept. He holds Master’s degrees in Theoretical Physics from ENS and Computational Mathematics from ENSTA in France. He previously was a Research Engineer at EPFL CVLab.



Stephan R. Richter leads a research team at Apple. Previously, he was a Research Scientist at the Intelligent Systems Lab at Intel. He holds B.Sc. and M.Sc. degrees in IT Systems Engineering from the Hasso-Plattner-Institute in Potsdam, Germany, and a Ph.D. degree in Computer Science from TU Darmstadt, Germany. His research interests span topics in computer vision and related areas.



Benoit Guillard is a PhD student at EPFL CVLab with Prof. Pascal Fua, working on deep learning methods to represent and manipulate 3D objects.



Timur Bagautdinov is a Research Scientist at Meta Reality Labs Research, working on photo-realistic telepresence and neural rendering. He holds a PhD from EPFL CVLab.



Edoardo Remelli is a Research Scientist at 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.



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 is a Professor of Computer Science at 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 on Pattern Analysis and Machine Intelligence*.

6 SUPPLEMENTARY MATERIAL

In this supplementary material, we first remind the reader of why Marching Cubes is not differentiable. We also provide additional details about our experiments on single view 3D reconstruction and drag minimization.

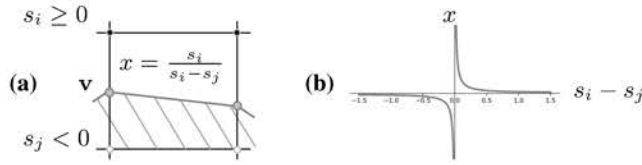


Fig. 12. **Marching Cubes differentiation.** (a) Marching Cubes determines the relative position x of a vertex \mathbf{v} along an edge via linear interpolation. This does not allow for effective back-propagation when topology changes because of a singularity when $s_i = s_j$. (b) We plot x , relative vertex position along an edge. Note the infinite discontinuity for $s_i = s_j$.

6.1 Non-differentiability of Marching Cubes

The Marching Cubes (MC) algorithm [8] extracts the zero level set of an implicit field and represents it *explicitly* as a set of triangles. As discussed in the related work section, it comprises the following steps: (1) sampling the implicit field on a discrete 3D grid, (2) detecting zero-crossing of the field along grid edges, (3) assembling surface topology, that is, the number of triangles within each cell and how they are connected, using a lookup table and (4) estimating the vertex location of each triangle by performing linear interpolation on the sampled implicit field. These steps can be understood as topology estimation followed by determination of surface geometry.

More formally, let $S = \{s_i\} \in \mathbb{R}^{N \times N \times N}$ be an implicit field sampled over a discrete Euclidean grid $G_{3D} \in \mathbb{R}^{N \times N \times N \times 3}$, where N denotes the resolution along each dimension. Within each voxel, surface topology is determined based on the sign of s_i at its 8 corners. This yields $2^8 = 256$ possible surface topologies within each voxel. Once they have been assembled into a consistent surface, vertices are created when the implicit field changes sign along one of the edges of the voxel. In such cases, the vertex location \mathbf{v} is determined by linear interpolation. Let $x \in [0, 1]$ denote the vertex relative location along an edge $(\mathbf{G}_i, \mathbf{G}_j)$, where \mathbf{G}_i and \mathbf{G}_j are grid corners such that $s_j < 0$ and $s_i \geq 0$. This implies that, if $x = 0$, then $\mathbf{v} = \mathbf{G}_i$ and conversely if $x = 1$ then $\mathbf{v} = \mathbf{G}_j$. In the MC algorithm, x is determined as the zero crossing of the interpolant of s_i, s_j , that is,

$$x = \frac{s_i}{s_i - s_j}, \quad (21)$$

as shown in Fig. 12(a). The vertex location is then taken to be

$$\mathbf{v} = \mathbf{G}_i + x(\mathbf{G}_j - \mathbf{G}_i). \quad (22)$$

Unfortunately, this function is discontinuous for $s_i = s_j$, as illustrated in Fig 12(b). Because of this, we cannot swap the signs of s_i and s_j during backpropagation. This prevents topology changes while differentiating, as discussed in [10].

6.2 Modeling non Watertight Surfaces

As explained in the main manuscript, our pipeline can also be used to represent non watertight surfaces, such as garments. We showed reconstructions of pants and t-shirts. In Fig. 13, we provide additional reconstructions.

6.3 Meshing an occupancy field

As shown in Fig. 14(a), marching cubes is not well-suited for meshing occupancy fields. Its linear interpolation step is designed to approximate vertex locations in case the sampled field is a signed distance function and does not perform well when predicted values are close to binary. Previous work applies mesh smoothing in a post-processing step to mitigate this [2]. We empirically found that amplifying the predicted occupancy values provides a simple but efficient approximation of the linear regime of a signed distance function. As shown in Fig. 14(b), applying an inverse sigmoid function within the Marching Cubes sampling loop yields smoother shapes at a negligible cost. Hence, this is what we do when meshing occupancy fields.

6.4 Comparing against Deep Marching Cubes

Deep Marching Cubes (DMC) [10] is designed to convert point clouds into a surface mesh probability distribution. It can handle topological changes but is limited to low resolution surfaces for the reasons discussed in the related work section. In Fig. 15, we compare our approach to DMC. We fit both representations to the rubber duck/cow dataset we introduced in the experiments section. We use a latent space of size 2 and report our results in terms of the CHD metric. As reported in the original paper, we found DMC to be unable to handle grids larger than 32^3 because it has to keep track of all possible mesh topologies defined within the grid. By contrast, deep implicit fields are not resolution limited. Hence, they can better capture high frequency details.

6.5 Single view 3D Reconstruction

We first provide additional details on the Single view 3D Reconstruction pipeline presented in the main manuscript. Then, for each experimental evaluation of the main paper, we first introduce metrics in details, and then provide additional qualitative results. To foster reproducibility, we have made our entire code-base publicly available.

6.5.1 Architecture

Fig 16 depicts our full pipeline. As in earlier work [2], [3], we condition our deep implicit field architecture on the input images via a residual image encoder [70], which maps input images to latent code vectors \mathbf{z} . Specifically, our encoder consists of a ResNet18 network, where we replace batch-normalization layers with instance normalization ones [81] so that to make harder for the network to use color cues to guide reconstruction. These latent codes are then used to condition the signed distance function Multi-Layer Perceptron (MLP) architecture of the main manuscript, consisting of 8 Perceptrons as well as residual connections, similarly to [1]. We train this architecture, which we dub *DeepMesh* (raw), by minimizing \mathcal{L}_{imp} (Eq.1 on the main manuscript) wrt. Θ on a training set of image-surface pairs.

At inference time, we exploit end-to-end differentiability to refine predictions as depicted in Fig 16.



Fig. 13. **Representing garments.** Our differentiable mesh parameterization can also be used to represent non-watertight surfaces. Here, we show reconstructions for the experiment presented in Section 4.1. of the main manuscript as well as ground truth open surfaces.



Fig. 14. **Marching Cubes on an occupancy field.** (a) running Marching Cubes on the occupancy field predicted by f_{Θ} yields artifacts. (b) Using an inverse sigmoid function to amplify the predicted occupancy values yields smoother shapes.

6.5.2 Evaluation on ShapeNet.

Recent work [32] has pointed out that for a typical shape in the ShapeNet test set, there is a very similar shape in the training set. To mitigate this, we carry our evaluations on new train/test splits which we design by subsampling the original datasets and rejecting shapes that have normal consistency above 98% for chairs and 96.8% for cars. Finally, we use the renderings provided in [33] for all the comparisons we report.

We use the following SVR metrics for evaluation purposes:

- **Chamfer l_2 pseudo-distance:** Common evaluation metric for measuring the distance between two uniformly sampled clouds of points P, Q , defined as

$$\text{CHD}(P, Q) = \sum_{\mathbf{p} \in P} \min_{\mathbf{q} \in Q} \|\mathbf{p} - \mathbf{q}\|_2^2 + \sum_{\mathbf{q} \in Q} \min_{\mathbf{p} \in P} \|\mathbf{p} - \mathbf{q}\|_2^2. \quad (23)$$

We evaluate this metric by sampling 2048 points from reconstructed and target shape, which are re-scaled to fit into a unit-radius sphere.

- **Normal Consistency:** We render normal maps for both the target and reconstructed shapes under 8 viewpoints, corresponding to the 8 vertices of a side 2 cube with cameras looking at its center. As depicted on Fig. 17, we get 8 pairs of normal maps that we denote $(\mathbf{n}_i, \tilde{\mathbf{n}}_i)$ with $1 \leq i \leq 8$. Our normal consistency score is the average over these 8 pairs of images of the pixelwise cosine

similarity between the reconstructed and target normal maps. With $\mathbf{n}_i \cap \tilde{\mathbf{n}}_i$ being the set of non-background pixel coordinates in both \mathbf{n}_i and $\tilde{\mathbf{n}}_i$, we have

$$\text{NC} = \frac{1}{8} \sum_{i=1}^8 \frac{1}{|\mathbf{n}_i \cap \tilde{\mathbf{n}}_i|} \sum_{(u,v) \in \mathbf{n}_i \cap \tilde{\mathbf{n}}_i} \frac{\mathbf{n}_i[u,v] \cdot \tilde{\mathbf{n}}_i[u,v]}{\|\mathbf{n}_i[u,v]\| \|\tilde{\mathbf{n}}_i[u,v]\|}. \quad (24)$$

6.5.3 Additional Qualitative Results

We provide additional qualitative comparative results for ShapeNet in Fig. 21. Fig. 22 depicts failure cases, which we take to be samples for which the refinement does not bring any improvement. These can mostly be attributed to topological errors made by the refinement process. In future work, we will therefore introduce loss functions that favor topological accuracy [82], [83], [84].

6.6 Aerodynamic Shape Optimization

Here we provide more details on how we performed the aerodynamic optimization experiments presented in the main manuscript. The overall pipeline for the optimisation process is depicted in Fig. 18, and additional optimization results are shown in Fig. 23.

6.6.1 Dataset

As described in the main manuscript, we consider the car split of the ShapeNet [49] dataset for this experiment. Since aerodynamic simulators typically require high quality surface triangulations to perform CFD simulations reliably, we (1) follow [74] and automatically remove internal part of each mesh as well as re-triangulate surfaces and (2) manually filter out corrupted surfaces. After that, we train a DeepSDF auto-decoder on the obtained data split and, using this model, we reconstruct the whole dataset from the learned parameterization. The last step is needed so that to provide fair initial conditions for each method of the comparison in Tab. 3 of the main manuscript, that is to allow all approaches to begin optimization from identical meshes.

We obtain ground truth pressure values for each car shape with OpenFoam [75], setting an *inflow velocity* of 15 meters per second and *airflow density* equal 1.18. Each simulation was run

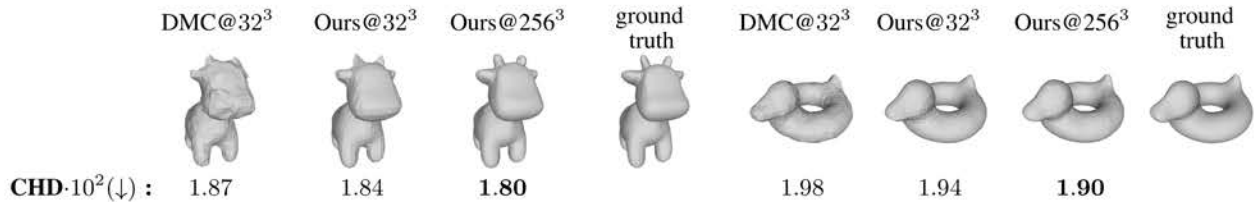


Fig. 15. **Comparison to Deep Marching Cubes** considering a latent space of size 2. Our metric is Chamfer distance, evaluated on 5000 samples for unit sphere normalized shapes.

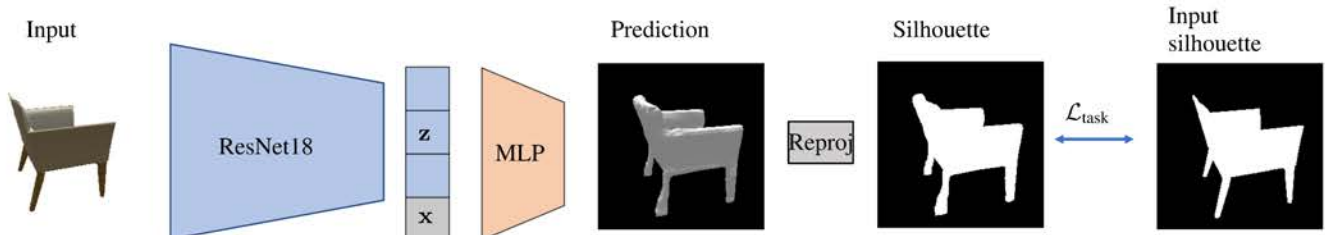


Fig. 16. **Silhouette-driven refinement.** At inference time, given an input image, we exploit the differentiability of *DeepMesh* to refine the predicted surface so that to match input silhouette in image space through Differentiable Rasterization [11] or contour matching [71].

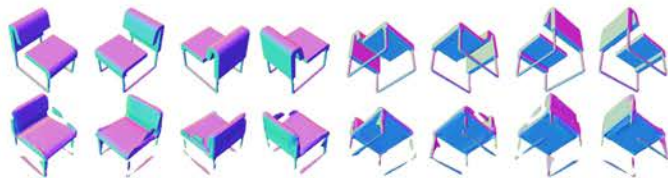


Fig. 17. **Normal consistency score:** we render target (top row) and reconstructed shapes (bottom row) normal maps from 8 fixed viewpoints, and take the average of pixelwise cosine similarity to compute an image based normal consistency score.

for at most 5000 time steps and took approximately 20 minutes to converge. Some result of the CFD simulations are depicted in the top row of Fig. 19.

We will make both the cleaned car split of ShapeNet and the simulated pressure values publicly available.

6.6.2 CFD prediction

We train a Mesh Convolutional Neural Network to regress pressure values given an input surface mesh, and then compute aerodynamic drag by integrating the regressed field. Specifically, we used the dense branch of the architecture proposed in [4] and replaced Geodesic Convolutions [85] by Spline ones [76] for efficiency. The predicted and simulated pressure values are compared in Fig. 19.

6.6.3 Implementation Details

In this section we provide the details needed to implement the baselines parameterizations presented in the main manuscript.

- **Vertex-wise optimization** In this baseline, we optimize surface geometry by flowing gradients directly into surface mesh vertices, that is without using a low-dimensional parameterization. In our experiments, we have found this strategy to produce unrealistic designs akin to adversarial attacks that, although are minimizing the drag predicted by the network, result in CFD simulations that do not

convergence. This confirms the need of using a low-dimensional parameterization to regularize optimization.

- **Scaling** We apply a function $f_{C_x, C_y, C_z}(V) = (C_x V_x, C_y V_y, C_z V_z)^T$ to each vertex of the initial shape. Here C_i are 3 parameters describing how to scale vertex coordinates along the corresponding axis. As we may see from the Tab. 3 of the main manuscript, such a simple parameterization already allows to improve our metric of interest.
- **FreeForm** Freeform deformation is a very popular class of approaches in engineering optimization. A variant of this parameterization was introduced in [4], where it led to good design performances. In our experiments we are using the parameterization described in [4] with only a small modification: to enforce the car left and right sides to be symmetrical we square sines in the corresponding terms. We also add l_2 -norm of the parameterization vector to the loss as a regularization.
- **PolyCube** Inspired by [5] we create a grid of control points to change the mesh. The grid size is $8 \times 8 \times 8$ and it is aligned to have 20% width padding along each axis. The displacement of each control point is limited to the size of each grid cell, by applying \tanh . During the optimization we shift each control point depending on the gradient it has and then tri-linearly interpolate the displacement to corresponding vertices. Finally, we enforce the displacement field to be regular by using Gaussian Smoothing ($\sigma = 1$, kernel size = 3). This results in a parameterization that allows for deformations that are very similar to the one of [5].

6.6.4 Additional Regularization for DeepMesh

As mentioned in the results section, to prevent the surface from collapsing to a point, we add the set of soft-constraints depicted by Fig. 20 to preserve space for the driver and engine and define a loss term $\mathcal{L}_{\text{constraint}}$. To avoid generating unrealistic designs, we

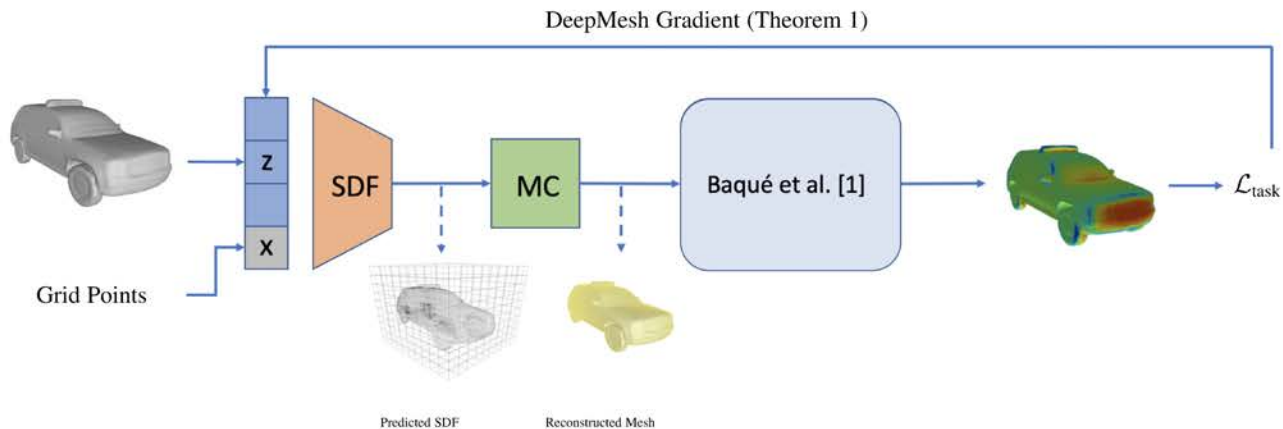


Fig. 18. **Aerodynamic optimization pipeline.** We encode a shape we want to optimize using *DeepSDF* (denoted as **SDF** block on the figure) and obtain latent code \mathbf{z} . Then we start our iterative process. First, we assemble an Euclidean grid and predict SDF values for each node of the grid. On this grid we run the Marching Cubes algorithm (**MC**) to extract a surface mesh. We then run the obtained shape through a Mesh CNN (**CFD**) to predict pressure field from which we compute drag as our objective function. Using the proposed algorithm we obtain gradients of the objective w.r.t. latent code \mathbf{z} and do an optimization step. The loop is repeated until convergence.

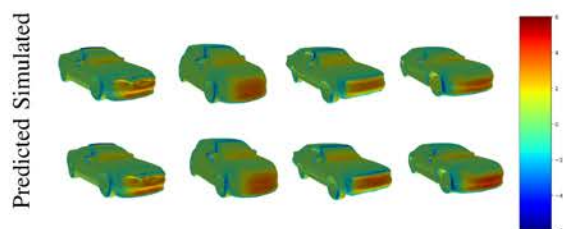


Fig. 19. **Simulated and predicted pressure fields.** Pressure fields for different shape simulated with OpenFoam (top) and predicted by Convolutional Neural Network (bottom).

also introduce an additional regularization term \mathcal{L}_{reg} that prevents \mathbf{z} from straying to far away from known designs. We take it to be

$$\mathcal{L}_{\text{reg}} = \alpha \sum_{\mathbf{z}' \in \mathcal{Z}_k} \frac{\|\mathbf{z} - \mathbf{z}'\|_2^2}{|\mathcal{Z}_k|}, \quad (25)$$

where $\mathcal{Z}_k = \mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_k$ denote the k closest latent vectors to \mathbf{z} from the training set. In our experiments we set $k = 10$, $\alpha = 0.2$. Minimizing \mathcal{L}_{reg} limits exploration of the latent space, thus guaranteeing more robust and realistic optimization outcomes.

In our aerodynamics optimization experiments, different initial shapes yield different final ones. We speculate that this behavior is due to the presence of local minima in the latent space of *DeepMesh*, even though we use the Adam optimizer [53], which is known for its ability to escape some of them. We are planning to address this problem more thoroughly in future.

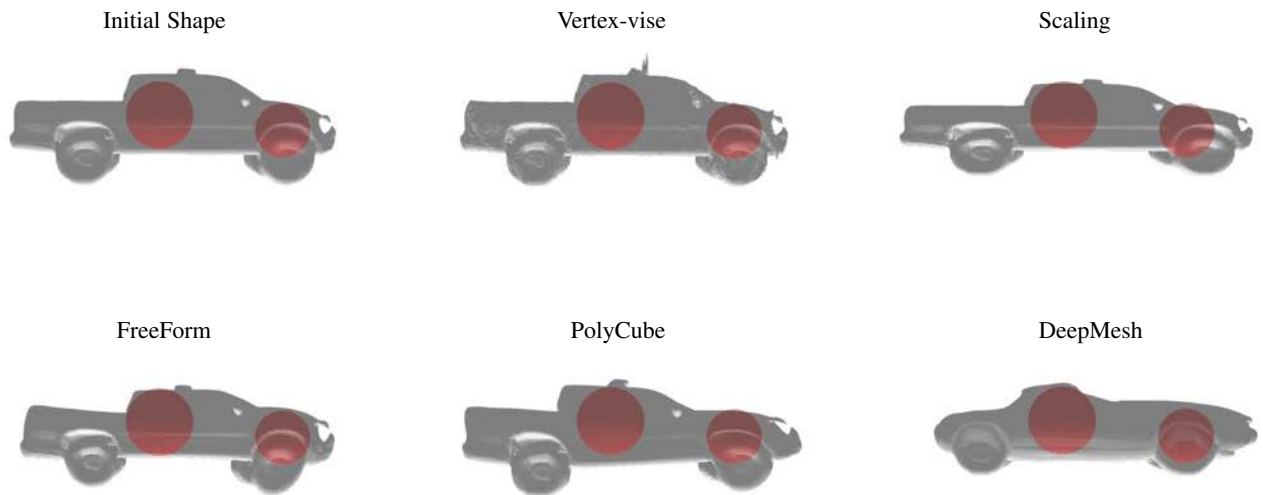


Fig. 20. **Preserving space for the driver and engine.** We define a loss function $\mathcal{L}_{\text{constraint}}$ that forces the reconstructed shape to contain the red spheres. The spheres are shown overlaid on the initial shape and then on the various results. Because the constraints are soft, they can be slightly violated.

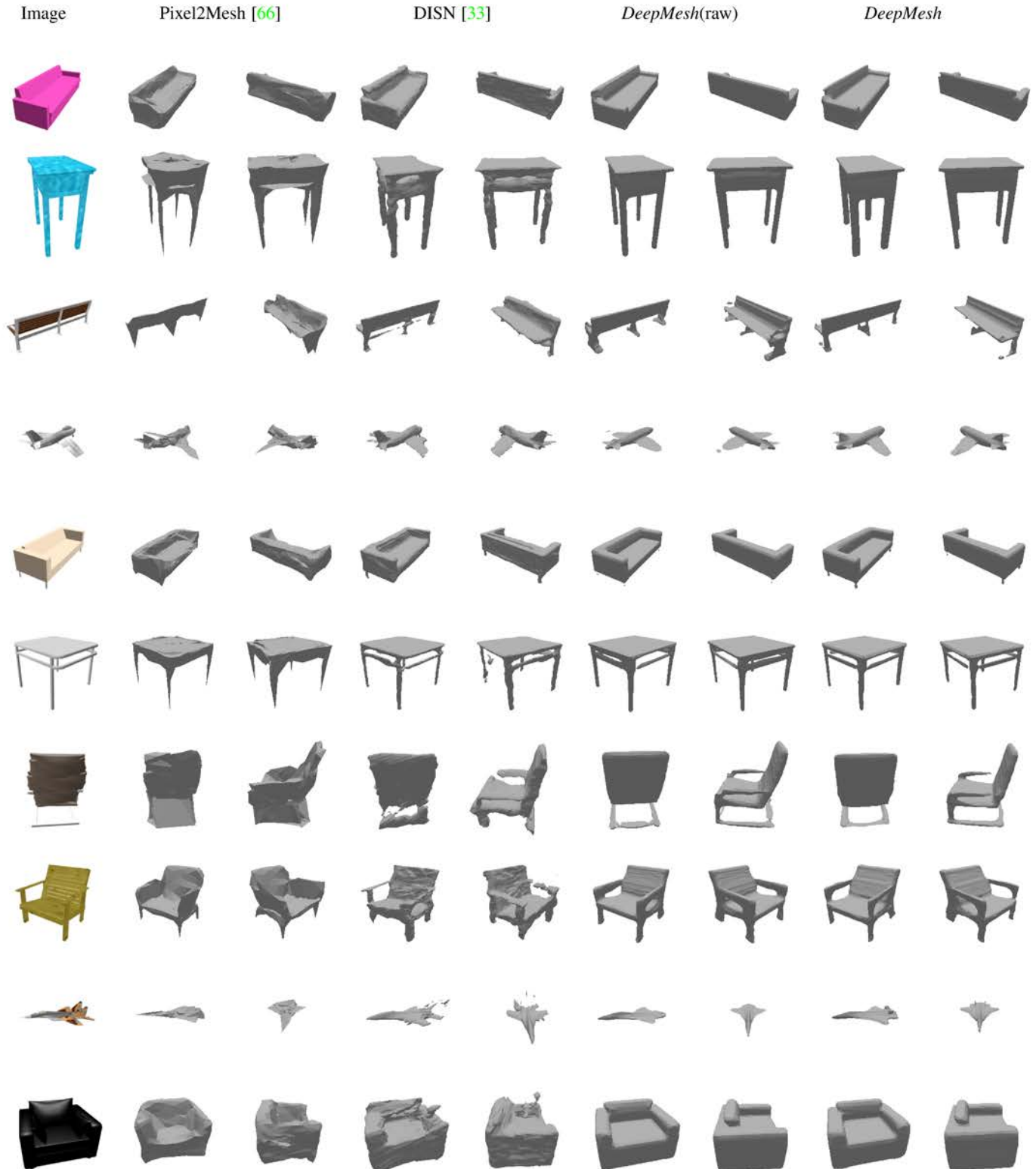


Fig. 21. Comparative results for SVR on ShapeNet.

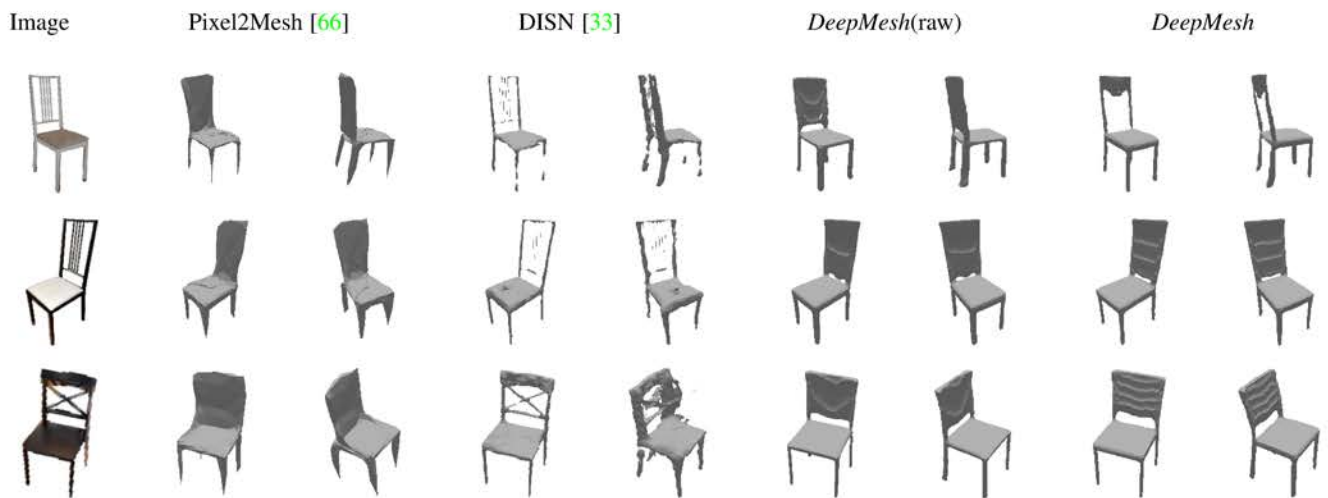


Fig. 22. **Failure cases for SVR on Pix3D.** Reconstruction refinement based on L_1 silhouette distance or chamfer matching fails to capture fine topological details for challenging samples.

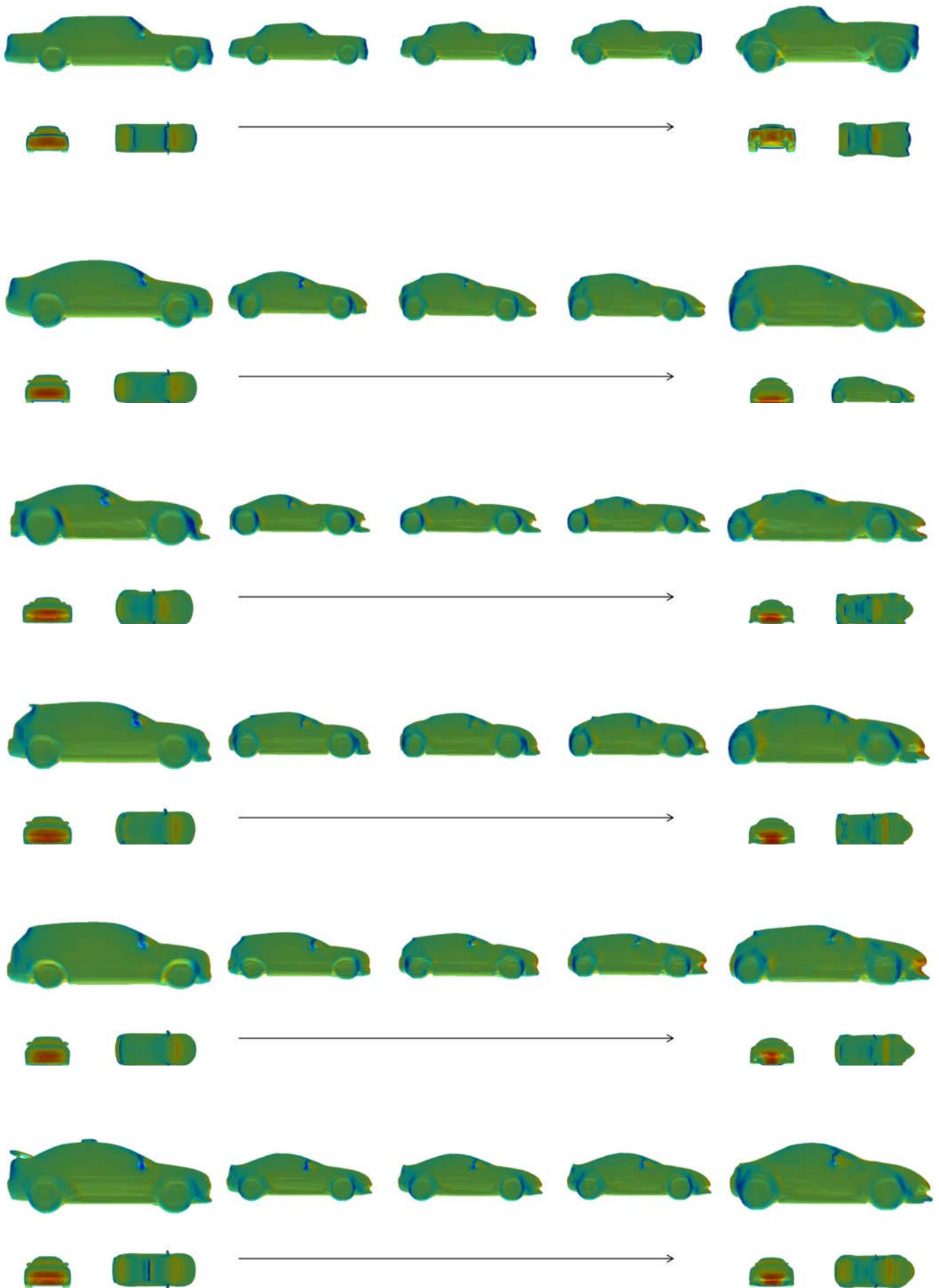


Fig. 23. DeepMesh aerodynamic optimizations.