EPFL

Thèse n° 10 386

Deep Learning for 3D Surface Modelling and Reconstruction

Présentée le 8 décembre 2023

Faculté informatique et communications
Laboratoire de vision par ordinateur
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Benoît Alain René GUILLARD

Acceptée sur proposition du jury

Dr M. Rajman, président du jury
Prof. P. Fua, directeur de thèse
Prof. P. Koehl, rapporteur
Prof. V. Lepetit, rapporteur
Prof. M. Pauly, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Acknowledgements

I express my deepest gratitude to my thesis advisor, Professor Pascal Fua, who guided me onto the path of this interesting topic, taught me how to navigate the intricacies of the academic world, and provided the perfect balance of guidance and independence throughout my PhD journey.

I would also like to extend my thanks to the members of the jury, Professor Mark Pauly, Professor Vincent Lepetit, Professor Patrice Koehl, and the committee president Dr. Martin Rajman, for dedicating their time and effort to reviewing and evaluating this thesis.

I am also indebted to other mentors at different research institutions: Sai Vemprala at Microsoft Research, Berta Bescos Torcal at Meta Reality Labs, and Marc Habermann at MPI Informatik. They taught me valuable lessons on conducting research beyond CVLab and helped navigate unfamiliar environments.

A special acknowledgement goes to Edoardo Remelli, who not only set me on the right course and initiated my progress but also became a valuable collaborator. More broadly, I would like to express my gratitude to my colleagues at CVLab for their collaboration and engaging conversations, which have contributed to the development of my research and foster a positive working environment.

My heartfelt thanks go to my amazing friends in Lausanne and beyond, in Lyon, Paris, London, and New York. Their presence has made these four years exceptionally enjoyable and has made time pass by in the blink of an eye.

Charlotte deserves my eternal gratitude for bringing joy and fun into my life and, at times, reminding me to take a break from my GPUs.

Finally, I am deeply grateful to my beloved parents for their unwavering support and invaluable advice.

Lausanne, October 4, 2023 Benoît Guillard

Abstract

In recent years, there has been a significant revolution in the field of deep learning, which has demonstrated its effectiveness in automatically capturing intricate patterns from large datasets. However, the majority of these successes in Computer Vision have been observed in the domain of 2D images. To extend these achievements to 3D applications, the development of appropriate tools and components is essential. The primary focus of this thesis is to explore the generation of 3D shapes using neural networks. To accomplish this, we propose the development of novel tools and algorithms specifically tailored for this purpose. These tools and algorithms are then applied to address concrete problems, including the reconstruction of 3D surfaces from images or sparse inputs, optimization with respect to physical quantities, and intuitive user editing of the generated shapes.

Firstly, we address the problem of reconstructing 3D shapes from 2D input images. To tackle this challenge, we propose a novel hybrid 3D shape representation that combines both voxels and 2D atlases. This representation leverages the benefits of both components: the coarse grid structure enables the principled lifting of 2D features to 3D using backprojection and 3D convolutions, which are well-suited for existing neural network architectures, while the 2D atlases provide the capability to model finer surface details. The resulting reconstruction pipeline learns a shape prior that encompasses entire object categories and achieves state-of-the-art performance on both synthetic and real images. Moreover, this approach naturally extends to the multiview scenario, allowing for robust reconstruction from multiple viewpoints.

Then, we introduce a novel approach for parameterizing watertight surfaces using deep implicit shapes. In this method, a deep neural network is employed to regress either a signed distance function or an occupancy field, which is subsequently meshed using readily available techniques. By restoring end-to-end differentiability, we demonstrate the effectiveness of this approach in generating a data-driven mesh parameterization that can dynamically modify its topology and generate smooth surfaces. Serving as a fully differentiable prior, this parameterization enables shape recovery from sparse observations using gradient descent and facilitates shape optimization based on desired physical behaviors. Additionally, we integrate this parameterization into a sketching interface, allowing for shape reconstruction and editing from simple line drawings. This intuitive user experience offers a novel approach to shape design that proves resilient to diverse sketching styles.

Finally, we extend the previous approach to handle open surfaces. By proposing an extension of a classical meshing procedure, we are able to reconstruct open surfaces using unsigned distance functions. Once again, we restore end-to-end differentiability, resulting in a robust shape parameterization. This parameterization is used for modeling garments on human bodies, and integrated into a draping pipeline that leverages the efficiency of neural networks. Thanks to its full differentiability, we can seamlessly recover and edit garments based on real observations.

Keywords: 3D deep learning, shape reconstruction, surface generation, data driven shape priors, implicit representations.

Résumé

Ces dernières années, il y a eu une révolution significative dans le domaine de l'apprentissage profond, qui a démontré son efficacité pour capturer automatiquement des motifs complexes à partir de grandes bases de données. Cependant, la majorité de ces succès en Vision par Ordinateur ont été observés dans le domaine des images 2D. Afin d'étendre ces réalisations aux applications 3D, il est essentiel de développer des outils et des composants appropriés. L'objectif principal de cette thèse est d'explorer la génération de formes 3D à l'aide de réseaux neuronaux. Nous proposons le développement de nouveaux outils et algorithmes spécifiquement adaptés à cette fin. Ces outils et algorithmes sont ensuite appliqués pour résoudre des problèmes concrets, tels que la reconstruction de surfaces 3D à partir d'images ou d'observations partielles, l'optimisation par rapport à des quantités physiques, et l'édition intuitive des formes générées par les utilisateurs.

Tout d'abord, nous abordons le problème de la reconstruction de formes 3D à partir d'images 2D. Pour relever ce défi, nous proposons une nouvelle représentation hybride de formes en 3D qui combine à la fois des voxels et des atlas 2D. Cette représentation exploite les avantages des deux composants : la structure en grille permet de transférer de manière rigoureuse les caractéristiques 2D vers la 3D en utilisant la rétroprojection et les convolutions 3D, qui sont adaptées aux architectures de réseaux neuronaux existantes, tandis que les atlas 2D permettent de modéliser les détails de surface plus fins. Le résultat est un algorithme de reconstruction qui a une connaissance de la forme d'objets courants, englobe des catégories d'objets entières et atteint des performances poussées sur des images synthétiques et réelles. De plus, cette approche s'étend naturellement au scénario multi-vues, permettant une reconstruction robuste à partir de plusieurs points de vue.

Ensuite, nous introduisons une nouvelle approche pour la paramétrisation de surfaces fermées en utilisant des formes implicites profondes. Dans cette méthode, un réseau neuronal profond est utilisé pour prédire soit une fonction de distance signée, soit un champ d'occupation, qui est ensuite maillé à l'aide de techniques déjà disponibles. En restaurant la différentiabilité de bout en bout, nous démontrons l'efficacité de cette approche pour générer une paramétrisation de maillages 3D qui est basée sur les données, capable de modifier dynamiquement sa topologie et de générer des surfaces lisses. En tant qu'a priori entièrement différentiable, cette paramétrisation permet de récupérer des formes à partir d'observations éparses en utilisant la descente de gradient, et facilite l'optimisation de formes en fonction des comportements physiques souhaités. De plus, nous intégrons cette paramétrisation dans une interface de dessin, permettant la reconstruction et la modification de formes à partir de simples coups de crayons. Cette expérience utilisateur intuitive offre une approche novatrice pour la conception de formes qui se révèle résiliente face à divers styles de dessin.

Enfin, nous étendons l'approche précédente pour prendre en compte les surfaces ouvertes. En proposant une extension d'une procédure de maillage classique, nous sommes en mesure de reconstruire des surfaces ouvertes à l'aide de fonctions de distance non signées. Une fois de plus, nous restaurons la différentiabilité de bout en bout, ce qui aboutit à une paramétrisation robuste de la forme. Cette paramétrisation est utilisée pour modéliser des vêtements sur des corps humains et intégrée dans un pipeline d'habillage qui tire parti de l'efficacité des réseaux neuronaux. Grâce à sa différentiabilité, nous pouvons obtenir et modifier facilement des vêtements à partir d'observations comme des images.

Mots-clés : apprentissage profond pour la 3D, reconstruction de formes, generation de surface, a prior de formes à partir de données, représentations implicites.

Contents

Acknowledgements i

Abstract (English/Français) iii

1 Introduction 1
- 1.1 Setting and Problem Definitions . . . 2
  - 1.1.1 Single View Reconstruction . . . 2
  - 1.1.2 Shape Parameterization . . . 3
- 1.2 Contributions . . . 4
  - 1.2.1 Hybrid Shape Decoder: Voxel Grid and 2D Atlases . . . 4
  - 1.2.2 Implicit Surfaces as Differentiable Watertight Mesh Parameterization . . 4
  - 1.2.3 Unsigned Distance Fields for Parameterizing and Meshing Open Surfaces 6
- 1.3 Outline . . . 6

2 Background 9
- 2.1 Meshes . . . 9
- 2.2 3D Voxel Grids . . . 9
- 2.3 Point Clouds . . . 10
- 2.4 Implicit Representations . . . 10
  - 2.4.1 Deep Signed Distances and Occupancies . . . 10
  - 2.4.2 Triangulating an Implicit Field . . . 11
  - 2.4.3 Differentiability . . . 11
  - 2.4.4 Open Surfaces . . . 12

3 UCLID-Net: Single View Reconstruction in Object Space 13
- 3.1 Introduction . . . 13
- 3.2 Related Work: Image Feature Extraction . . . 14
- 3.3 Method . . . 15
  - 3.3.1 Back-Projecting Feature and Depth Maps . . . 16
  - 3.3.2 Hybrid Shape Decoder . . . 17
  - 3.3.3 Implementation Details . . . 19
- 3.4 Experiments . . . 19
  - 3.4.1 Experimental Setup . . . 19
  - 3.4.2 Comparative Results . . . 20

3.4.3 From Single- to Multi-View Reconstruction 22
3.4.4 Ablation Study 22
3.5 Conclusion 23

4 DeepMesh: Differentiable Iso-Surface Extraction 25
4.1 Introduction 25
4.2 Related Work: Converting Implicit Functions to Surface Meshes 27
4.2.1 Emulating Iso-Surface Extraction 28
4.2.2 Writing Objective Functions in terms of Implicit Fields 28
4.3 Method 28
4.3.1 Deep Implicit Field Representation 29
4.3.2 Differentiable Iso-Surface Extraction 30
4.4 Experiments 34
4.4.1 Differentiable Topology Changes 35
4.4.2 Single view 3D Reconstruction 36
4.4.3 Aerodynamic Shape Optimization 41
4.4.4 Structural Shape Optimization 44
4.4.5 Scene Reconstruction 45
4.4.6 End-to-End Training 46
4.4.7 Execution Speed 48
4.5 Conclusion 48

5 Sketch2Mesh: Reconstructing and Editing 3D Shapes from Sketches 51
5.1 Introduction 51
5.2 Related Work: 3D Reconstruction from Sketches 53
5.3 Method 54
5.3.1 Formalization 54
5.3.2 Using Differential Rendering 55
5.3.3 Minimizing the 2D Chamfer Distance 55
5.3.4 Using a Partial Sketch 56
5.4 Experiments 57
5.4.1 Datasets 57
5.4.2 Metrics 59
5.4.3 Choosing the Best Method 59
5.4.4 Comparison against State-of-the-Art Methods 61
5.4.5 Interactive 3D editing 62
5.5 Conclusion 62

6 MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks 65
6.1 Introduction 65
6.2 Related Work: Triangulating Implicit Non-Watertight Surfaces 67
6.3 Method 67
6.3.1 From UDF to Triangulated Mesh 68

6.3.2 Differentiating through Iso-Surface Extraction 70
6.4 Experiments 72
6.4.1 Network and Metrics 72
6.4.2 Mesh Quality and Triangulation Speed 73
6.4.3 Using Differentiability to Fit Sparse Data 75
6.4.4 Differentiable Topology Change 77
6.4.5 Generalization to other UDF Networks 78
6.4.6 Limitations 79
6.5 Conclusion 80

7 DrapeNet: Garment Generation and Self-Supervised Draping 81
7.1 Introduction 81
7.2 Related Work: Draping Garments over 3D Bodies 83
7.3 Method 84
7.3.1 Garment Generative Network 85
7.3.2 Garment Draping Network 86
7.4 Experiments 90
7.4.1 Settings, Datasets and Metrics 90
7.4.2 Garment Paramerization 90
7.4.3 Garment Draping 91
7.4.4 Fitting Observations 95
7.5 Conclusion 97

8 Conclusions 99
8.1 Summary 99
8.2 Limitations and Future work 101

A Appendix 103
A.1 Supplementary material for UCLID-Net 103
A.1.1 Metrics 103
A.1.2 Network details 105
A.1.3 Per-category results on ShapeNet 107
A.2 Supplementary material for DeepMesh 109
A.2.1 Non-differentiability of Marching Cubes 109
A.2.2 Meshing an occupancy field 110
A.2.3 Failure Case: Vanishing Surface 110
A.2.4 Comparing against Deep Marching Cubes 111
A.2.5 Single view 3D Reconstruction 112
A.2.6 Aerodynamic Shape Optimization 114
A.3 Supplementary material for Sketch2Mesh 121
A.3.1 External Contours 121
A.3.2 Comparison of the two Refinement Approaches 122
A.4 Supplementary material for MeshUDF 126

A.4.1 Network Training 126
A.4.2 Metrics 126
A.4.3 Differentiating through Iso-Surface Extraction 128
A.4.4 Meshing approximate or real UDFs 130
A.4.5 Ablation study: pseudo-sign and breadth-first exploration 131
A.4.6 Optimization from random initial latent codes 134
A.4.7 Additional results 135
A.5 Supplementary material for DrapeNet 137
A.5.1 Network Architectures and Training 137
A.5.2 Loss Terms and Ablation Studies 139
A.5.3 Additional Results 143
A.5.4 Human Evaluation 148

Bibliography **165**

Curriculum Vitae **167**

1 Introduction

Deep learning is a subfield of machine learning that focuses on the development of algorithms inspired by the structure and function of the human brain's neural networks. It involves training artificial neural networks with multiple layers (hence the term "deep") to learn and extract meaningful representations and patterns from complex data. These algorithms leverage large amounts of data to iteratively improve their performance during a process known as *training*, where the network adjusts its internal parameters to minimize the difference between predicted outputs and expected ones. These networks excel at automatically learning hierarchical representations of data.

Deep learning has recently garnered considerable attention and achieved remarkable success in various domains, including computer vision, natural language processing, and speech recognition. Although early implementations and theories of deep neural networks have been available for decades, deep learning did not experience significant success until recent years. This success can be attributed to several key factors:

1. *The availability of large-scale datasets.* Deep learning models rely on extensive amounts of data for training, making the curation and distribution of datasets crucial.
2. *Hardware acceleration with GPUs.* The use of graphics processing units (GPUs) for parallelizing operations has greatly accelerated the computational speed of deep learning, during both training and inference stages.
3. *The existence of standard building blocks, network architectures, and best practices.* Essential components such as optimizers (e.g., Adam), layers (e.g., normalization, convolutions, attention), and training paradigms (e.g., GANs, VAEs, diffusion) were first established and shared by the research community, and have now become widely adopted and are readily available in deep learning libraries such as TensorFlow, JAX, and PyTorch.

In this thesis, we focus on and contribute to the latter point, specifically for generating 3D surfaces with neural networks, and propose new representations and algorithms for this use case. Indeed, deep learning has exhibited remarkable advancements in the generation of text (e.g., ChatGPT), sound (e.g., Magenta), and image (e.g., Dalle). These advancements have been made possible due to the availability of relevant components tailored for these modalities. However, in the case of 3D modalities, while components for *ingesting* 3D data with neural networks, such as point clouds or meshes, exist, there is no definitive solution for *generating* 3D surfaces. This is the primary focus of our thesis: exploring 3D representations and neural network building blocks specifically designed for generating 3D shape representations, also known as shape decoders. We aim to propose new representations, investigate their advantages, address the challenges they present, and demonstrate their utility in various applications.

Advancing the field of computer vision serves as a fundamental motivation for developing the ability to model 3D shapes with neural networks. Computer vision aims to comprehend, interpret, and reconstruct visual information using algorithms. By enabling machines to "see" and interpret visual data in a manner similar to human vision, we can extract 3D scenes from 2D images. As humans, we possess the innate capacity to understand the 3D geometry of our surroundings and interpret shapes, allowing us to mentally reconstruct 3D representations from a single image of a common object. Furthermore, we can visualize occluded regions and form a holistic perception with high confidence for most usual objects. Taking a broader perspective, 3D representations are essential for developing intelligent systems capable of accurately modeling and interacting with the real world. Since our physical environment is inherently 3D, we, as humans, serve as prime examples of perception and planning systems that construct mental 3D representations of our surroundings.

Enabling neural networks to generate 3D shapes serves another important objective: harnessing them as potent tools for the creation of digital 3D assets. By integrating these networks into software systems, they can offer intuitive methods for generating content across various domains, including video games, industrial prototypes, and digital characters. Deep learning presents a promising avenue for enhancing the ideation process in these fields. By providing a novel and robust approach to parameterizing 3D surfaces, neural networks can streamline complex interactive design pipelines. This enables explicit control over surface deformations (such as an elongated rear end on a car), while preserving the essential characteristics that define the objects (e.g., four wheels, mirrors...). Additionally, this approach facilitates the concurrent optimization of physical performance factors (such as ensuring aerodynamic efficiency and lightweight construction in automotive design).

1.1 Setting and Problem Definitions

1.1.1 Single View Reconstruction

Single view reconstruction (SVR), also referred to as single image 3D reconstruction or monocular 3D reconstruction, is the process of extracting 3D information about a scene or object from a single 2D image. It involves estimating the underlying 3D structure, shape, and spatial arrangement of the scene or object using solely the visual information present in the single image. The objective is to infer the geometric properties of the 3D scene, thereby creating a plausible 3D representation. Single view reconstruction goes beyond monocular depth or normal estimation by reconstructing the geometry of occluded regions as well.

The task of single view reconstruction is highly challenging due to the need to exploit various cues such as perspective, shading, texture, and object priors to estimate the 3D properties from a single 2D projection. It inherently suffers from ambiguity and thus requires to rely on learned priors for appearance, shading, scale, and shape. Consequently, it serves as an ideal use case for employing neural networks, as demonstrated in this thesis. Single View Reconstruction is also commonly used to evaluate and compare deep learning approaches to represent shapes [CXG+16b, TDB15, TDB16, TRR+19, GMJ19, XWC+19].

Analogous to human vision, our interaction with and observation of everyday objects contribute to developing an understanding of their geometry. We also generalize beyond specific object instances, and learn an understanding for common object categories as a whole. These interactions parallel the training process of artificial neural networks. Subsequently, we can naturally reconstruct a comprehensive mental 3D image of an object from a single 2D view, or at least a probable approximation.

In this thesis, we present examples of single view reconstruction using neural networks applied to RGB images of known object categories such as cars, chairs, and garments. Additionally, we explore the reconstruction from sketches as another 2D modality.

1.1.2 Shape Parameterization

To effectively represent and manipulate a prior over a shape category, it is customary to employ a shape parameterization that relies on a compact set of low-dimensional descriptors capturing the specific geometric properties and characteristics of the shape. The objective is to find expressive representations that efficiently capture the shape variations and inherent properties of the objects of interest. Shape parameterization methods can take various forms, depending on the specific application and requirements. These methods may involve representing shapes using key points [SP86, KKM10], curves [Pie91, LW94], surface patches, or other geometric primitives. The associated parameters within these representations can describe diverse aspects of the shape, including its position, orientation, scale, curvature, and other geometric properties.

In this thesis, we propose the use of neural networks for parameterizing classes of shapes that share common characteristics and geometrical attributes. A key requirement for our approach is differentiability to facilitate gradient-based optimization. Additionally, network-based parameterizations possess the advantageous property of being data-driven [BWS+18, BV99], as opposed to handcrafted [RTTP17, UB18], thereby leveraging the power of learned representations from data.

1.2 Contributions

The main objective of this thesis is to propose 3D shape representations that are compatible with deep learning, and integrate them in shape decoder networks. We explore their benefits, solve some of the challenges they pose, and demonstrate their usefulness for different applications. We introduce below our main contributions.

1.2.1 Hybrid Shape Decoder: Voxel Grid and 2D Atlases

Most deep learning-based single-view reconstruction approaches commonly employ encoder-decoder architectures, and rarely preserve the Euclidean structure of the 3D space objects exist in. In Chapter 3, we present a novel approach that addresses this limitation by constructing a geometry-preserving 3-dimensional latent space. This enables the network to simultaneously learn global shape regularities and local reasoning in the object coordinate space, resulting in enhanced performance.

Our proposed method uses a hybrid representation, consisting of a voxel grid combined with refined patch primitives known as 2D atlases within each voxel. While the voxel grid only provides a coarse representation, it nevertheless facilitates the efficient localization of predicted surfaces. This allows us to incorporate localized image and depth cues into our latent space, which significantly improves single-view reconstruction.

We demonstrate both on ShapeNet synthetic images, which are often used for benchmarking purposes, and on real-world images that our approach outperforms existing ones. Furthermore, the single-view pipeline naturally extends to multi-view reconstruction, which we also show.

This chapter is based on the conference paper [GRF20]:
B. Guillard, E. Remelli, and P. Fua, *UCLID-Net: Single View Reconstruction in Object Space*, at NeurIPS, 2020.

The corresponding source code is available at:
https://github.com/cvlab-epfl/UCLID-Net

1.2.2 Implicit Surfaces as Differentiable Watertight Mesh Parameterization

The hybrid representation introduced above is highly effective for the task of single view reconstruction. However, it generates 3D surfaces that consist of disconnected components, with one patch assigned to each surface voxel. This characteristic significantly limits the usability of these surfaces in downstream applications. To remove this limitation, we turn our attention to implicit shape representations.

In an implicit representation, surfaces are modeled as the level set of neural networks trained to regress signed distance fields (SDFs) or occupancy, as proposed in [PFS+19, MON+19]. Extracting an explicit surface from this implicit representation with Marching Cubes [LLVT03] or Dual Contouring [JLSW02] yields smooth watertight meshes. However, this meshing step disrupts end-to-end differentiability, and the resulting mesh vertices are not differentiable with respect to the neural network parameters.

In Chapter 4 we solve this by introducing gradients for mesh vertices with respect to the neural network parameters. This is achieved by examining how perturbations in the implicit field affect local surface geometry. Consequently, we develop a data-driven and differentiable mesh parameterization that is based on implicit surfaces. It can represent watertight surfaces with arbitrary topologies for a given shape category. This parameterization serves as a prior for fitting sparse observations, such as 2D silhouettes, through gradient descent, enabling the recovery of 3D shapes.

In Chapter 5 we leverage this parameterization to explore the reconstruction and edition of 3D shapes from sketches and individual pen strokes, and explain why this task poses different challenges compared to RGB images. We demonstrate how the underlying implicit and differentiable shape parameterization allows to build a simple pipeline that is robust to different sketching styles. It uses a render-and-compare approach that we propose, which considers the outer contours of sketches. Furthermore, we showcase the versatility of this pipeline for interactive shape editing using basic pen strokes, offering an intuitive approach to 3D content creation.

These chapters are based on the conference papers [RLR+20, GRYF21], and the preprint [GRL+22] which is under review at TPAMI as a journal extension of [RLR+20]:

E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua, *MeshSDF: Differentiable Iso-Surface Extraction*, at NeurIPS, 2020.

B. Guillard, E. Remelli, A. Lukoianov, P. Yvernay, S. Richter, T. Bagautdinov, P. Baque, and P. Fua, *DeepMesh: Differentiable Iso-Surface Extraction*, arXiv Preprint 2022,

B. Guillard*, E. Remelli*, P. Yvernay, and P. Fua, *Sketch2mesh: Reconstructing and Editing 3D Shapes from Sketches*, at ICCV, 2021.

The corresponding source code is available at:
https://github.com/cvlab-epfl/MeshSDF
https://github.com/cvlab-epfl/sketch2mesh

1.2.3 Unsigned Distance Fields for Parameterizing and Meshing Open Surfaces

Occupancy fields and signed distance functions are effective for modeling watertight surfaces that separate the 3D space into two regions: inside and outside. However, these methods are limited when it comes to representing open surfaces. In such cases, unsigned distance fields (UDFs) offer a suitable alternative. UDFs are characterized by being positive everywhere, providing distance information from any point in space to the surface of interest, which corresponds to the 0-level set. By leveraging UDFs, open surfaces can be effectively and accurately represented within the implicit modeling framework.

Prior research [CMPM20, ZWLS21] has proposed using neural networks to regress unsigned distance fields (UDFs), following a similar approach as with signed distance fields (SDFs). However, these approaches face challenges when it comes to efficiently and effectively meshing UDFs. In some cases, they resort to reconstructing point clouds or employ slow meshing techniques, because available meshing techniques of implicit fields involve distinguishing inner and outer regions. In Chapter 6, we address this issue by introducing an extension of the marching cubes algorithm specifically designed for meshing UDFs. Our approach involves identifying surface elements through the analysis of spatial gradient orientations. Furthermore, we ensure differentiability of the resulting mesh vertices with respect to the network parameters, similar to the watertight mesh parameterization discussed earlier. This enables the learning of a parameterization for open surface meshes such as garments, but also enhances the representation of generic meshes with inner components. Moreover, it simplifies their preprocessing since they do not need to be made watertight.

In Chapter 7, we leverage the proposed technique to model garments, and we learn to simulate their draping onto different body shapes using an additional neural network. This integration results in a garment draping pipeline that is characterized by its speed, visually appealing results, differentiability, and requires little supervision to train.

These chapters are based on the conference papers [GSF22, LLG⁺23]:

B. Guillard, F. Stella, and P. Fua, *MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks*, at ECCV, 2022.

L. De Luigi*, R. Li*, B. Guillard, M. Salzmann, and P. Fua, *DrapeNet: Generating Garments and Draping them with Self-Supervision*, at CVPR, 2023.

The corresponding source code is available at:
https://github.com/cvlab-epfl/MeshUDF
https://github.com/liren2515/DrapeNet

1.3 Outline

The remainder of the thesis is organized as follows.

In Chapter 2, we present a concise review of the prominent 3D surface representations, focusing particularly on their utilization in deep learning and shape generation. We explore how existing approaches use neural networks to generate point clouds, voxel grids, deformed template meshes, and implicit fields as means to represent 3D surfaces, and examine their individual strengths and limitations.

Then, our main technical contributions are introduced in the following chapters:

- In Chapter 3 we introduce our new neural network architecture generating a hybrid representation incorporating voxel grids and local surface patches samples as point clouds, and demonstrate its competitive performance on the single view reconstruction task.
- In Chapter 4 we turn to implicit representations of 3D shapes with neural networks, and propose a theoretically sound way to differentiate through iso-surface extraction.
- In Chapter 5 we exploit it for reconstructing and editing 3D shapes from sketches. We propose a simple render-and-compare approach that considers the outer contours of sketches, and show it is robust to different sketching styles.
- In Chapter 6 we present a novel method for meshing unsigned distance fields, thereby extending deep implicit representations to open surfaces.
- In Chapter 7 we exploit it to train a garment parameterization network in a draping pipeline that is fast, realistic, and requires little supervision to train.

Finally, in Chapter 8 we conclude by summarizing our work and its limitations, and briefly propose future research avenues.

2 Background

In this chapter, we introduce existing 3D surface representations and explain how they can be used in the context of deep learning. Of particular interest to us is the generation of 3D surfaces with neural networks, called *shape decoders*.

2.1 Meshes

Among existing 3D surface representations, meshes made of vertices and faces are one of the most popular and versatile types, and they are the de facto standard representation for most applications. Many early surface-modeling methods focused on deforming pre-existing templates based on such meshes that were either limited by design to a fixed topology [Fua96, SF09] or required *ad hoc* heuristics that do not generalize well [MT99]. Furthermore, because meshes can have variable numbers of vertices and facets, it is challenging to make this representation suitable to deep learning architectures. A standard approach has therefore been to use graph convolutions or fully connected networks to deform a predefined template [MBM+17, WZL+18]. Hence, it is limited to a fixed topology by design. Moreover, displacing vertices separately can lead to invalid meshes with self intersections, flipped faces, or rough surfaces, and regularizing the deformations is thus necessary but not trivial [WKF21, WGFB22].

A promising alternative [GFK+18] is to use a union of surface patches instead, which can handle arbitrary topologies. However, this method does not offer any guarantee that patches stitch together correctly and, in practice, yields non-watertight surfaces.

2.2 3D Voxel Grids

Voxel grids provide a straightforward extension of rasterized 2D images into the 3D domain, where each cell in a dense 3D grid represents a voxel (volume element) instead of a pixel (picture element). Many processing techniques of 2D image arrays, including deep learning methods, can be applied to voxel grids by incorporating an additional dimension. For instance, pioneering deep Shape Volume Regression (SVR) models achieved success by employing 3D convolutions to regress voxelized shapes [CXG$^+$16a]. This restricts them to coarse resolutions because of their cubic computational and memory cost. This drawback can be mitigated using local subdivision schemes [HTM17, TDB17]. MarrNet [WWX$^+$17] and Pix3D [SWZ$^+$18] regress voxelized shapes as well but also incorporate depth, normal, and silhouette predictions as intermediate representations.

To combine the strengths of voxel and mesh representations, Mesh R-CNN [GMJ19] uses a hybrid shape decoder that first regresses coarse voxels, which are then refined into mesh vertices using graph convolutions. In Chapter 3, our approach is in the same spirit with two key differences. First, our coarse occupancy grid is used to instantiate folding patches and to sample 3D surface points in the AtlasNet [GFK$^+$18] manner. However, unlike in AtlasNet, the locations of the sampled 3D points and the folding creating them are tightly coupled. Second, we regress shapes in object space, thus leveraging stronger object priors.

2.3 Point Clouds

A more lightweight representation of 3D surfaces consists in storing a set of 3D coordinates, or point clouds. From a single input image, PSGN [FSG17] regresses sparse scalar values with a neural network, directly interpreted as 3D coordinates of a point cloud with fixed size and mild continuity. AtlasNet [GFK$^+$18] and FoldingNet [YFST18] introduce a per-patch surface parameterization, which can be sampled as a point cloud from a set of learned parametric surfaces. One limitation, however, is that the patches it produces sometimes overlap each other or collapse during training [BPG$^+$19].

Despite being lightweight, point clouds lack connectivity information, and by default do not include surface normals. This makes them harder to manipulate for downstream tasks, hence many methods focus on reconstructing meshes from point clouds [PJL$^+$21, KBH06, BMR$^+$99].

2.4 Implicit Representations

2.4.1 Deep Signed Distances and Occupancies

Another alternative is to use an implicit representation, where a 3D surface is described by the zero crossings of a volumetric function $\Psi : \mathbb{R}^3 \rightarrow \mathbb{R}$ [Set99] whose values can be adjusted. The strength of this implicit representation is that the zero-crossing surface can change topology without explicit re-parameterization. Until recently, its main drawback was thought to be that working with volumes stored as dense 3D grids, instead of surfaces, massively increased the computational burden. This changed dramatically in 2019 with the introduction of continuous deep implicit-fields. They represent 3D shapes as level sets of deep fully connected networks that map 3D coordinates to a *signed distance function* [PFS$^+$19] (SDF) or an *occu-*

pancy field [MON$^+$19, CZ19]. This mapping yields a continuous shape representation that is lightweight but not limited in resolution. This representation has been successfully used for single-view reconstruction [MON$^+$19, CZ19, XWC$^+$19] and 3D shape completion [CAP20].

Signed distance and occupancy fields have case-specific benefits. For the applications we consider in this thesis, 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 [FGF$^+$05], but does not always hold for SDFs. Similarly, computing ground truth SDF values of a mesh with internal surface elements yields a false zero-level set with no change of sign, and this source of inaccuracy is removed when using occupancy.

2.4.2 Triangulating an Implicit Field

Shapes are represented *implicitly* using Signed Distance Fields (SDFs) or occupancy fields, where a specific level set of the field (e.g., 0 for SDFs and conventionally 0.5 for occupancy fields) defines the shape. To obtain an *explicit* 3D mesh representation, the field needs to be triangulated. To this effect, marching cubes was originally proposed in [LC87] and refined in [Che95, LB03, LLVT03, DK91] to triangulate one isosurface of a 3D scalar field. It marches sequentially across cubic grid cells and if field values at neighboring corners are on different sides of the target level set, triangular facets are created according to a manually defined lookup table. Vertices of these triangle facets are adjusted by linear interpolation over the field values. Since then, newer methods have been developed such as dual methods [JLSW02]. They are better at triangulating surfaces with sharp edges at the expense of increased complexity and requiring a richer input. Hence, due to its simplicity and flexibility, along with the availability of efficient implementations, the original algorithm of [LLVT03] remains in wide use [MON$^+$19, PFS$^+$19, PNM$^+$20, HAESB20, XWC$^+$19]. More recently, [CZ21, CTFZ22] proposed a data driven approach at improving sharp features reconstructed by these algorithms.

2.4.3 Differentiability

For applications requiring explicit surface parameterizations, the non-differentiability of standard approaches to iso-surface extraction [LC87] remained an obstacle to exploiting the advantages of implicit representations. [LDG18] proposed a probabilistic approach for allowing backpropagation, yet it is confined to low resolutions because of its memory requirements. In Chapter 4, we overcome this and introduce a differentiable way to produce explicit surface mesh representations from deep SDFs or occupancies. We show that, by reasoning about how implicit-field perturbations affect local surface geometry, one can differentiate the 3D location of surface samples with respect to the underlying deep implicit-field. This insight results in an end-to-end differentiable architecture that takes as input a compact latent vector and outputs a 3D watertight mesh. In other words, it enables the use of implicit fields as data-driven and differentiable mesh parameterizations. We use it in Chapter 5 and couple it with sketch-based deformations.

Other methods [NMOG20, YKM^{+}20] propose a solution to differentiate through iso-surface extraction, but they are specifically tailored to differentiable rasterization or rendering. By contrast, our method for implicit differentiation is agnostic to the downstream task. Our expression is similar to the one of [SS11], 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.

2.4.4 Open Surfaces

SDFs and occupancy field can only represent watertight surfaces. Thus, to represent open surfaces, such as clothes, it is possible to use inflated SDFs surrounding them. However, this entails a loss in accuracy and there has been a recent push to replace SDFs by *unsigned distance functions* (UDFs) [CMPM20, ZWLS21, VKS^{+}21]. One difficulty in so doing was that Marching Cubes was not designed with UDFs in mind, and obtaining explicit surfaces from these UDFs was therefore non-trivial. Other works augment signed distance fields with covariant fields to encode open surface garments [SOTC22, BRB^{+}19]. More straightforwardly, we address this in Chapter 6 by modifying the Marching Cubes algorithm to operate with UDFs. We also derive gradients for this new mesh extraction procedure, despite surface normals having ambiguous orientations.

3 UCLID-Net: Single View Reconstruction in Object Space

This chapter is based on the conference paper [GRF20]:

B. Guillard, E. Remelli, and P. Fua, *UCLID-Net: Single View Reconstruction in Object Space*, at NeurIPS, 2020.

3.1 Introduction

Most state-of-the-art deep geometric learning Single-View Reconstruction approaches (SVR) rely on encoder-decoder architectures that output either explicit shape parametrizations [GMJ19, GFK$^+$18, WZL$^+$18] or implicit representations [MON$^+$19, XWC$^+$19, CZ19]. However, the representations they learn rarely preserve the Euclidean structure of the 3D space objects exist in, and rather rely on a global vector embedding of the input image at a semantic level. In this paper, we show that building a geometry preserving 3-dimensional representation helps the network concurrently learn global shape regularities and local reasoning in the object coordinate space and, as a result, boosts performance. This corroborates the observation that choosing the right coordinate frame for the output of a deep network matters a great deal [TRR$^+$19].

In our work, we use camera projection matrices to explicitly link camera- and object-centric coordinate frames. This allows us to reason about geometry and learn object priors in a common 3D coordinate system. More specifically, we use regressed camera pose information to back-project 2D feature maps to 3D feature grids at several scales. This is achieved within our novel architecture that comprises a 2D image encoder and a 3D shape decoder. They feature symmetrical downsampling and upsampling parts and communicate through multi-scale skip connections, as in the U-Net architecture [RFB15]. However, unlike in other approaches, the bottleneck is made of 3D feature grids and we use back-projection layers [JGZ$^+$17, IBLM19, STH$^+$19] to lift 2D feature maps to 3D grids. As a result, feature localization from the input view is preserved. In other words, our feature embedding has a Euclidean structure and is aligned with object coordinate frame. Fig. 3.1 depicts this process. In reference to its characteristics, we dub our architecture UCLID-Net.

Earlier attempts at passing 2D features to a shape decoder via local feature extraction [WZL$^+$18, XWC$^+$19] enabled spatial information to flow to the decoder in a non semantic manner, often with limited impact on the final result. In these approaches, the same local feature is attributed to all points lying along a camera ray. By contrast, UCLID-Net uses 3D convolutions to volumetrically process local features before passing them to the local shape decoders. This allows them to make different contributions at different places along camera rays. To further promote geometrical reasoning, it never computes a global vector encoding of the input image. Instead, it relies on localized feature grids, either 2D in the image plane or 3D in object space. Finally, the geometric nature of the 3D feature grids enables us to exploit estimated depth maps and further boost reconstruction performance.

We demonstrate both on ShapeNet synthetic images, which are often used for benchmarking purposes, and on real-world images that our approach outperforms state-of-the-art ones. Our contribution is therefore a demonstration that creating a Euclidean preserving latent space provides a clear benefit for single-image reconstruction and a practical approach to taking advantage of it. Finally, the single-view pipeline naturally extends to multi-view reconstruction, which we also provide an example for.

3.2 Related Work: Image Feature Extraction

Most recent SVR methods rely on a 2D-CNN to create an image description that is then passed to a 3D shape decoder that generates a 3D output. What differentiates them is the nature of their output which is strongly related to the structure of their shape decoder, as already discussed. They also differ in their approach to local feature extraction, which we briefly describe below.

Most SVR methods discussed above rely on a vectorized embedding passing from image encoder to shape decoder. This embedding typically ignores image feature localization and produces a global image descriptor. As shown in [TRR$^+$19], such approaches are therefore prone to behaving like classifiers that simply retrieve shapes from a learned catalog. Hence, no true geometric reasoning occurs and recognition occurs at the scale of whole objects while ignoring fine details.

There have been several attempts at preserving feature localization from the input image by passing local vectors from 2D feature maps of the image encoder to the shape decoder. In [WZL$^+$18, GMJ19], features from the 2D plane are propagated to the mesh convolution network that operates in the camera space. In DISN [XWC$^+$19], features from the 2D plane are extracted and serve as local inputs to a SDF regressor, directly in object space. Unfortunately, features extracted in this manner do not incorporate any notion of depth and local shape regressors get the same input all along a camera ray. As a result and as shown in Fig. 3.2, DISN can reconstruct shapes with the correct outline when projected in the original viewpoint but that are nevertheless incorrect. In practice, this occurs when the network relies on both global and local features, but not when it relies on global features only. In other words, it seems

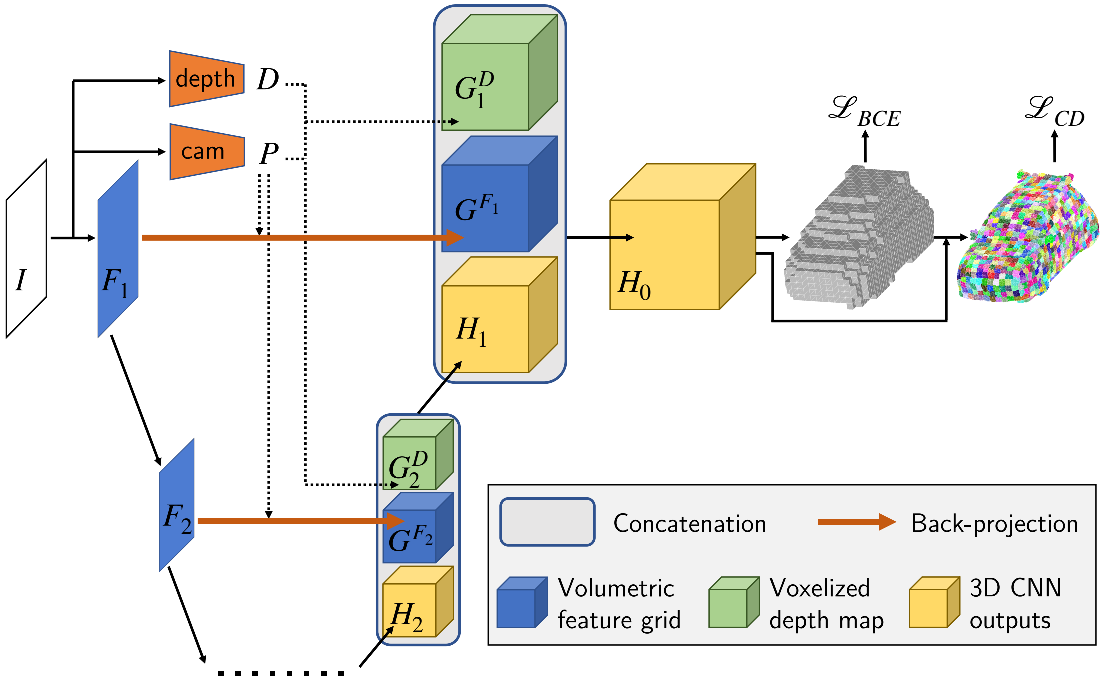


Figure 3.1 – **UCLID-Net.** Given input image I, a CNN encoder estimates 2D feature maps F_s for scales s from 1 to S while pre-trained CNNs regress a depth map D and a camera pose P. P is used to backproject the feature maps F_s to object aligned 3D feature grids G^{F_s} for $1 \leq s \leq S$ without using depth information. In parallel, S corresponding voxelized depth grids G_s^D are built from D and P without using feature information. A 3D CNN then aggregates feature and depth grids from the lowest to the highest resolution into outputs $H_S, \ldots, H_0$ of increasing resolutions. From H_0, fully connected layers regress a coarse voxel shape, which is then refined into a point cloud using local patch foldings. Supervision comes in the form of binary cross-entropy on the coarse output and Chamfer distance on the final 3D point cloud.

that local features allow the network to take an undesirable shortcut by making silhouette recovery excessively easy, especially when the background is uniform. The depth constraint is too weakly enforced by the latent space, and must be carried out by the fully connected network regressing signed distance value. By contrast, our approach avoids this pitfall, as shown in Fig. 3.2(f). This is allowed by two key differences: (i) the shape decoder relies on 3D convolutions to handle global spatial arrangement based on backprojected 2D features, before fully connected networks locally regress shape parts, and (ii) predicted depth maps are made available as inputs to the shape decoder.

3.3 Method

At the heart of UCLID-Net is a representation that preserves the Euclidean structure of the 3D world in which the shape we want to reconstruct lives. To encode the input image into it and then decode it into a 3D shape, we use the architecture depicted by Fig. 3.1. A CNN image

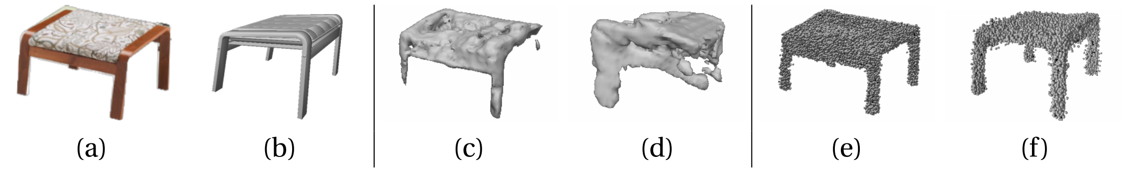


Figure 3.2 – (a) Input photograph from Pix3D [SWZ+18]. (b) Ground truth shape seen from a different viewpoint. (c,d) DISN [XWC+19] reconstruction seen from the viewpoints of (a) and (b), respectively. (e,f) Our reconstruction seen from the viewpoints of (a) and (b), respectively. For DISN, local feature extraction makes it easy to recover the silhouette in (c) but fails to deliver the required depth information. Our approach avoids this pitfall.

encoder computes feature maps at S different scales while auxiliary ones produce a depth map estimate D and a camera projection model $P : \mathbb{R}^3 \rightarrow \mathbb{R}^2$. P allows us to back-project image feature onto the 3D space along camera rays and D to localize the features at the probable location of the surface on each of these ray. The 2D feature maps and depth maps are back-projected to 3D grids that serve as input to the shape decoder, as shown by Fig. 3.3. This yields a coarse voxelized shape that is then refined into a point cloud. If estimates of either the pose P or the depth map D happen to be available *a priori*, we can use them instead of regressing them. We will show in the results section that this provides a small performance boost when they are accurate but not a very large one because our predictions tend to be good enough for our purposes, that is, lifting the features to the 3D grids.

The back-projection mechanism we use is depicted by Fig. 3.3. It is similar to the one of [JGZ+17, IBLM19, STH+19] and has a major weakness when used for single view reconstruction. All voxels along a camera ray receive the same feature information, which can result in failures such as the one depicted by Fig. 3.2 if passed as is to local shape decoders. To remedy this, we concatenate feature grids with voxelized depth maps. The result is then processed as a whole using 3D convolutions before being passed to local decoders. In the remainder of this section, we first introduce the basic back-projection mechanism, and then describe how our shape decoder fuses feature grids with depth information using a 3D CNN before locally regressing shapes.

3.3.1 Back-Projecting Feature and Depth Maps

We align all objects in the dataset to be canonically oriented within each class, centered at the origin, and scaled to fill bounding box $[-1,1]^3$. Given such a 3D object, a CNN produces a 2D feature map $F \in \mathbb{R}^{f \times H \times W}$ for input image I. Using P, the camera projection used to render it into image $I \in \mathbb{R}^{3 \times H \times W}$, we back-project F into object space as follows.

As in [JGZ+17, IBLM19], we subdivide bounding box $[-1,1]^3$ into $G^F \in \mathbb{R}^{f \times N \times N \times N}$, a regular

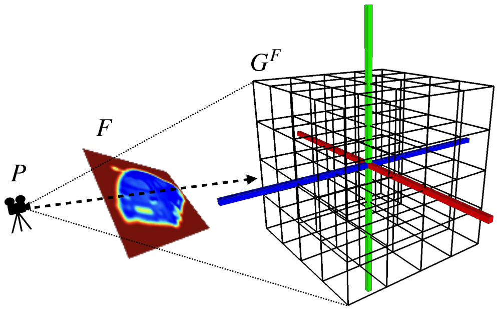


Figure 3.3 – **Backprojecting 2D features maps to 3D grids.** Rays are cast from camera P through 2D feature map F to fill 3D grid G^F. It is applied to 2D feature maps from the image encoder to provide object space aligned 3D feature grids as inputs to the shape decoder

3D grid. Each voxel (x, y, z) contains the f-dimensional feature vector

$$G^F_{xyz} = F\{P \begin{pmatrix} x \\ y \\ z \end{pmatrix}\}, \tag{3.1}$$

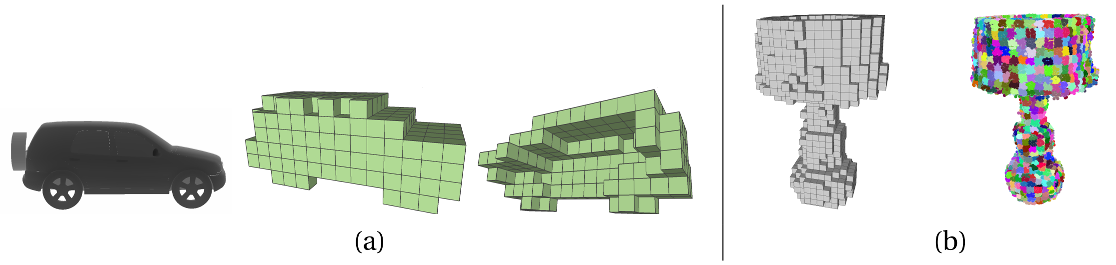
where $\{\cdot\}$ denotes bilinear interpolation on the 2D feature map. As illustrated by Fig. 3.3, back-projecting can be understood as illuminating a grid of voxels with light rays that are cast by the camera and pass through the 2D feature map. This preserves geometric structure of the surface and 2D features are positioned consistently in 3D space.

In practice, we back-project 2D feature maps $(F_1, \ldots, F_S)$ of decreasing spatial resolutions, which yield 3D feature grids $(G^{F_1}, \ldots, G^{F_S})$ of decreasing sizes $(N_1, \ldots, N_S)$. We linearly scale the projected coordinates to account for decreasing resolution.

We process depth maps in a different manner to exploit the available depth value at each pixel. Given a 2D depth map $D \in \mathbb{R}_+^{H\times W}$ of an object seen from camera with projection matrix P, we first back-project the depth map to the corresponding 3D point cloud in object space. This point cloud is used to populate binary occupancy grids such as the one depicted by Fig. 3.4(a). As for feature maps, we use this mechanism to produce a set of binary depth grids $(G_1^D, \ldots, G_S^D)$ of decreasing sizes $(N_1, \ldots, N_S)$.

3.3.2 Hybrid Shape Decoder

The feature grids discussed above contain learned features but lack an explicit notion of depth. The values in its voxels are the same along a camera ray. By contrast, the depth grids structurally carry depth information in a binary occupancy grid but without any explicit feature information. One approach to merging these two kinds of information would be to clamp projected features using depth. However, this is not optimal for two reasons. First, the depth maps can be imprecise and the decoder should learn to correct for that. Second, it can be advantageous to push feature information not only to the visible part of the surfaces


(a)

(b)

Figure 3.4 – (a) **Back-projecting depth maps.** Input depth map and back-projected depth grid seen from two different view points. (b) **Outputs of the** occ **and** $fold$ **MLPs** introduced in Section 3.3.2. One is an occupancy grid and the other a cloud of 3D points generated by individually folding patches. The points are colored according to which patch generated them.

but also to their occluded ones. Instead, we devised a shape decoder that takes as input the pairs of feature and depth grids at different scales $\{(G^{F_1}, G_1^D) ..., (G^{F_S}, G_S^D)\}$ we introduced in Section. 3.3.1 and outputs a point cloud.

Our decoder uses residual layers that rely on regular 3D convolutions and transposed ones to aggregate the input pairs in a bottom-up manner. We denote by $layer_s$ the layer at scale s, and $concat$ concatenation along the feature dimension of same size 3D grids. $layer_s$ takes as input a feature grid of size N_s and outputs a grid H_{s-1} of size N_{s-1}. If $N_{s-1} > N_s$, $layer_s$ performs upsampling, otherwise if $N_{s-1} = N_s$, the resolution remains unchanged. At the lowest scale, $layer_S$ constructs its output from feature grid G^{F_S} and depth grid G_S^D as

$$H_{S-1} = layer_S(concat(G^{F_S}, G_S^D)) \,. \tag{3.2}$$

At subsequent scales $1 \leq s < S$, the output of the previous layer is also used and we write

$$H_{s-1} = layer_s(concat(G^{F_s}, G_s^D, H_s)) \,. \tag{3.3}$$

The 3D convolutions ensure that voxels in the final feature grid H_0 can receive information emanating from different lines of sight and are therefore key to addressing the limitations of methods that only rely on local feature extraction [XWC⁺19]. H_0 is passed to two downstream Multi Layer Perceptrons (MLPs), we will refer to as occ and $fold$. occ returns a coarse surface occupancy grid. Within each voxel predicted to be occupied, $fold$ creates one local patch that refines the prediction of occ and recovers high-frequency details in the manner of AtlasNet [GFK⁺18]. Both MLPs process each voxel of H_0 independently. Fig. 3.4(b) depicts their output in a specific case. We describe them in more detail in the appendix.

Let $\tilde{O} = occ(H_0)$ be the occupancy grid generated by occ and

$$\tilde{X} = \bigcup_{\substack{xyz \\ \tilde{O}_{xyz} > \tau}} \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} + fold(u, v | (H_0)_{xyz}) \mid (u, v) \in \Lambda \right\} \tag{3.4}$$

be the union of the point clouds generated by $fold$ in each individual H_0 voxel in which the occupancy is above a threshold τ. As in [GFK$^+$18, YFST18], $fold$ continuously maps a discrete set of 2D parameters $\Lambda \subset [0,1]^2$ to 3D points in space, which makes it possible to sample it at any resolution. During the training, we minimize a weighted sum of the cross-entropy between $\widetilde{O}$ and the ground-truth surface occupancy and of the Chamfer-L_2 distance between $\widetilde{X}$ and a point cloud sampling of the ground-truth 3D model.

3.3.3 Implementation Details

In practice, our UCLID-Net architecture has $S = 4$ scales with grid sizes $N_1 = N_2 = 28$, $N_3 = 14$, $N_4 = 7$. The image encoder is a ResNet18 [HZRS16], in which we replaced the batch normalization layers by instance normalization ones [UVL16]. Feature map F_s is the output of the s-th residual layer. The shape decoder mirrors the encoder, but in the 3D domain. It uses residual blocks, with transposed convolutions to increase resolution when required. Last feature grid H_0 of the decoder has spatial resolution $N_0 = 28$, with 40 feature channels. The 8 first features serve as input to occ, and the last 32 to $fold$. occ is made of a single fully connected layer while $fold$ comprises 7 and performs two successive folds as in [YFST18]. The network is implemented in Pytorch, and trained for 150 epochs using the Adam optimizer, with initial learning rate 10^{-3}, decreased to 10^{-4} after 100 epochs.

We take the camera to be a simple pinhole one with fixed intrinsic parameters and train a CNN to regress rotation and translation from RGB images. Its architecture and training are similar to what is described in [XWC$^+$19] except we replaced its VGG-16 backbone by a ResNet18. To regress depth maps from images, we train another off-the-shelf CNN with a feature pyramid architecture [Che18]. These auxiliary networks are trained independently from the main UCLID-Net, but using the same training samples.

3.4 Experiments

3.4.1 Experimental Setup

Datasets. Given the difficulty of annotation, there are relatively few 3D datasets for geometric deep learning. We use the following two:

ShapeNet Core [CFG$^+$15] features 38000 shapes belonging 13 object categories. Within each category objects are aligned with each other and we rescale them to fit into a $[-1,1]^3$ bounding box. For training and validation purposes, we use the RGB renderings from 36 viewpoints provided in DISN [XWC$^+$19] with more variation and higher resolution than those of [CXG$^+$16a]. We use the same testing and training splits but re-generated the depth maps because the provided ones are clipped along the z-axis.

PIX3D [SWZ$^+$18] is a collection of pairs of real images of furniture with ground truth 3D models and pose annotations. With 395 3D shapes and 10,069 images, it contains far less

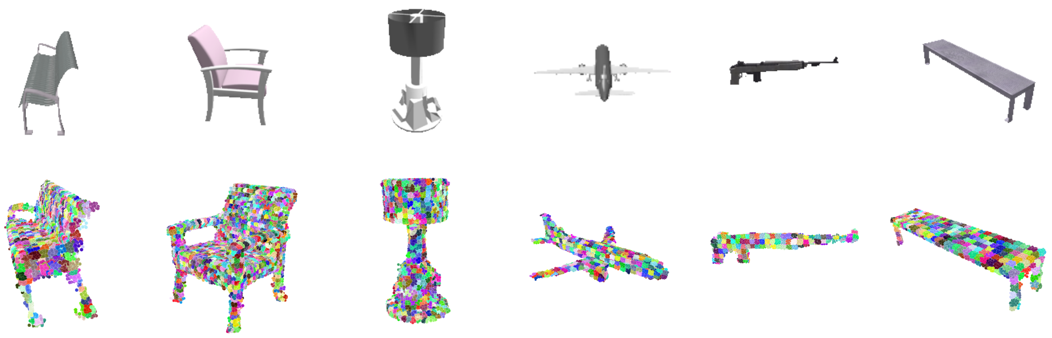


Figure 3.5 – **ShapeNet objects reconstructed by UCLID-Net.** Top row: Input view. Bottom row: Final point cloud. The points are colored according to the patch that generated them.

samples than ShapeNet. We therefore use it for validation only, on approximately 2.5k images of chairs.

Baselines and Metrics. We test our UCLID-Net against several state-of-the-art approaches: AtlasNet [GFK⁺18] provides a set of 25 patches sampled as a point cloud, Pixel2Mesh [WZL⁺18] regresses a mesh with fixed topology, Mesh R-CNN [GMJ19] a mesh with varying topological structure, and DISN [XWC⁺19] uses an implicit shape representation in the form of a signed distance function. For Pixel2Mesh, we use the improved reimplementation from [GMJ19] with a deeper backbone, which we refer to as Pixel2Mesh$^+$. All methods are retrained on the dataset described above, each according to their original training procedures.

We report our results and those of the baselines in terms of five separate metrics, Chamfer L1 and L2 Distances (CD–L_1, CD–L_2), Earth Mover's Distance (EMD), shell-IoU (sIoU), and average F-Score for a distance threshold of 5% (F@5%), which we describe in more detail in the appendix.

3.4.2 Comparative Results

ShapeNet. In Fig. 3.5, we provide qualitative UCLID-Net reconstruction results. In Tab. 3.6(a), we compare it quantitatively against our baselines. UCLID-Net outperforms all other methods. We provide the results in aggregate and refer the interested reader to the appendix for per-category results. As in [XWC⁺19], all metrics are computed on shapes scaled to fit a unit radius sphere, and CD–L_2 and EMD values are scaled by 10^3 and 10^2, respectively. Note that these results were obtained using the depth maps and camera poses regressed by our auxiliary regressors. In other words, the input was only the image. We will see in the ablation study below that they can be further improved by supplying the ground-truth depth maps, which points towards a potential for further performance gains by using a more sophisticated depth regressor than the one we currently use.

Method	CD-L_2 (↓)	EMD (↓)	sIoU (↑)	F@5% (↑)
AtlasNet	13.0	8.0	15	89.3
Pixel2Mesh$^+$	7.0	3.8	30	95.0
Mesh R-CNN	9.0	4.7	24	92.5
DISN	9.7	2.6	30	90.7
Ours	**6.3**	**2.5**	**37**	**96.2**

(ShapeNet)

Method	CD-L_1 (↓)	EMD (↓)
Pix3D	11.9	11.8
AtlasNet	12.5	12.8
Pixel2Mesh$^+$	10.0	12.3
Mesh R-CNN	10.8	13.7
DISN	10.4	11.7
Ours	**7.5**	**8.7**

(Pix3D)

Figure 3.6 – **Comparative results.** For ShapeNet, we re-train and re-evaluate all methods. For Pix3D, lines 1-2 are duplicated from [SWZ$^+$18], while lines 3-6 depict our own evaluation using the same protocol. The up and down arrows next to the metric indicate whether a higher or lower value is better.

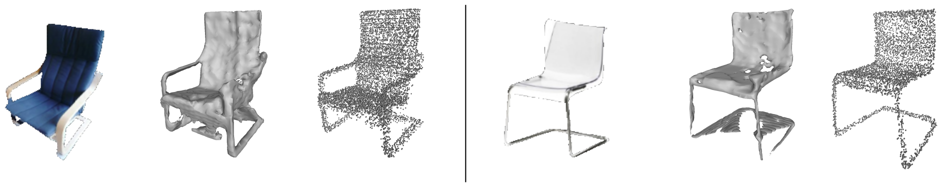


Figure 3.7 – **Reconstructions on Pix3D photographs:** from left to right, twice: input, DISN, ours.

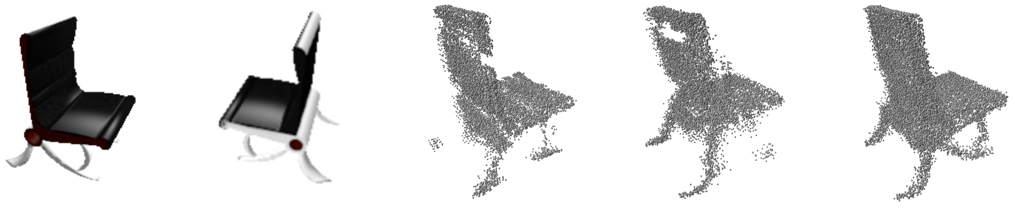

(a) (b) (c) (d) (e)

Figure 3.8 – **Two-views reconstruction.** (a,b) Two input images of the same chair from ShapeNet. (c) Reconstruction using only the first one. (d) Reconstruction using only the second one. (e) Improved reconstruction using both images.

Pix3D. In Fig. 3.7, we provide qualitative UCLID-Net reconstruction results. In Tab. 3.6(b), we compare it quantitatively against our baselines. We conform to the evaluation protocol of [SWZ$^+$18] and report the Chamfer-L1 distance (CD–L_1) and EMD on point clouds of size 1024. The CD–L_1 and EMD values are scaled by 10^2. UCLID-Net again outperforms all other methods. The only difference with the ShapeNet case is that both DISN and UCLID-Net used the available camera models whereas none of the other methods leverages camera information.

3.4.3 From Single- to Multi-View Reconstruction

A further strength of UCLID-Net is that its internal feature representations make it suitable for multi-view reconstruction. Given depth and feature grids provided by the image encoder from multiple views of the same object, their simple point-wise addition at each scale enables us to combine them in a spatially relevant manner. For input views a and b, the encoder produces feature/depth grids collections $\{(G_a^{F_1}, G_{1,a}^D)\ldots,(G_a^{F_S}, G_{S,a}^D)\}$ and $\{(G_b^{F_1}, G_{1,b}^D)\ldots,(G_b^{F_S}, G_{S,b}^D)\}$. In this setting, we feed $\{(G_a^{F_1}+G_b^{F_1}, G_{1,a}^D+G_{1,b}^D)\ldots,(G_a^{F_S}+G_b^{F_S}, G_{S,a}^D+G_{S,b}^D)\}$ to the shape decoder and let it merge details from both views. For best results, the decoder is fine-tuned to account for the change in magnitude of its inputs. As can be seen in Fig. 3.8, this delivers better reconstructions than those obtained from each view independently.

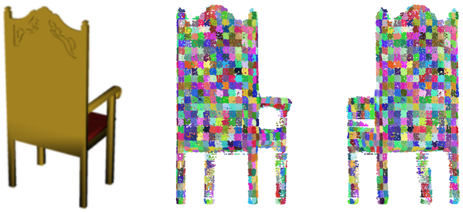
3.4.4 Ablation Study

To quantify the impact of regressing camera poses and depth maps, we conducted an ablation study on the ShapeNet car category. In Fig. 3.9(a), we report CD-L_2 and EMD for different network configurations. Here, *CAR* is trained and evaluated on the cars subset, with inferred depth maps and camera poses. *CAM* is trained and evaluated with inferred depth maps, but ground truth camera poses. *CAD* is trained and evaluated with ground truth camera poses and depth maps. Finally, *ALL* is trained on 13 object categories with inferred depth maps and cameras as it was in all the experiments above, but evaluated on cars only.

Using ground truth data annotation for depth and pose improves reconstruction quality. The

Method	CD-L_2 (↓)	EMD (↓)
CAR	4.08	2.23
CAM	3.83	2.16
CAD	3.80	2.14
ALL	4.03	2.23

(a)



(b)

Figure 3.9 – (a) **Ablation study:** comparative results on a single object category. (b) **Failure mode.** From left-to-right: input view, reconstruction seen from the back-right, seen from the back-left. The visible armrest is correctly carved. The other one (occluded in the input) is mistakenly reconstructed as solid.

margin is not significant, which indicates that the regressed poses and depth maps are mostly good enough. Nevertheless, our pipeline is versatile enough to take advantage of additional information, such as depth map from a laser scanner or an accurate camera model obtained using classic photogrammetry techniques, when it is available. Note also that *ALL* marginally gets better performance than *CAR*. Training the network on multiple classes does not degrade performance when evaluated on a single class. In fact, having other categories in the training set increases the overall data volume, which seems to be beneficial.

In Fig. 3.9(b), we present an interesting failure case. The visible armrest is correctly carved out while the occluded one is reconstructed as being solid. While incorrect, this result indicates that UCLID-Net has the ability to reason locally and does not simply retrieve a shape from the training database, as described in [TRR$^+$19].

3.5 Conclusion

We have shown that building intermediate representations that preserve the Euclidean structure of the 3D objects we try to model is beneficial. It enables us to outperform state-of-the-art approaches to single view reconstruction. We have also investigated the use of multiple-views for which our representations are also well suited. In future work, we will extend our approach to handle video sequences for which camera poses can be regressed using either SLAM-type methods or learning-based ones. We expect that the benefits we have observed in the single-view case will carry over and allow full scene reconstruction.

4 DeepMesh: Differentiable Iso-Surface Extraction

This chapter is based on the preprint [GRL$^+$22]:

B. Guillard, E. Remelli, A. Lukoianov, P. Yvernay, S. Richter, T. Bagautdinov, P. Baque, and P. Fua, *Deepmesh: Differentiable Iso-Surface Extraction*, arXiv Preprint 2022,
which is under review at TPAMI as a journal extension of the conference paper [RLR$^+$20]:

E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua, *Meshsdf: Differentiable Iso-Surface Extraction*, at NeurIPS, 2020.

4.1 Introduction

Geometric Deep Learning has recently witnessed a breakthrough with the advent of Deep Implicit Fields (DIFs) [PFS$^+$19, MON$^+$19, CZ19]. 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 [BRFF18, UB18] 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 [NDVZJ19, PJH16].

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 [LC87, NY06]. However, these approaches are not fully differentiable [LDG18]. 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 [LDG18, WRKF20]. These techniques, however, remain limited to low-

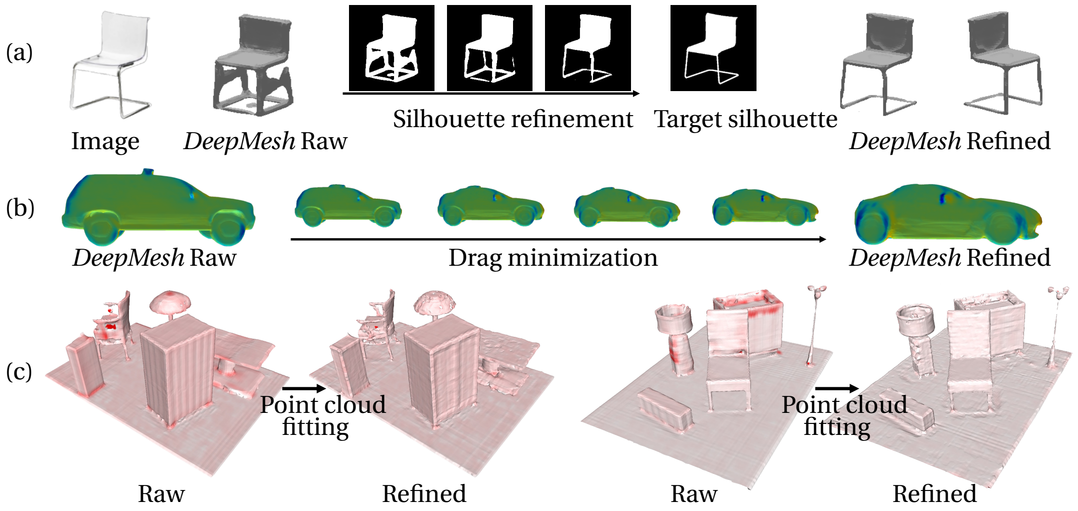


Figure 4.1 – **DeepMesh**. (a) We condition our representation on an input image and output an initial 3D mesh, which we refine via differentiable rasterization [KUH18], thereby exploiting DeepMesh's end-to-end differentiability. (b) We use our parameterization as a powerful regularizer for aerodynamic 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*) [PNM$^+$20]. In these two examples, the raw output of [PNM$^+$20] is shown on the left and the refined version on the right. The errors are shown in red and are smaller after refinement.

resolution meshes [LDG18] or fixed topologies [WRKF20]. An alternative approach is to reformulate downstream tasks, such as differentiable rendering [JJHZ20, LWL19] or surface reconstruction [MPJ$^+$19], 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. 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 isosurface. 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 [RLR$^+$20] that focused on the 0-isosurface of signed distance functions. We extend it here to isosurface 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. 4.1(a).
2. Similarly, we use the surface triangulations to compute the aerodynamic properties of 3D shapes and refine them, as shown in Fig. 4.1(b).
3. We use our paradigm in conjunction with DIF-based methods to improve their performance in a plug-and-play fashion by adding loss terms that can be computed on the meshes. This highlights the importance to be able to handle both SDFs and occupancy grids.
4. We demonstrate that we can use our approach not only to better exploit the results of pre-trained networks but to actually train them better.

In all these cases, our end-to-end differentiable parameterization gives us an edge over state-of-the art algorithms. Note, however, that our approach relies on latent variable models to capture priors applicable to entire object categories, unlike some of the recent multi-view approaches [WLL$^+$21, MPT$^+$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.

4.2 Related Work: Converting Implicit Functions to Surface Meshes

The Marching Cube (MC) algorithm [LC87, NY06] 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 [LDG18], as illustrated in Fig. 4.2(a). Because this is a central motivation for this work, we provide a more detailed analysis of this shortcoming in the appendix. 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.

4.2.1 Emulating Iso-Surface Extraction

In [LDG18] 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 [WRKF20] deforms a mesh primitive and adaptively increases its resolution. While this makes it possible to represent high resolution meshes, it prevents changes of topology.

4.2.2 Writing Objective Functions in terms of Implicit Fields

In [MPJ$^+$19], variational analysis is used to re-formulate standard surface mesh priors, such as those that enforce smoothness, in terms of implicit fields. Although elegant, this technique requires carrying out complex derivations for each new loss function and can only operate on an Euclidean grid of fixed resolution. The differentiable renderers of [JJHZ20] and [LZP$^+$20] rely on sphere tracing and operate directly in terms of implicit fields. Unfortunately, since it is computationally intractable to densely sample the underlying volume, these approaches either define implicit fields over a low-resolution Euclidean grid [JJHZ20] or rely on heuristics to accelerate ray-tracing [LZP$^+$20], while reducing accuracy. 3D volume sampling efficiency can be improved by introducing a sparse set of anchor points when performing ray-tracing [LWL19]. However, this requires reformulating standard surface mesh regularizers in terms of implicit fields using computationally intensive finite differences. Furthermore, these approaches are tailored to differentiable rendering, and are not directly applicable to different settings that require explicit surface modeling, such as computational fluid dynamics. This also applies to [NMOG20, YKM$^+$20] 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.

4.3 Method

Tasks such as Single view 3D Reconstruction (SVR) [KTEM18, HF20] or shape design in the context of CFD [BRFF18] are commonly performed by deforming the shape of a 3D surface mesh $\mathscr{M} = (V, F)$, where $V = \{\mathbf{v}_1, \mathbf{v}_2, ...\}$ denotes vertex positions in $\mathbb{R}^3$ and F facets, to minimize a task-specific loss function $\mathscr{L}_{\text{task}}(\mathscr{M})$. $\mathscr{L}_{\text{task}}$ 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 [BV99, PFS$^+$19, BWS$^+$18] or hand-crafted [BRFF18, UB18, RTTP17]. 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

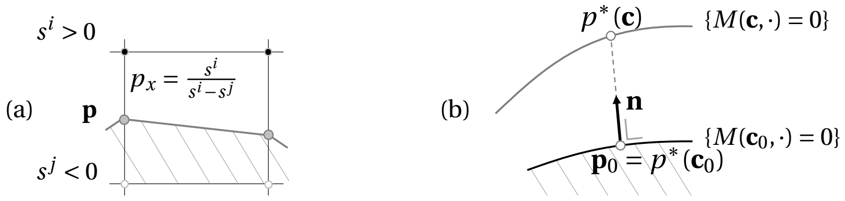

$s^i > 0$
$p_x = \frac{s^i}{s^i - s^j}$
(a)
p
$s^j < 0$
$p^*(\mathbf{c})$
$\{M(\mathbf{c}, \cdot) = 0\}$
(b)
n
$\{M(\mathbf{c}_0, \cdot) = 0\}$
$\mathbf{p}_0 = p^*(\mathbf{c}_0)$

Figure 4.2 – **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 [LDG18]. (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 $\mathbf{c}_0 \hookrightarrow \mathbf{c}$ and local surface change $\mathbf{p}_0 \hookrightarrow p^*(\mathbf{c})$, which we exploit to compute $\frac{\partial p^*(\mathbf{c})}{\partial \mathbf{c}}$ even when the topology changes.

alternative but conversely require a *differentiable* conversion to explicit representations in order to backpropagate gradients of $\mathscr{L}_{\text{task}}$.

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

4.3.1 Deep Implicit Field Representation

In this work, we represent a generic watertight surface S implicitly by a function $s : \mathbb{R}^3 \to \mathbb{R}$. Typical choices for s include the Signed Distance Function (SDF) where $s(\mathbf{x})$ 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 $\mathscr{D}$, such as ShapeNet [CFG+15], we train a Multi-Layer Perceptron (MLP) f_Θ as in [PPV19] to approximate s over such set of surfaces $\mathscr{D}$ by minimizing

$$\mathscr{L}_{\text{imp}}(\{\mathbf{z}_S\}_{S \in \mathscr{D}}, \Theta) = \mathscr{L}_{\text{data}}(\{\mathbf{z}_S\}_{S \in \mathscr{D}}, \Theta) + \lambda_{\text{reg}} \sum_{S \in \mathscr{D}} \|\mathbf{z}_S\|_2^2 \,, \tag{4.1}$$

where $\mathbf{z}_S \in \mathbb{R}^Z$ is the Z-dimensional encoding of surface S, Θ denotes network parameters, L_{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 $\mathscr{L}_{\text{data}}$ to be the L_1 loss

$$\mathscr{L}_{\text{data}} = \sum_{S \in \mathscr{D}} \frac{1}{|X_S|} \sum_{\mathbf{x} \in X_S} |f_\Theta(\mathbf{z}_S, \mathbf{x}) - s(\mathbf{x})| \,, \tag{4.2}$$

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

$$\begin{aligned}\mathscr{L}_{\text{data}} = -\sum_{S \in \mathscr{D}} \frac{1}{|X_S|} \sum_{\mathbf{x} \in X_S} & s(\mathbf{x}) \log(f_\Theta(\mathbf{z}_S, \mathbf{x})) \\ & + (1 - s(\mathbf{x})) \log(1 - f_\Theta(\mathbf{z}_S, \mathbf{x})) \,. \end{aligned} \tag{4.3}$$

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.

4.3.2 Differentiable Iso-Surface Extraction

Once the weights Θ of Eq. 4.1 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 $\mathscr{L}_{\text{task}}$ 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. 4.2(a). More formally, we need to evaluate

$$\frac{\partial \mathscr{L}_{\text{task}}}{\partial \mathbf{c}} = \sum_{\mathbf{x} \in V} \frac{\partial \mathscr{L}_{\text{task}}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{c}} \,, \tag{4.4}$$

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 $\mathscr{L}_{\text{task}}$ 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 $\partial \mathscr{L}_{\text{task}} / \partial \mathbf{c}$ by summing over the mesh vertices but we could use any other sampling of the surface.

Differentiating the Loss Function

In this work, we take inspiration from classical functional analysis [AJT02] 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. To this end, we prove below that

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. 4.4 yields

$$\frac{\partial \mathscr{L}_{\text{task}}}{\partial \mathbf{c}} = -\sum_{\mathbf{x}\in V} \frac{\partial \mathscr{L}_{\text{task}}}{\partial \mathbf{x}} \frac{\nabla f_\Theta}{\|\nabla f_\Theta\|^2} \frac{\partial f_\Theta}{\partial \mathbf{c}} \,. \tag{4.5}$$

Note that when s is an SDF, $\|\nabla s\| = 1$ and therefore $\|\nabla f_\Theta\| \approx 1$. The $\|\nabla f_\Theta\|$ term from Eq. 4.5 can then be ignored, which is consistent with the result we presented in [RLR⁺20].

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

Theorem 2 (Implicit Function Theorem - IFT) *Let* $F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ *and* $\mathbf{c}_0 \in \mathbb{R}^m, \mathbf{p}_0 \in \mathbb{R}^n$ *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* $p^* : \mathbb{R}^m \to \mathbb{R}^n$ *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. $\partial p^*(\mathbf{c}_0) = -\left[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)\right]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0)$, *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 $M : \mathbb{R}^m \times \mathbb{R}^3 \to \mathbb{R}$ 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 $F : \mathbb{R}^m \times \mathbb{R}^3 \to \mathbb{R}^3$ as

$$F(\mathbf{c}, \mathbf{p}) \mapsto \begin{pmatrix} M(\mathbf{c},\mathbf{p}) \\ (\mathbf{p}-\mathbf{p}_0)\cdot\mathbf{u} \\ (\mathbf{p}-\mathbf{p}_0)\cdot\mathbf{v} \end{pmatrix}, \tag{4.6}$$

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. 4.2(b).

Given the IFT applied to F, there is a mapping $p^* : \mathbb{R}^m \to \mathbb{R}^3$ 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. $\partial p^*(\mathbf{c}_0) = -\left[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)\right]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0)$.

We have

$$\partial_c F(\mathbf{c}_0, \mathbf{p}_0) = \begin{pmatrix} \partial_c M(\mathbf{c}_0,\mathbf{p}_0) \\ 0 \\ 0 \end{pmatrix} \in \mathbb{R}^{3\times m}, \tag{4.7}$$

$$\partial_p F(\mathbf{c}_0, \mathbf{p}_0) = \begin{pmatrix} \mathbf{n} \\ \mathbf{u} \\ \mathbf{v} \end{pmatrix} \in \mathbb{R}^{3\times 3} \tag{4.8}$$

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 $\left[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)\right]^{-1}$. 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

$$\begin{aligned} \partial p^*(\mathbf{c}_0) &= -\left[\partial_p F(\mathbf{c}_0, \mathbf{p}_0)\right]^{-1} \partial_c F(\mathbf{c}_0, \mathbf{p}_0), \qquad (4.9) \\ &= -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \partial_c F(\mathbf{c}_0, \mathbf{p}_0) \in \mathbb{R}^{3\times m}. \end{aligned}$$

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

$$\frac{\partial \mathbf{p}_0}{\partial \mathbf{c}} = -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \partial_c F(\mathbf{c}_0, \mathbf{p}_0), \tag{4.10}$$

$$= -\frac{\mathbf{n}}{\|\mathbf{n}\|^2} \frac{\partial f_\Theta(\mathbf{z}, \mathbf{p}_0)}{\partial \mathbf{c}} \tag{4.11}$$

here **c** either stands for the latent vector **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. 4.6 results in points being mapped to their closest neighbor on the infinitesimally deformed surface. Our gradients stem from this choice.

Forward and Backward Passes

Algorithm 1 *DeepMesh* Forward

```
input: latent code $\mathbf{z}$, DIF weights $\Theta$
output: surface mesh $\mathcal{M} = (V, F)$
assemble coarse 3D grid $G$
sample field on grid $S = f_\Theta(\mathbf{z}, G)$
while $G$ has not reached target resolution:
    $G_s = \mathrm{split}(G)$
    $S = S + f_\Theta(\mathbf{z}, G_s)$
    $G = G + G_s$
extract iso-surface $(V, F) = \mathrm{MC}(S)$
Return $\mathcal{M} = (V, F)$
```

Algorithm 2 *DeepMesh* Backward

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

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 [MON⁺19, VKS⁺21]. 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.

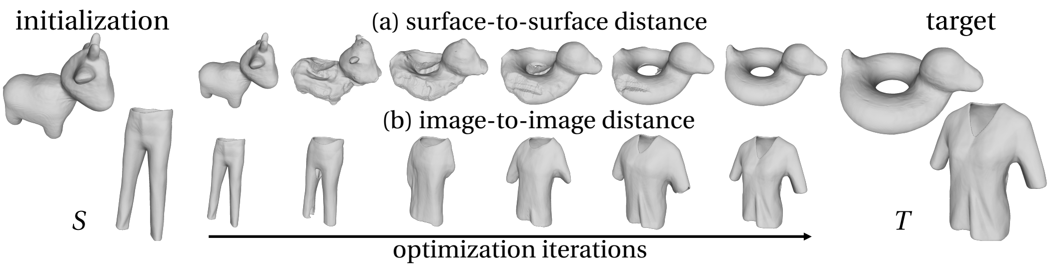


Figure 4.3 – **Topology-Variant Parameterization.** We minimize (a) a surface-to-surface or (b) an image-to-image distance with respect to the latent vector **z** to transform an initial shape into a target one that has a different genus. This demonstrates that we can backpropagate gradient information from mesh vertices to latent vector while modifying surface mesh topology. During the optimization of **z**, it traverses parts of the latent space that do not correspond to smooth or semantically valid output shapes, but it eventually reaches the target.

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 [Yat20] to extract the vertices and vertex normals needed to perform the backward pass. The backward pass then performs the computation of Eq. 4.5. 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. 4.5. 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.

4.4 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.

4.4.1 Differentiable Topology Changes

In the experiment depicted by Fig. 4.3 we train two separate networks f_{Θ_1} and f_{Θ_2} that implement the approximate implicit field of Eq. 4.1. f_{Θ_1} is a deep occupancy network trained to minimize the loss of Eq. 4.3 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. 4.2 on four different articles of clothing, a t-shirt, a pair of pants, a dress, and a sweater. Note that the clothes are represented as open surface meshes without inside/outside regions. Hence, they are not watertight surfaces. To nevertheless represent them using a signed distance function, we first compute unsigned distances to the surfaces, subtract a small $\epsilon = 0.01$ value and treat the result as a signed distance function. This amounts to representing the garments as watertight thin surfaces of thickness 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 [ZWLS21, GSF22, CMPM20, LLL+22].

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.

End-to-end Differentiability

In Fig. 4.3, 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. 4.3.2 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

$$\mathcal{L}_{\text{task1}}(C,T) = \min_{c \in C} d(c,T) + \min_{t \in T} d(C,t)\,, \tag{4.12}$$

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

$$\mathcal{L}_{\text{task2}}(C,T) = \|\text{DR}(C) - \text{DR}(T)\|_1\,, \tag{4.13}$$

where DR is the output of a differentiable rasterizer [KUH18] 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 [MPJ+19, JJHZ20, LZP+20, LWL19], we did *not* have to reformulate the loss functions in terms of implicit surfaces.

Comparison to Implicit Field Differentiable Rendering

Recent advances in differentiable rendering [JJHZ20, LZP⁺20, VSJ22] have shown that is possible to render continuous SDFs differentiably by carefully designing a differentiable version of the sphere tracing algorithm. By contrast, we simply use *DeepMesh*'s end-to-end differentiability to exploit an *off-the-shelf* differentiable rasterizer of meshes to achieve the same result.

To highlight the advantages of doing so, we take f_{Θ_1}, initialize the latent code $\mathbf{z}$ to the one of the cow, and then minimize the silhouette distance $\mathscr{L}_{\text{task2}}$ with respect to the duck. In Tab. 4.1 we compare our approach to [LZP⁺20]. 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.

Method	$10^3 \cdot l_2$ silhouette distance ↓	# network queries ↓	run time [s] ↓
Sphere Tracing [LZP⁺20]	5.97	898k	1.24
DeepMesh	**4.63**	**266k**	**0.29**

Table 4.1 – **Comparison to Implicit Field Differentiable Rendering**. To fit a 2D silhouette, rendering the implicit field with sphere tracing [LZP⁺20] (most efficient settings, 512^2 pixel renderings) is slower and less effective than extracting an explicit mesh with our method and rendering it with an off-the-shelf mesh rasterizer [KUH18] (isosurface at 256^3, 512^2 pixel renderings).

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. 4.4 by repeating the optimization of the latent code $\mathbf{z}$ to minimize the surface-to-surface distance $\mathscr{L}_{\text{task1}}$ of Eq. 4.12 using three different approaches to extracting 3D meshes from iso-surfaces: Marching cubes [LLVT03], marching tetrahedra [DK91] and dual contouring [JLSW02]. 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 [Yat20] that can easily interface with PyTorch [PGC⁺17].

4.4.2 Single view 3D Reconstruction

Single view 3D Reconstruction (SVR) has emerged as a standardized benchmark to evaluate 3D shape representations [CXG⁺16a, FSG17, GFK⁺18, WZL⁺18, CZ19, MON⁺19, PKS⁺18, GMJ19,

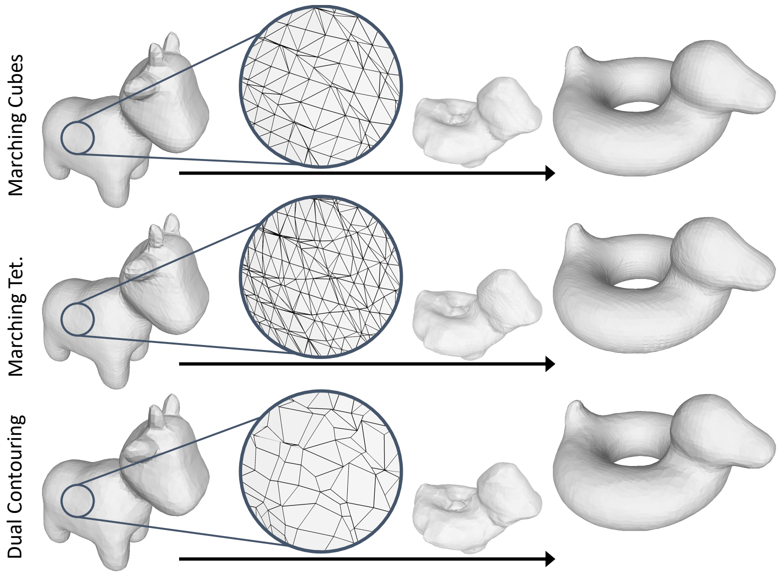


Figure 4.4 – **Different surface extraction methods** yield the same surface-to-surface optimization as in Fig. 4.3(a), and our gradients can be used equally well with Marching Cubes (top), Marching Tetrahedra (middle) or Dual Contouring (bottom).

RR18, XWC⁺19, TRR⁺19]. We demonstrate that it is straightforward to apply our approach to this task on two standard datasets, ShapeNet [CFG⁺15] and Pix3D [XJX⁺18], and improve standard architecture with a refinement step that performs *analysis by synthesis*, a procedure also called *render-and-compare*.

Differentiable Meshes for SVR.

As in [MON⁺19, CZ19], we condition our deep implicit field architecture on the input images via a residual image encoder [HZRS16], which maps input images to latent code vectors $\mathbf{z}$. These latent codes are then used to condition the architecture of Sec. 4.3.1 and compute the value of deep implicit function f_Θ. Finally, we minimize $\mathscr{L}_{\text{imp}}$ (Eq. 4.1) wrt. Θ on a training set of image-surface pairs generated on the ShapeNet Core [CFG⁺15] 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 [XWC⁺19]. 3D supervision points are generated according to the procedure of [PFS⁺19]. 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. 4.2 and 4.3, 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. 4.1. 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 $\mathscr{M}$, that is, given the camera pose associated to the image and the current

Metric	Model	Refine	car	chair
CHD $\cdot 10^4$ ↓	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%)

Table 4.2 – **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).

value of **z**, we project the reconstructed mesh back to the image plane so that the projection matches the object silhouette $\mathscr{S}$ in the image as well as possible. To this end, we define the task-specific loss function L_{task} to be minimized, as discussed in Section 4.3, in one of two ways:

$$\mathscr{L}_{\text{task3}} = \|\text{DR}_{\text{silhouette}}(\mathscr{M}(\mathbf{z})) - T\|_1 , \tag{4.14}$$

$$\mathscr{L}_{\text{task4}} = \sum_{a \in A} \min_{b \in B} \|a - b\|^2 + \sum_{b \in B} \min_{a \in A} \|a - b\|^2 . \tag{4.15}$$

In Eq. 4.14, T denotes the silhouette of the target surface and $\text{DR}_{\text{silhouette}}$ is the differentiable rasterizer of [KUH18] that produces a binary mask from the mesh generated by the latent vector **z**. In Eq. 4.15, $A \subset [-1,1]^2$ denotes the 2D coordinates of T's external contour while $B \subset [-1,1]^2$ denotes those of the external contour of $\mathscr{M}(\mathbf{z})$. We refer the reader to [GRYF21] and the next chapter of this thesis for more details on this objective function. Note that, unlike that of $\mathscr{L}_{\text{task3}}$, the computation of $\mathscr{L}_{\text{task4}}$ 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 [KB15b] to minimize either $\mathscr{L}_{\text{task3}}$ or $\mathscr{L}_{\text{task4}}$ with respect to **z**. This yields four possible combinations of model and loss function and we report their respective performance in Tab. 4.2. 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

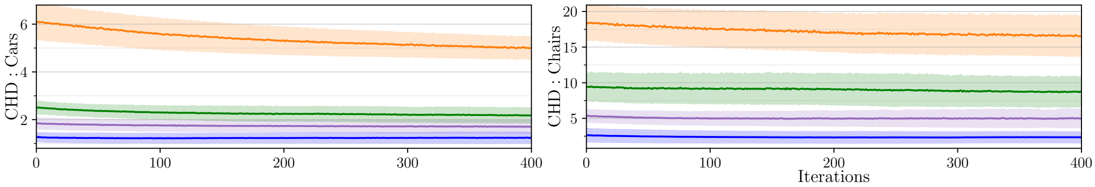


Figure 4.5 – **CHD improvement** over the 400 refinement iterations of *DeepMesh* for cars (top) and chairs (bottom), grouped by quartile in the initial CHD value (from orange = worse 25% of initial shapes, to blue = top 25%). Here *DeepMesh* uses an SDF and refines the contour matching.

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

We have four configurations in total: an SDF or Occupancy network, with refinement using $\mathscr{L}_{\text{task3}}$ (DR) or $\mathscr{L}_{\text{task4}}$ (CHD). All four configurations deliver an improvement in terms of both metrics compared to using the trained network in a simple feed-forward manner. However, the combination of using a signed distance field and minimizing the 2D chamfer distance of $\mathscr{L}_{\text{task4}}$ delivers the largest one. We will therefore refer to it as *DeepMesh* and use it in the remainder of this section, unless otherwise specified.

In Fig. 4.5 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. We hypothesize that signed distance networks perform better due to the 3D supervision points being generated according to the procedure of [PFS$^+$19], which might favor SDF networks over Occupancies.

Comparative Results on ShapeNet

In Tab. 4.3, we compare our approach against state-of-the-art reconstruction approaches of watertight meshes: Generating surface meshes with fixed topology [WZL$^+$18], generating meshes from voxelized intermediate representations [GMJ19], and representing surface meshes using signed distance functions [XWC$^+$19]. We used the standard train/test splits and renderings described above for all benchmarked methods.

DeepMesh (raw) refers to reconstructions obtained using our encoder-decoder architecture based on signed distance fields but without refinement, which is similar to those of [MON$^+$19, CZ19], without any further refinement, whereas *DeepMesh* incorporates the final refinement

Metric	Method	car	chair
CHD $\cdot 10^4 \downarrow$	Mesh R-CNN [GMJ19]	4.55	11.13
	Pixel2Mesh [WZL+18]	4.72	12.19
	DISN [XWC+19]	3.59	8.77
	DeepMesh (raw)	2.96	9.07
	DeepMesh	**2.56**	**8.22**
NC % ↑	Mesh R-CNN [GMJ19]	89.09	74.82
	Pixel2Mesh [WZL+18]	89.00	72.21
	DISN [XWC+19]	91.73	78.58
	DeepMesh (raw)	92.29	78.74
	DeepMesh	**92.56**	**80.17**

Table 4.3 – **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	DISN [XWC+19]	DeepMesh (raw)	DeepMesh
CHD $\cdot 10^3 \downarrow$	5.150	4.850	**4.063**(↓ 16.3%)
NC % ↑	56.94	62.76	**64.28** (↑ 2.4 %)

Table 4.4 – **SVR comparative results on Pix3D Chairs.** Our full approach outperforms our best competitor in all metrics on real images.

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 appendix.

Comparative results on Pix3D.

Whereas ShapeNet contains only rendered images, Pix3D [XJX+18] is a test dataset that comprises real images paired to 3D models. Here, we focus on the chair object category and discard truncated images to create a test set of 2530 images. We use it to compare our method with our best competitor [XWC+19] according to Tab. 4.3. 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. 4.4 and in Fig. 4.6. 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 [XWC+19]. As for ShapeNet, our full model *DeepMesh* outperforms all other approaches.

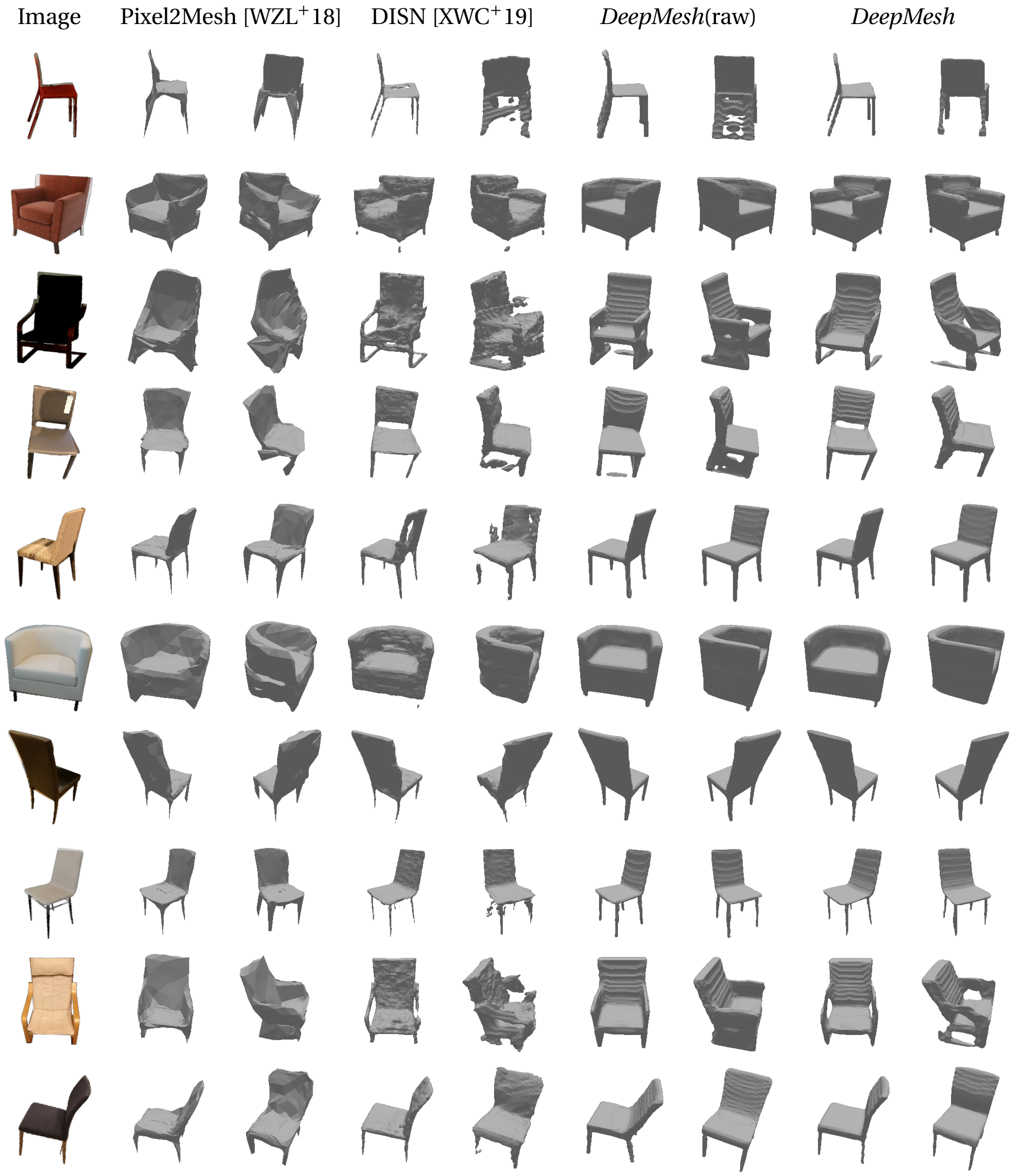


Figure 4.6 – **Comparative results for SVR on Pix3D.** We compare our refined predictions to runner-up approaches for the experiment in Tab. 4.4. *DeepMesh* can represent arbitrary topology as well as learn strong shape priors, resulting in reconstructions that are consistent even when observed from view-points different from the input one. For more results see Appendix.

4.4.3 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 [TK11, XLY$^+$17, BRFF18, UB18] 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 [BRFF18, UB18] 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.

Experimental Setup.

We started with the ShapeNet car split by automatic deletion of all the internal car parts [SSB13] and then manually selected $N = 1400$ shapes suitable for CFD simulation. For each surface $\mathscr{M}_i$ we ran OpenFoam [JJT$^+$07] to predict a pressure field p_i exerted by air traveling at 15 meters per second towards the car. The resulting training set $\{\mathscr{M}_i, p_i\}_{i=1}^{N}$ was then used to train a Mesh Convolutional Neural Network [FLWM18] g_β to predict the pressure field $p = g_\beta(\mathscr{M})$, as in [BRFF18]. We use $\{\mathscr{M}_i\}_{i=1}^{N}$ to also learn the representation of Sec. 4.3.1 and train the network that implements f_Θ of Eq. 4.1. As in Section 4.4.2, we train both an occupancy network and signed-distance network, which we dub *DeepMesh-OCC* and *DeepMesh-SDF*, respectively.

The shapes are deformed to minimize the aerodynamic objective function

$$\mathscr{L}_{\text{task5}}(\mathscr{M}) = \iint_{\mathscr{M}} g_\beta \, \mathbf{n}_x \, \mathrm{d}\mathscr{M} + \mathscr{L}_{\text{constraint}}(\mathscr{M}) + \mathscr{L}_{\text{reg}}(\mathscr{M}) \,, \tag{4.16}$$

where $\mathbf{n}_x$ denotes the projection of surface normals along airflow direction, the integral term approximates drag given the predicted pressure field [MOHR13], $\mathscr{L}_{\text{constraint}}$ is a loss that forces the result to preserve space the engine and the passenger compartment, and $\mathscr{L}_{\text{reg}}$ is a regularization term that prevents $\mathbf{z}$ from moving too far away from known shapes. $\mathscr{L}_{\text{constraint}}$ and $\mathscr{L}_{\text{reg}}$ are described in more detail in the appendix. $\mathscr{L}_{\text{task5}}$ is formulated as a global optimization, and does not explicitly encourage the optimized shape to adhere to the initial one. However, because of the complex landscape of the latent space, different initializations converge to different shapes, as visualized in the supplementary material.

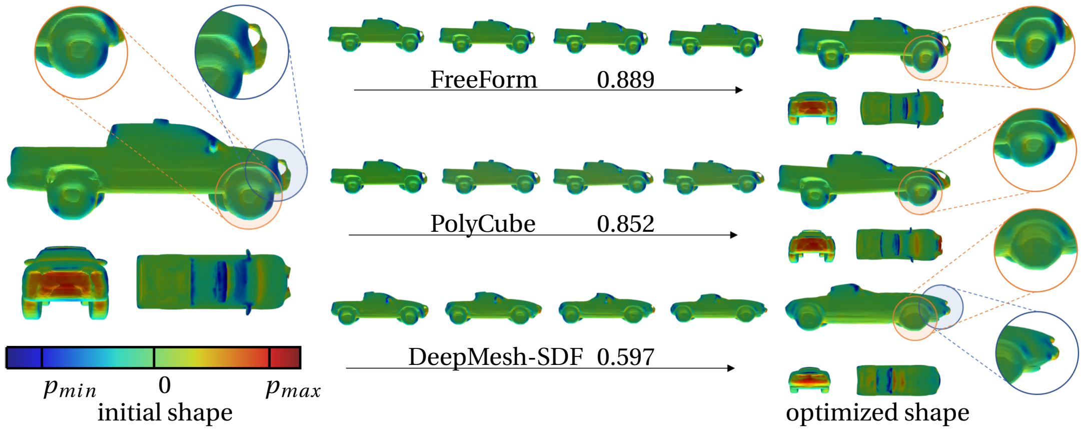


Figure 4.7 – **Drag minimization.** Starting from an initial shape (left column), $\mathscr{L}_{task}$ is minimized using three different parameterizations: FreeForm (top), PolyCube (middle), and our *DeepMesh* (bottom). The middle column depicts the optimization process and the relative improvements in terms of $\mathscr{L}_{task}$. 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).

Parameterization	None	Scaling	FreeForm [BRFF18]	PolyCube [UB18]	DeepMesh-SDF	DeepMesh-OCC
Degrees of Freedom	~ 100k	3	21	~ 332	256	256
Simulated $\mathscr{L}^{\%}_{\text{task}} \downarrow$	not converged	0.931 ± 0.014	0.844 ± 0.171	0.841 ± 0.203	$\mathbf{0.675 \pm 0.167}$	0.721 ± 0.154

Table 4.5 – **CFD-driven optimization**.We minimize drag on car shapes comparing different surface parameterizations. Numbers in the table (avg ± std) denote relative improvement of the objective function $\mathscr{L}^{\%}_{\text{task}} = \mathscr{L}_{\text{task}} / \mathscr{L}^{t=0}_{\text{task}}$ for the optimized shape, as obtained by CFD simulation in OpenFoam.

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 [BRFF18], which extends scaling to higher order terms as well as periodical ones and (4) the *PolyCube* parameterization of [UB18] 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. 4.16 for a subset of 8 randomly chosen cars in Table 4.5, and show qualitative ones in Fig. 4.7. 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. 4.7(c). We provide additional results in the appendix. As can be seen in Table 4.2, *DeepMesh-SDF* slightly outperforms *DeepMesh-OCC*. We conjecture this is due to our sampling strategy, which follows closely the one of [PFS$^+$19] and might therefore favor SDF networks.

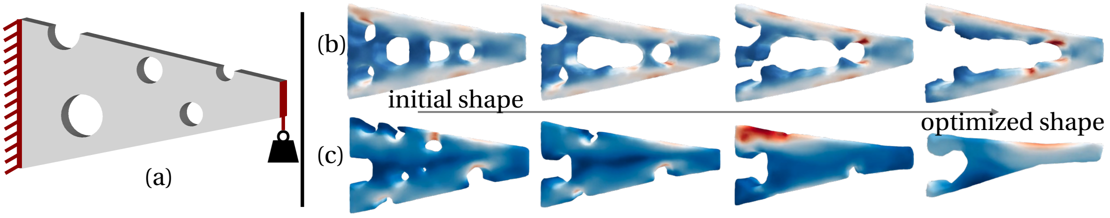


Figure 4.8 – **Stress minimization.** **(a)** Beams are attached to a wall on their left side and bear a constant load on their right side. They are all of the same length, but have variable thickness, height, and number of randomly positioned holes. **(b,c)** Starting from two initial beams, we use our *DeepMesh* parameterization and minimize both mechanical stresses—shown as colors—and volume. This allows for changing their topology and results in plausible beams.

4.4.4 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. 4.8(a). For each resulting surface $\mathcal{M}_i$, we used the finite element solver FINO [Sea16] 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 [FLWM18] 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. 4.3.1 and train the network that implements f_Θ of Eq. 4.1 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

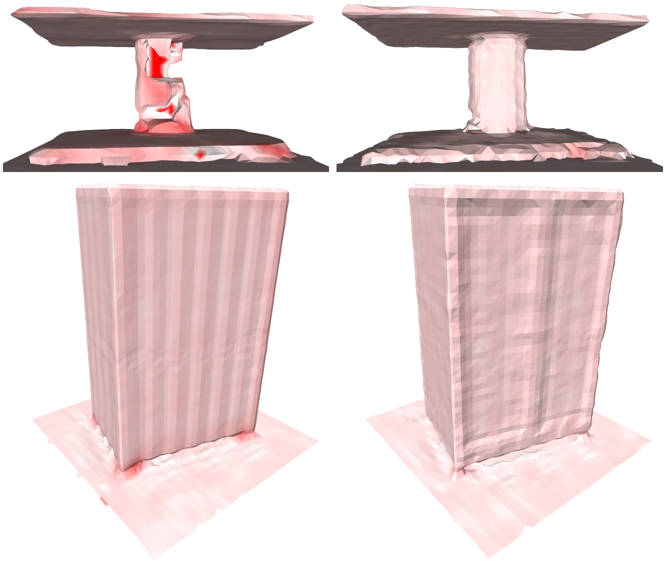
$$\mathcal{L}_{\text{task6}}(\mathcal{M}) = \iint_{\mathcal{M}} g_\beta \, \mathrm{d}\mathcal{M} + \lambda \mathcal{L}_{\text{volume}}(\mathcal{M}) + \mathcal{L}_{\text{reg}}(\mathcal{M}) \,, \tag{4.17}$$

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, $\mathcal{L}_{\text{volume}}$ is a loss encouraging the beams to have a small volume, λ is a scalar balancing the two objectives, and $\mathcal{L}_{\text{reg}}$ is the regularization term of Eq. 4.16. $\mathcal{L}_{\text{volume}}$ penalizes low SDF values on a regular 3D grid G of query points. We write it as

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

	Original	Optimized
CHD ↓	0.954	**0.529**
IoU ↑	81.67 %	**88.74 %**

Table 4.6 – **Refining reconstructed scenes from ConvOccNet**.



Raw Refined

Figure 4.9 – **Refining CON [PNM+20] 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. 4.1(c - left). Chamfer error are shown in red, and reduced after refinement.

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

Fig. 4.8(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.

4.4.5 Scene Reconstruction

In the examples of Section 4.4.2 and 4.4.3, 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 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 the ability of our method to handle not only SDFs but also occupancy fields is valuable.

We use the pretrained scene reconstruction network of [PNM+20] that regresses an occupancy

Metric	Category	*DeepSDF*	*DeepMesh*
CHD $\cdot 10^4$ (↓)	*Chairs*	2.75	**2.56** (↓ 6.9 %)
	Lamps	8.12	**7.59** (↓ 6.5 %)
NC (↑)	*Chairs*	80.9	**81.9** (↑ 1.2 %)
	Lamps	73.9	**75.1** (↑ 1.6 %)

Table 4.7 – **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.

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 $G \in \mathbb{R}^{32\times32\times32\times32}$ and in three projected 2D feature planes $P_1, P_2, P_3 \in \mathbb{R}^{128\times128\times32}$, 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

$$\mathscr{L}_{\text{task6}} = \sum_{p\in P} \min_{a\in M_{\mathbf{z}}} \|a-b\|^2 \,, \tag{4.18}$$

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

In Tab. 4.6 we compute the average Chamfer distance and Intersection over Union (IoU) [PNM⁺20] with the ground truth meshes for the 2 provided test scenes. The improvements are substantial, as can be seen in Fig. 4.1(c) and Fig. 4.9.

4.4.6 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 isosurface extraction scheme can also be used to train f_Θ and to backpropagate gradients to the network weights Θ. In other words, we consider 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 [PFS⁺19]. In the original paper, Θ along with the latent representations are learned by minimizing the implicit loss function $\mathscr{L}_{\text{imp}}$ of Eq. 4.1 and its accuracy is assessed in terms of the chamfer

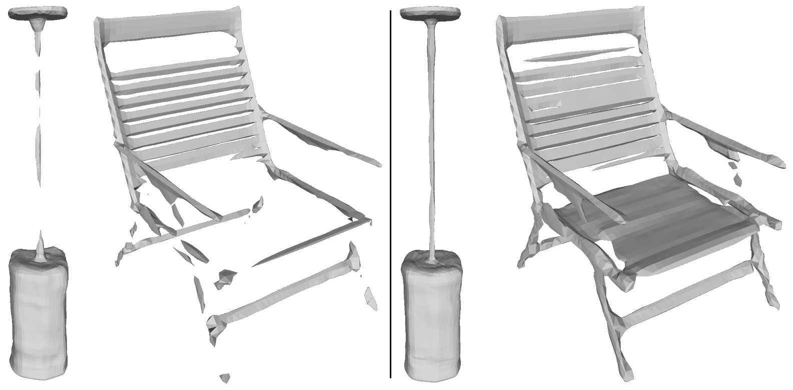


Figure 4.10 – **Learning network weights by minimizing the Chamfer loss. DeepSDF** is trained by minimizing errors on predicted SDF values, which can result in thin components being poorly reconstructed. **DeepMesh** is trained by also minimizing Chamfer distances, which penalizes such mistakes. The resulting network reconstructs thin structures better.

distance between target shapes and reconstructed ones. It can be written as

$$\mathscr{L}_{\text{chamfer}} = \sum_{\mathbf{p}\in P} \min_{\mathbf{q}\in Q} \|\mathbf{p}-\mathbf{q}\|_2^2 + \sum_{\mathbf{q}\in Q} \min_{\mathbf{p}\in P} \|\mathbf{p}-\mathbf{q}\|_2^2 \,, \tag{4.19}$$

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

In practice, we first minimize $\mathscr{L}_{\text{imp}}$ to learn a first version of Θ and of the latent vectors $\mathbf{z}$ of Eq. 4.1. As described in [PFS$^+$19], this yields a network that we will refer to as *DeepSDF*. We then freeze the latent vectors and minimize $\mathscr{L}_{\text{imp}} + \mathscr{L}_{\text{chamfer}}$ 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 [CFG$^+$15]. For chairs, we use the same data split and samples as in Sec. 4.4.2. 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. 4.10 and quantitatively in Tab. 4.7, where we report metrics on the test sets by fitting latent codes to SDF samples of unseen shapes. Minimizing $\mathscr{L}_{\text{chamfer}}$ 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 appendix. In this specific scenario, where

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

Table 4.8 – **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 isosurface extraction.

volumetric supervision is available, we simply initialize the network using $\mathscr{L}_{\text{imp}}$ and retain this term during the fine-tuning phase. An alternative approach involves initializing the signed distance function with that of a sphere [MCR22].

4.4.7 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. 4.8, we compare forward and backward times for losses either on the field's values ($\mathscr{L}_{data}$ of Eq. 4.2) or through isosurface extraction ($\mathscr{L}_{\text{chamfer}}$ of Eq. 4.19). The network is a DeepSDF with 8 layers of size 512, and we report average times over the testing chairs of ShapeNet. For $\mathscr{L}_{data}$ we apply it on the default amout of 8192 points per batch. For $\mathscr{L}_{\text{chamfer}}$ we run isosurface 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 isosurface extraction is 2 orders of magnitude slower than simply computing SDF values. However, with the coarse-to-fine strategy presented in Sec. 4.3.2, 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.

4.5 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 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 [CBZ+19] 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 [AL20a].

5 Sketch2Mesh: Reconstructing and Editing 3D Shapes from Sketches

This chapter is based on the conference paper [GRYF21]:

B. Guillard*, E. Remelli*, P. Yvernay, and P. Fua, *Sketch2mesh: Reconstructing and Editing 3D Shapes from Sketches*, at ICCV, 2021.

5.1 Introduction

Reconstructing 3D shapes from hand-drawn sketches has the potential to revolutionize the way designers, industrial engineers, and artists interact with Computer Aided Design (CAD) systems. Not only would it address the industrial need to digitize vast amounts of legacy models, an insurmountable task, but it would allow practitioners to interact with shapes by drawing in 2D, which is natural to them, instead of having to sculpt 3D shapes produced by cumbersome 3D scanners.

Current deep learning approaches [LGK$^+$17, DAI$^+$18, ZGZS20, ZQG$^+$20] that regress 3D point clouds and volumetric grids from 2D sketches have shown promise despite being trained

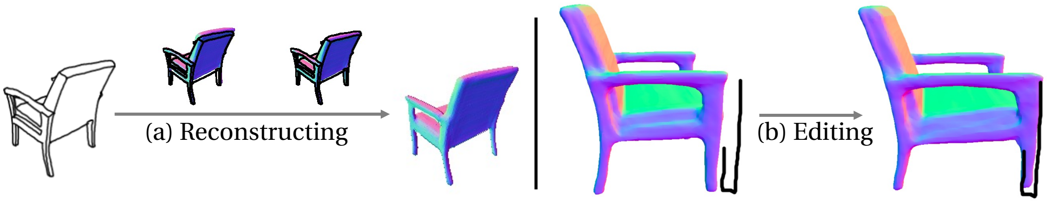


Figure 5.1 – **Sketch2Mesh.** We propose a pipeline for reconstructing and editing 3D shapes from line drawings. We train an encoder/decoder architecture to regress surface meshes from synthetic sketches. Our network learns a compact representation of 3D shapes that is suitable for downstream optimization: **(a)** When presented with sketches drawn in a style different from that of the training ones– for example a real drawing – aligning the projected external contours to the input sketch bridges the domain gap. **(b)** The same formulation can be used to enable unexperienced users to edit reconstructed shapes via simple 2D pen strokes.

on synthetic data, but yield coarse 3D surface representations that are cumbersome to edit. Furthermore, they require multi-view sketches for effective reconstruction [DAI+18] or are restricted to a fixed set of views [LGK+17].

Meanwhile Single View Reconstruction (SVR) approaches have progressed rapidly thanks to the introduction of new shape representations [GFK+18, PFS+19, MON+19, RLR+20] along with novel architectures [WZL+18, GMJ19, GRF20, XWC+19] that exploit image-plane feature pooling to align reconstructions to input images. Hence, it can seem like a natural idea to also use them for reconstruction from sketches. Unfortunately, as we will show, the sparse nature of sketch images makes it difficult for state-of-the-art SVR networks relying on local feature pooling from the image plane to perform well. This difficulty is compounded by the fact that different people sketch differently, which introduces a great deal of variability in the training process and makes generalization problematic. Furthermore, these architectures do not learn a compact representation of 3D shapes, which makes the learned models unsuitable for down stream applications requiring a strong shape prior, such as shape editing.

To overcome these challenges, we train an encoder/decoder architecture [RLR+20] to produce a 3D mesh estimate given an input line drawing. This yields a compact latent representation that acts as an information bottleneck. At inference time, given a previously unseen camera-calibrated sketch, we compute the corresponding latent vector and refine its components to make the projected 3D shape it parameterizes match the sketch as well as possible. In effect, this compensates for the style difference between the input sketch and those that were used for training purposes. We propose and investigate two different ways to do this:

1. *Sketch2Mesh/Render*. We use a state-of-the-art image translation technique [IZZE17] trained to synthesize foreground/background images from sketches and then use the resulting images as targets for differentiable rasterization [RLT+20, RLR+20, PFAK20].
2. *Sketch2Mesh/Chamfer*. We directly optimize the position of the 3D shape's projected contours to make them coincide with those of the input sketch by minimizing a 2D Chamfer distance.

Remarkably, *Sketch2Mesh/Chamfer*, even though simpler, works as well or better than *Sketch2Mesh/Render*. The former exploits only *external* object contours for refinement purposes, which helps with generalization because most graphics designers draw these external contours in a similar way. It also makes it unnecessary the auxiliary network that turns sketches into foreground/background images.

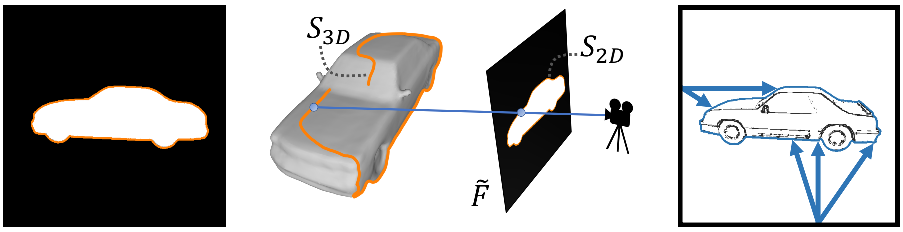
A further strength of *Sketch2Mesh/Chamfer* is that it does not require backpropagation from a full rasterized image but only from sparse contours. Hence, it is naturally applicable for local refinement given a camera-calibrated partial sketch. And, unlike earlier work [NSACO05, KHR02, KG07] on shape editing from local pen strokes it allows us to take into account a strong shape prior by relying on the latent vector, ensuring that shapes can be edited robustly with sparse 2D pen strokes.

5.2 Related Work: 3D Reconstruction from Sketches

Reconstructing 3D models from line drawings has also been an active research area for many decades. Early attempts tackled the inherent ambiguity of this inverse problem by either assuming that the drawn lines represent specific shape features [MM89, IMT06] or by constraining the class of 3D shapes that can be handled [LF92, LS96, CSMS13, JHR+15]. More recently inflatable surface models [DSC+20] demonstrated easy animation of the reconstructed shapes, but still constrain the artist to draw from a side view of the object and are limited to a fixed topology. The emergence of deep learning has given rise to models [LGK+17, DAI+18, JFD20] that can be far more expressive and have therefore boosted both the performance and generalization of algorithms that parse sketches into 3D shapes. Given an input sketch, [LGK+17] regress depth and normal maps from 12 viewpoints, and fuse them to obtain a dense point cloud from which a surface mesh is extracted. Their pipeline, however, must be trained for each input sketch viewpoint, making it incompatible with a free viewpoint sketching interface. In [DAI+18], a 3D convolutional network trained on a catalog of simple shape primitives regresses occupancy grids from sketches. In addition to the limited output resolution, a refinement strategy based on sketches from multiple views is needed for effective reconstruction. [JFD20] jointly projects 3D shapes and their front, side and top views occluding contours in the embedding space of a VAE. Their pipeline is trained on a single sketch style (occluding contours) and outputs volumetric grids. At inference time it retrieves the closest embedding code that was seen during training, thus limiting its generalization capabilities.

Single View Reconstruction. Recently, Single View Reconstruction (SVR) from RGB images has also experienced tremendous progresses thanks to both the introduction of new shape representations discussed above and to he introduction of new SVR architectures [WZL+18, GMJ19, GRF20, XWC+19] relying on image-plane feature pooling to align reconstructions to input images. Unfortunately, many of these methods rely on feature pooling and therefore lack a compact latent representation that can be used for downstream applications that require strong shape priors, such as refinement or editing. However, there are SVR methods that feature compact surface representations and we discuss below those that leverage either differentiable rendering or contours, as we do.

Refinement using differentiable rendering. Recent work [RLT+20, RLR+20, PFAK20] has shown that 2D buffers -such as silhouettes or depth maps- can be used to refine 3D reconstructions produced by encoder/decoder architectures and thus allow networks trained on synthetic RGB renders to yield accurate reconstruction on real world images. These approaches rely on either estimating 2D buffers from input images — using state-of-the-art segmentation/depth estimation networks trained on large-scale real world datasets [LMB+14] — or acquiring the additional information through specific sensors. Applying refinement techniques to line-drawings would require to use an auxiliary network to infer occupancy masks from input sketches. However we found that such networks struggle at generalizing to different sketching styles. This is due to the lack of diversified large-scale line-drawings datasets [GSH+19], and makes refinement through differentiable rasterization less effective,



(a) (b) (c)

Figure 5.2 – **External contours in 2D and 3D. (a)** The external contours of the projected mesh are shown in orange. They form the S_{2D} set of Eq. 5.4. **(b)** The corresponding 3D points on the mesh are also overlaid in orange. They form the S_{3D} set of Eq. 5.3. **(c)** We filter the original sketch to keep only the external contours, which will be matched against S_{2D}.

or in some cases detrimental.

Refinement by matching Silhouettes. Silhouettes have long been used to track articulated and rigid objects by modeling them using volumetric primitives whose occluding contours can be computed given a pose estimate. The quality of these contours can then be evaluated using either the chamfer distance to image edges [GD95] or more sophisticated measures [ST03, AT04]. Other approaches to exploiting external contours rely on minimizing the distance between the 3D model and the lines of sights defined by these contours [ISF07]. Our approach follows this tradition but combines silhouette alignment to a far more powerful latent representation.

5.3 Method

5.3.1 Formalization

Let $\mathbf{C} \in \{0,1\}^{H\times W}$ be a binary image representing a sketch and let $\Lambda : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ denote the function that projects 3D points into that image. By convention, $\mathbf{C}[i,j]$ is 0 if it is marked by a pen stroke, and $\mathbf{C}[i,j] = 1$ otherwise.

We learn an encoder $\mathscr{E}$ and a decoder $\mathscr{D}$ such that $\mathscr{D} \circ \mathscr{E}(\mathbf{C})$ yields a mesh $\mathscr{M}_\Theta = (\mathbf{V}_\Theta, \mathbf{F}_\Theta)$. $\Theta = \mathscr{E}(\mathbf{C})$ is the latent vector that parameterizes our shapes. $\mathbf{V}_\Theta$ and $\mathbf{F}_\Theta$ represent the 3D vertices and facets. In practice, we use the MeshSDF encoding/decoding network architecture of [RLR$^+$20]. In general, $\mathscr{M}_\Theta$ represents a 3D shape whose projection $\Lambda(\mathscr{M}_\Theta)$ only roughly matches the sketch $\mathbf{C}$. Hence, our subsequent goal is to refine Θ so as to improve the match.

We can achieve this in of two ways. We can turn the sketch into a foreground/background image and use differentiable rasterization to ensure that the projection of $\mathscr{M}_\Theta$ matches that image. Alternatively, we can minimize the 2D Chamfer distance between the sketch and the projection. We describe both alternatives below.

5.3.2 Using Differential Rendering

In this method that we dubbed *Sketch2Mesh/Render*, we train an image translation technique [IZZE17] to synthesize foreground/background images from sketches. We denote as $\mathbf{M} \in \{0,1\}^{H\times W}$ this foreground/background image estimated from the input sketch **C**. On the other hand, we use the differentiable rasterizer [RRN+20] $\mathscr{R}^{F/B}$ to render a foreground/background mask $\widetilde{\mathbf{M}} = \mathscr{R}_{\Lambda}^{F/B}(\mathscr{M}_{\Theta})$ of the projection of $\mathscr{M}_{\Theta}$ by Λ. In $\widetilde{\mathbf{M}}$, a pixel value is 1 if it projects to the surface of the mesh $\mathscr{M}_{\Theta}$, and 0 otherwise. Finally, we refine $\mathscr{M}_{\Theta}$ shape by minimizing

$$\mathscr{L}_{F/B} = \left\| \mathbf{M} - \widetilde{\mathbf{M}} \right\|^2 , \tag{5.1}$$

the L_2 difference between **M** and $\widetilde{\mathbf{M}}$ with respect to Θ.

While conceptually straightforward, this approach is in fact quite complex because it depends on two off-the-shelf but complex pieces of software, the rasterizer [RRN+20] and image-translator [IZZE17], one of which has to be trained properly. We now turn to a simpler technique that can be implemented from scratch and does not rely on an auxiliary neural network.

5.3.3 Minimizing the 2D Chamfer Distance

The simpler *Sketch2Mesh/Chamfer* approach involves directly finding those 3D mesh points that project to the contour of the foreground image and then minimizing the Chamfer distance between this contour and the sketch.

Finding External Contours in 2D and 3D

To identify surface points on $\mathscr{M}_{\Theta}$ that project to exterior contour pixels, we first use Λ to project the whole mesh onto a $H \times W$ binary image $\widetilde{\mathbf{F}}$ in which all pixels are one except those belonging to external contours, such as those shown in orange in Fig. 5.2(a). Then, for each zero-valued pixel **p** in $\widetilde{\mathbf{F}}$, we look for a 3D point **P** on one of the mesh facets that projects to it, that is, a point that is visible and such that $\Lambda(\mathbf{P}) = \mathbf{p}$. In theory, this can be done by finding to which facet **p** belongs and then computing the intersection between the line of sight and the plane defined by that facet. In practice, we use Pytorch3d [RRN+20] which provides us with the facet number along with the barycentric coordinates of **P** within that facet. Hence, we write

$$\mathbf{P} = \alpha_1 \mathbf{V}_1 + \alpha_2 \mathbf{V}_2 + \alpha_3 \mathbf{V}_3 , \tag{5.2}$$

with $\mathbf{V}_1$, $\mathbf{V}_1$ and $\mathbf{V}_3$ are the vertices of the fact to which **P** belongs and $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Since the coordinates of the three vertices are differentiable functions of Θ, so are those of **P**. Repeating this operation for all external contour points yields a set of 3D points S_{3D} such that

$$\forall \mathbf{P} \in S_{3D} \quad \widetilde{\mathbf{F}}[\Lambda(\mathbf{P})] = 0 , \tag{5.3}$$

along with a corresponding set of 2D projections

$$S_{2D} = \{\Lambda(\mathbf{P}) | \mathbf{P} \in S_{3D}\} \,. \tag{5.4}$$

Fig. 5.2(b) depicts such a set.

Objective function

To exploit the target sketch **C**, we first filter it to only preserve external contours. To this end, we shoot rays from the 4 image borders and only retain the first black pixels hit by a ray, as shown in Fig. 5.2(c). This yields a filtered sketch $\mathbf{F} \in \{0,1\}^{H \times W}$. As before, $\mathbf{F}[\mathbf{p}] = 0$ for pixels **p** belonging to external contour and $\mathbf{F}[\mathbf{p}] = 1$ for others. The ray-shooting algorithm we use is described in details in the appendix.

Our goal being for **F**, the filtered sketch, and $\widetilde{\mathbf{F}}$, the external contours of the projected triangulation introduced in the previous subsection, to match as well as possible, we write the objective function to be minimized as the bidirectional 2D Chamfer loss

$$\mathcal{L}_{CD} = \sum_{\mathbf{u} \in S_{2D}} \min_{\mathbf{v} | \mathbf{F}[\mathbf{v}]=0} \|\mathbf{u} - \mathbf{v}\|^2 + \sum_{\mathbf{v} | \mathbf{F}[\mathbf{v}]=0} \min_{\mathbf{u} \in S_{2D}} \|\mathbf{u} - \mathbf{v}\|^2 \,. \tag{5.5}$$

The coordinates of the 3D vertices in S_{3D} are differentiable with respect to Θ. Since Λ is differentiable, so are their 2D projections in S_{2D} and $\mathcal{L}_{CD}$ as whole.

5.3.4 Using a Partial Sketch

Minimizing the 2D Chamfer distance between external contours as described above does not require the input sketch to depict the shape in its entirety. This enables us to take advantage of partial sketches made of a single stroke. In this case, we can simply take the filtered sketch **F** introduced above to be the sketch itself. But we must ensure that parts of the surface which project far away from the sketch remain unchanged. The rationale for this is that the initial shape should be preserved except where modifications are specified. To this end, we regularize the refinement procedure as follows.

Given the initial value Θ_0 of the latent vector we want to refine along with differentiable rasterizers [RRN$^+$20] $\mathcal{R}^N$ and $\mathcal{R}^{F/B}$ that return the normal maps $\mathbf{N}_\Theta$ and foreground/background mask $\mathbf{M}_\Theta$ given mesh $\mathcal{M}_\Theta$, respectively, we minimize

$$\mathcal{L}_{partial} = \mathcal{L}_{CD} + \left\| \mathbb{1}_t \circ (\mathbf{M}_\Theta - \mathbf{M}_{\Theta_0}) \right\|^2 + \left\| \mathbb{1}_t \circ (\mathbf{N}_\Theta - \mathbf{N}_{\Theta_0}) \right\|^2$$

where $\mathcal{L}_{CD}$ is the Chamfer distance of Eq. 5.5, $\mathbb{1}_t$ is a mask that is zero within a distance t of the sketch and one further away, and $\circ$ is the element wise product. In other words, the parts of the surface that project near to the sketch should match it and the others should keep their original normals and boundaries.

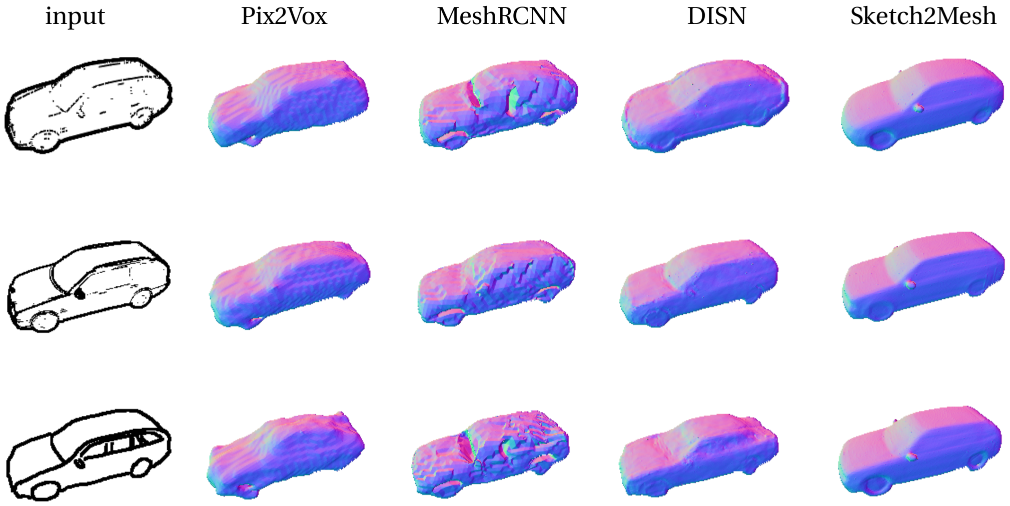


Figure 5.3 – **Robustness to changes in sketch style.** Given a `Suggestive` sketch (top), a `SketchFD` one (middle), or a hand-drawn one (bottom), Sketch2Mesh—unlike Pix2Vox [DAI+18], MeshRCNN [GMJ19], and DISN [XWC+19]—yields reconstructions that are similar to each other and close to the ground-truth.

Crucially, this is something that could not be done using the approach of Section 5.3.2, which requires complete sketches. This comes at the cost of having to use a differential renderer, unlike the approach of Section 5.3.2. But this still does not require a trained network for image translation, which makes it easy to deploy.

5.4 Experiments

5.4.1 Datasets

Publicly available large-scale line-drawings datasets with associated 3D models are rare. We therefore test our approach on two datasets, one for chairs that is available [ZQG+20] and another for cars that we created ourselves. To further test, and crucially, to train our approach, we used 3D models from the well-established ShapeNet [CFG+15] to render 2D sketches.

Rendered Car and Chair Sketches. We use the car and chair categories from ShapeNet [CFG+15] both for training and testing. We adopt the same train/test splits as in [RLR+20]. For cars we use 1311 training samples and 113 test samples. The equivalent numbers are 5268 and 127 for chairs. For each object and corresponding 3D mesh, we randomly sample 16 azimuth and elevation angles. The cameras point at the object centroid while their distance to it and their focal lengths are kept fixed. To demonstrate robustness to sketching style, we generate two different 256 × 256 binary sketches for each viewpoint, as shown in the top two rows a Fig. 5.3. We will refer to them as `Suggestive` and `SketchFD` sketches, as described below.

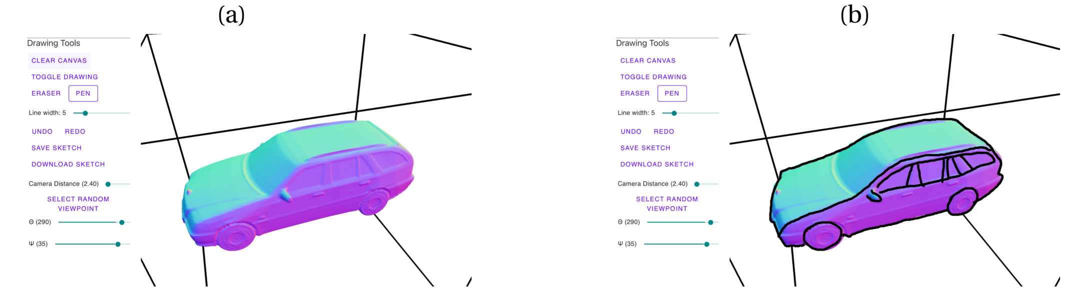


Figure 5.4 – **Data acquisition interface.** (a) To guide unexperienced users and limit imprecision, we display the normal map as seen from a specific viewpoint. (b) The user can use a pen to draw freely on the resulting image.

Suggestive. We use the companion software of [DFRS03] to render sketches displaying that contain both occluding and suggestive contours. Suggestive contours are lines drawn on visible parts of the surface where a true occluding contour would first appear given a minimal viewpoint change. They are designed to emulate real line drawings in which lines other lines than the occluding contours are drawn to increase expressivity.

SketchFD. We also use the older rendering approach of [ST90]. We run an edge detector over the normal and depth maps of the rendered object. Edges in the depth map correspond to depth discontinuities while edges in the normal map correspond to sharp ridges and valleys. This yields synthetic sketches that, although conveying the same information, look very different from the ones on [DFRS03], as can be seen on the left of Fig. 5.3.

Hand-Drawn Car Sketches We asked 5 students with no prior experience in 3D design to draw by-hand the 113 cars from the ShapeNet test set. To this end, we developed the sketching interface depicted by Fig. 5.4 that runs on a standard tablet. The participants drew over normal maps rendered from the selected viewpoint so as to provide them guidance and ensure they all drew a similar car and used a known perspective. However, they were free to make the pen strokes they wanted. Hence, this dataset thus exhibits natural variations of style. To allow for comparison with results on the rendered sketches, we used the same viewpoint, which we will use to demonstrate that style change by itself is an obstacle to generalization for many methods.

Hand-Drawn Chair Sketches We use 177 chair sketches from the ProSketch dataset [ZQG+20]. The chairs are seen from the front, profile, or a 45°azimuth view, as shown in Fig. 5.5. These viewpoints do not match the randomly selected ones we used for training, which makes this dataset especially challenging. Sample sketches and reconstructions are shown in Fig. 5.5

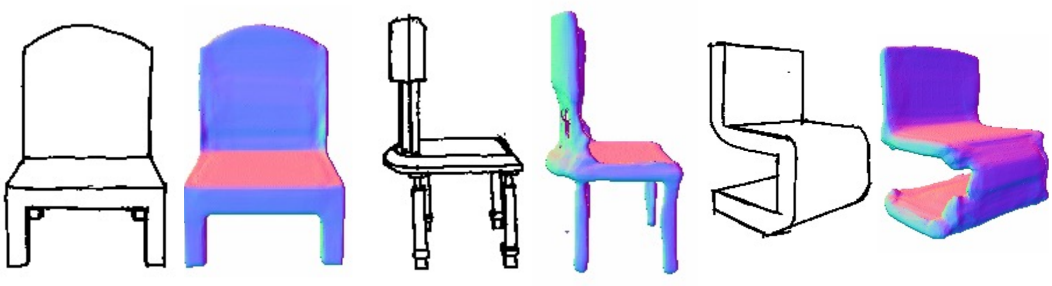


Figure 5.5 – **ProSketch.** Input hand-drawn chair sketches [ZQG+20] and *Sketch2Mesh* reconstructions.

5.4.2 Metrics

As reconstruction metric, we use a 3D Chamfer loss (CD-l_2, the lower the better). It is computed by sampling $N = 10000$ points on the reconstructed mesh to form a first point cloud $\mathbf{C}_1$ and N on the ground truth mesh to form a second point cloud $\mathbf{C}_2$. We then compute

$$\text{CD-}l_2 = \frac{1}{N}\sum_{x\in\mathbf{C}_1}\min_{y\in\mathbf{y}}\|x-y\|^2 + \frac{1}{N}\sum_{y\in\mathbf{C}_2}\min_{x\in\mathbf{x}}\|y-x\|^2 \ .$$

We also report a normal consistency measure (NC, the higher the better), by taking the average pixel-wise dot product between normal maps of the reconstructed shape and the ground truth one.

5.4.3 Choosing the Best Method

Metric	Method	**Cars**, test drawing style:			**Chairs**, test drawing style:		
		Suggestive	SketchFD	Hand-drawn	Suggestive	SketchFD	Hand-drawn
CD-$l_2 \cdot 10^3$ ↓	*Initial*	1.613	4.658	6.818	8.572	15.691	18.752
	Sketch2Mesh/Render	**1.400**	4.253	5.752	7.471	12.865	17.519
	Sketch2Mesh/Chamfer	1.420	**3.132**	**4.395**	**7.180**	**12.248**	**13.787**
Normal Consistency ↑	*Initial*	91.14	84.73	81.40	80.86	72.83	61.17
	Sketch2Mesh/Render	**92.41**	86.18	83.88	**83.99**	75.37	65.23
	Sketch2Mesh/Chamfer	92.20	**87.00**	**84.75**	82.61	**76.27**	**67.67**

Table 5.1 – **Reconstruction metrics** when using the encoding/decoding network trained on `Suggestive` synthetic sketches of cars (left) and of chairs (right), and tested on all 3 datasets. We show *initial* results before refinement and then using our two refinement methods. Note that *Sketch2Mesh/Chamfer* does better than *Sketch2Mesh/Render* on the styles it has *not* been trained for, indicating a greater robustness to style changes.

Recall from the method section, that we have proposed two variants of our approach to refining our 3D meshes. *Sketch2Mesh/Render* operates by turning the sketch into a foreground/background image and minimizing the distance between that image and the mesh projection while *Sketch2Mesh/Chamfer* deforms the mesh to minimize the 2D Chamfer distance between the external contours of its projection and those of the sketch.

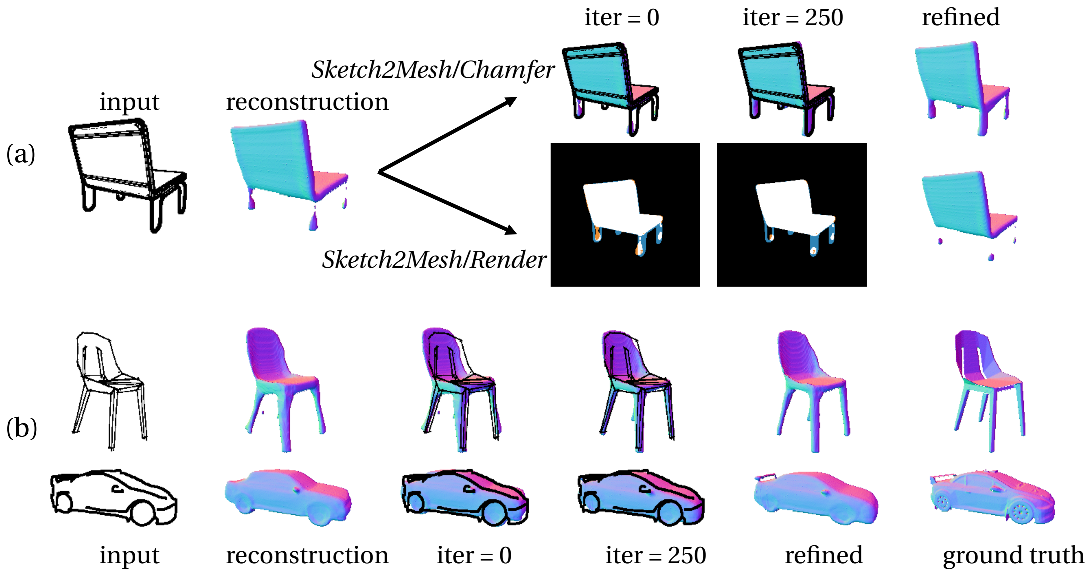


Figure 5.6 – **Mesh refinement. (a)** Comparison of *Sketch2Mesh/Chamfer* (top) and *Sketch2Mesh/Render* (bottom). *Sketch2Mesh/Chamfer* handles thin components such as the legs of the chair better because it leverages sparse information. We examine this in more detail in the appendix. **(b)** *Sketch2Mesh/Chamfer* results on challenging line drawings of a chairs and a car. We show the iterations from the initial mesh produced by the network that takes the sketch as input, which is then progressively refined.

Once the latent representation has been learned on either `Suggestive` or `SketchFD` contours, *Sketch2Mesh/Chamfer* can be used without any further training. By contrast, *Sketch2Mesh/Render* requires an image translation network to predict foreground/background masks from sketches. Here, we use the one of [IZZE17] with a UNet [RFB15] as its generator and in the LSGAN setting [MLX$^+$16]. We train four separate instances of it on ShapeNet , one for each shape category (cars and chairs) and for each sketch rendering style (`Suggestive` and `SketchFD`).

This being done, we can compare *Sketch2Mesh/Render* against *Sketch2Mesh/Chamfer* on the test sets for both categories of object and the three categories of drawing we use, `Suggestive`, `SketchFD`, and `Hand Drawn`. We show qualitative results in Figs. 5.5 and 5.6. We report quantitative results in Tab. 5.1 for models trained on `Suggestive` contours. Similar results on `SketchFD` contours are presented in the appendix. Overall, both *Sketch2Mesh/Render* and *Sketch2Mesh/Chamfer* improve the initial metrics but *Sketch2Mesh/Chamfer* appears to be more robust to style changes. In other words, *Sketch2Mesh/Render* overfits to the style it is trained on and does not do as well as *Sketch2Mesh/Chamfer* when tested on a different one. Adding this to the fact that *Sketch2Mesh/Chamfer*, unlike *Sketch2Mesh/Render*, does not require to train an auxiliary network clearly makes it the better approach. We will therefore use it in the remainder of the paper except otherwise noted and will refer to it as *Sketch2Mesh* for brevity.

5.4.4 Comparison against State-of-the-Art Methods

Training Drawing Style: Suggestive

Metric	Method	Test Drawing Style		
		Suggestive	SketchFD	Hand-drawn
CD-$l^2 \cdot 10^3$ ↓	Pix2Vox [DAI+18]	2.336	6.237	8.599
	MeshRCNN [GMJ19]	3.491	6.923	7.849
	DISN [XWC+19]	1.529	7.764	10.396
	Sketch2Mesh	**1.420**	**3.132**	**4.396**
Normal Consistency ↑	Pix2Vox [DAI+18]	89.07	80.49	76.70
	MeshRCNN [GMJ19]	84.19	79.93	77.91
	DISN [XWC+19]	92.15	79.51	72.52
	Sketch2Mesh	**92.20**	**87.00**	**84.74**

Training Drawing Style: SketchFD

CD-$l^2 \cdot 10^3$ ↓	Pix2Vox [GFK+18]	3.529	2.475	3.146
	MeshRCNN [GMJ19]	3.117	3.596	4.829
	DISN [XWC+19]	4.036	1.573	3.763
	Sketch2Mesh	**2.419**	**1.516**	**2.047**
Normal Consistency ↑	Pix2Vox [GFK+18]	87.11	89.21	86.27
	MeshRCNN [GMJ19]	83.22	82.81	80.83
	DISN [XWC+19]	86.34	91.30	87.66
	Sketch2Mesh	**91.23**	**92.09**	**91.03**

Table 5.2 – **Comparative results on Cars:** we compare our method to existing baselines, when trained on 2 styles of sketches: `Suggestive` (top) or `SketchFD` (bottom), and tested on both `Suggestive`, `SketchFD`, and `Hand-drawn`. Our method outperforms others in all scenarios.

We now compare *Sketch2Mesh* against state-of-the-art methods that produce watertight meshes as we do. To this end, we train the architecture of [DAI+18] that regresses volumetric grids from sketches, which we dub *Pix2Vox*. We also compare to recent SVR method that rely on perceptual feature pooling from the image plane DISN [XWC+19] and MeshRCNN [GMJ19]. For a fair comparison, we use them in conjunction with the same image encoder as we do, ResNet18 [HZRS16].

We show qualitative results in Fig. 5.3. We report quantitative results on ShapeNet Cars and Chairs in Tables 5.2 and 5.3 when the latent representation have been learned either on `Suggestive` or `SketchFD` contours. On cars, *Sketch2Mesh* clearly outperforms the other methods. On chairs, MeshRCNN is very competitive, especially in terms of CD-l_2. But, as shown in Fig. 5.7, the meshes it produces are hardly usable, even though we uses the *Pretty* setup of the algorithm that attempts to regularize them. This is a well known phenomenon reported by its authors themselves. By contrast, our meshes can directly be used for downstream applications, without further preprocessing.

For completeness, we note that a very recent paper [ZGZS20] also advocates using foreground/background masks to improve 3D reconstruction from sketches. However, instead of refining the mesh produced by a network using such as a mesh as done by *Sketch2Mesh/Render*, it recommends feeding the mask as an additional input to the network that produces the initial 3D shape. In Tab. 5.4, we compare this approach to ours when the network is trained using the `SketchFD` sketches on cars and tested on `Suggestive`. Both *Sketch2Mesh/Render*

Training Drawing Style: Suggestive

Metric	Method	Test Drawing Style		
		Suggestive	SketchFD	Hand-drawn
CD-$l^2 \cdot 10^3$ ↓	Pix2Vox [DAI+18]	22.953	33.46	62.132
	MeshRCNN [GMJ19]	**6.775**	**10.718**	19.055
	DISN [XWC+19]	7.045	18.104	23.282
	Sketch2Mesh	7.180	12.248	**13.787**
Normal Consistency ↑	Pix2Vox [GFK+18]	73.01	64.28	40.12
	MeshRCNN [GMJ19]	76.91	72.77	58.03
	DISN [XWC+19]	80.44	54.10	51.81
	Sketch2Mesh	**82.61**	**76.27**	**67.67**

Training Drawing Style: SketchFD

CD-$l^2 \cdot 10^3$ ↓	Pix2Vox [GFK+18]	34.759	22.690	46.687
	MeshRCNN [GMJ19]	**9.530**	**5.812**	16.620
	DISN [XWC+19]	13.059	8.628	18.104
	Sketch2Mesh	**9.524**	6.737	**12.585**
Normal Consistency ↑	Pix2Vox [GFK+18]	65.97	72.52	52.96
	MeshRCNN [GMJ19]	77.62	**84.75**	69.76
	DISN [XWC+19]	73.39	80.21	62.58
	Sketch2Mesh	**81.00**	83.10	**70.39**

Table 5.3 – **Comparative results on Chairs:** we compare our method to existing baselines, when trained on 2 styles of sketches: `Suggestive` (top) or `SketchFD` (bottom), and tested on both `Suggestive`, `SketchFD`, and `Hand-drawn`. Our method outperforms others in all scenarios, except some cases where MeshRCNN achieves better metrics. However, as shown in Fig. 5.7, the meshes produced by MeshRCNN are hardly usable.

and *Sketch2Mesh/Chamfer* outperform it.

5.4.5 Interactive 3D editing

An important feature of *Sketch2Mesh* is that is can exploit sketches made of a single stroke to refine previously obtained shapes as discussed in Section 5.3.4, as shown in Fig. 5.8. To showcase the interactivity of our approach we built a web based user interface. The user may draw a sketch with the mouse or a touch enabled device and submit it to *Sketch2Mesh*. Then, successive partial sketches can also be input and matched by the optimizer.

5.5 Conclusion

We have proposed an approach to deriving 3D shapes from sketches that relies on an encoder/decoder architecture to compute a latent surface representation of the sketch. It can in turn be refined to match the external contours of the sketch. It handles sketches drawn in a style it was not specifically trained for and outperforms state-of-the-art methods. Furthermore, it allows for interactive refinements by specifying partial 2D contours the object's projection must match, provided that perspective camera parameters are associated to the sketch. This can be achieved easily on a tablet using a stylus-based interface to draw.

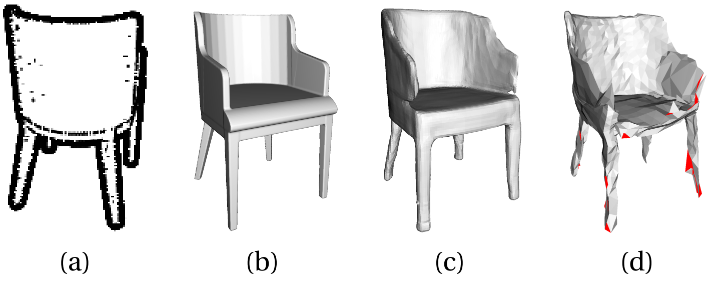


Figure 5.7 – **Comparison with MeshRCNN:** **(a)** Input sketch **(b)** Ground truth shape, **(c)** *Sketch2Mesh* reconstruction, with CD-l_2=1.98, **(d)** MeshRCNN reconstruction, with CD-l_2=1.91. The flipped facets are shown in red. Despite having a slightly higher CD-l_2, our reconstruction is far more usable for further processing and, arguably, resembles the ground truth more than the MeshRCNN one.

Method	Metric	
	CD-$l^2 \cdot 10^3$ ↓	NC ↑
MeshSDF [RLR+20]	3.231	89.67
MeshSDF [RLR+20] + mask	3.124	90.05
Sketch2Mesh/Render	2.538	90.92
Sketch2Mesh/Chamfer	**2.419**	**91.23**

Table 5.4 – **Comparison with the approach of [ZGZS20]:** conditioning the reconstruction network on object silhouettes (+ mask) improves metrics, but our refinement approaches are more effective.

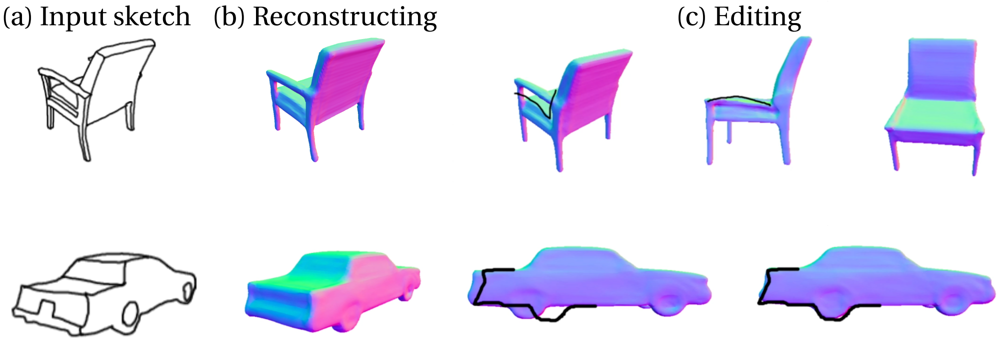


Figure 5.8 – **Interactive reconstruction & editing.** We developed an interface where the user can draw an initial sketch **(a)** to obtain its 3D reconstruction **(b)**. One can then manipulate the object in 3D and draw one or more desired modifications **(c)**. 3D surfaces are then optimized to match each constraint, solving the optimization problem of Section 5.3.4. The strong prior learned by our model allows to preserve global properties such as symmetry despite users provide sparse 2D strokes in input.

We can see two natural improvements to our work. One is linked to the learned priors in our parametrization. Although the priors are usually good at preserving global shape properties such as symmetry, they can be either too constraining or not enough when for partial refinements. We would like some priors to actually be constraints—the wheels of the cars must be round and cannot touch the wheel wells and the feet of the chairs must all have the same length, for example—in addition to those imposed by 2D sketches so that our technique can be turned into a full-fledged tool for Computer Assisted Design. Another research direction would be to incorporate interior lines in our refinement process. This is also an interesting challenge since we don't want to sacrifice the generalization ability this simple technique allowed us to achieve.

6 MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks

This chapter is based on the conference paper [GSF22]:

B. Guillard, F. Stella, and P. Fua, *MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks*, at ECCV, 2022.

6.1 Introduction

In recent years, deep implicit surfaces [PFS$^+$19, MON$^+$19, CZ19] have emerged as a powerful tool to represent and manipulate watertight surfaces. Furthermore, for applications that require an explicit 3D mesh, such as sophisticated rendering including complex physical properties [NDVZJ19] or optimizing physical performance [BRFF18], they can be used to parameterize explicit 3D meshes whose topology can change while preserving differentiability [AHY$^+$19, RLR$^+$20, GRL$^+$22].

However, these approaches can only handle watertight surfaces. Because common 3D datasets such as ShapeNet [CFG$^+$15] contain non-watertight meshes, one needs to preprocess them to create a watertight outer shell [PFS$^+$19, XWC$^+$19]. This is time consuming and ignores potentially useful inner components, such as seats in a car. An alternative is to rely on network initialization or regularization techniques to directly learn from raw data [AL20a, AL20b] but this significantly slows down the training procedure.

This therefore leaves open the problem of modeling non-watertight surfaces implictly. It has been shown in [CMPM20, ZWLS21, VKS$^+$21, CPA$^+$21] that occupancy fields and signed distance functions (SDFs) could be replaced by unsigned ones (UDFs) for this purpose. However, unlike for SDFs, there are no fast algorithms to directly mesh UDFs. Hence, these methods rely on a two-step process that first extracts a dense point cloud that can then be triangulated using slow standard techniques [BMR$^+$99]. Alternatively, non-watertight surfaces can be represented as watertight thin ones surrounding them [CPA$^+$21, GRL$^+$22, VKS$^+$21]. This amounts to meshing the ϵ iso-surface of an UDF using marching cubes [LC87], for ϵ being a small strictly positive scalar. Unfortunately, that degrades reconstruction accuracy because

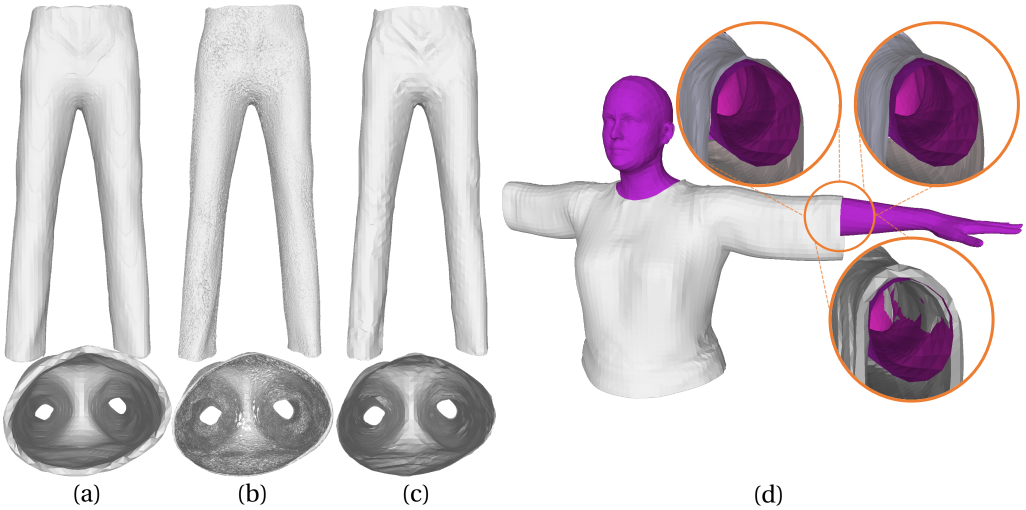


Figure 6.1 – **Meshing the UDF of a garment.** We present front and top views. **(a)** Inflating shapes to turn open surfaces into watertight ones [VKS$^+$21, CPA$^+$21, GRL$^+$22] inherently reduces accuracy by making the surface thicker, as shown in top view. **(b)** Triangulating a cloud of 3D points collapsed on the 0-level set [CMPM20] is time-consuming and tends to produce rough surfaces. **(c)** Directly meshing the UDF using our approach is more accurate and less likely to produce artifacts. In addition, it makes the iso-surface extraction process differentiable. (**d**) We mesh the UDF of a shirt and display it on a human body. The three insets represent the ground truth shirt, the reconstruction with our method, and the inflation approach, respectively. Our approach—in the upper right inset—produces fewer artifacts and no penetrations with the body.

the thin surfaces cannot be infinitely so, as ϵ cannot be arbitrarily small. Furthermore, some applications such as draping simulation [LCT18, TWL$^+$18, GCP$^+$22] require surfaces to be single-faced and cannot be used in conjunction with this approach.

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

In short, our contribution is a new approach to meshing UDFs and parameterizing 3D meshes to model non-watertight surfaces whose topology can change while preserving differentiability, which is something that had only been achieved for watertight surfaces before. We use it in conjunction with a learned shape prior to optimize fitting to partial observations via gradient descent.

We demonstrate it achieves better reconstruction accuracy than current deep-learning based approaches to handling non-watertight surfaces, in a fraction of the computation time, as illustrated by Fig. 6.1.

6.2 Related Work: Triangulating Implicit Non-Watertight Surfaces

Unfortunately, neither the original marching cubes algorithm nor any of its recent improvements are designed to handle non-watertight surfaces. One way around this is to surround the target surface with a thin watertight one [CPA$^+$21, GRL$^+$22, VKS$^+$21], as shown in Fig. 6.1(a). One can think of the process as *inflating* a watertight surface around the original one. Marching cubes can then be used to triangulate the inflated surface, but the result will be some distance away from the target surface, resulting in a loss of accuracy. Another approach is to sample the minimum level set of an UDF field, as in NDF [CMPM20] and AnchorUDF [ZWLS21]. This is done by projecting randomly initialized points on the surface using gradient descent. To ensure full coverage, points are re-sampled and perturbed during the process. This produces a cloud, but not a triangulated mesh with information about the connectivity of neighboring points. Then the ball-pivoting method [BMR$^+$99], which connects neighboring points one triplet at a time, is used to mesh the cloud, as shown in Fig. 6.1(b). It is slow and inherently sequential.

6.3 Method

We now present our core contribution, a fast and differentiable approach to extracting triangulated isosurfaces from unsigned distance fields produced by a neural network. Let us consider a network that implements a function

$$\begin{aligned} \phi : \mathbb{R}^C \times \mathbb{R}^3 &\rightarrow \mathbb{R}^+ , \\ \mathbf{z}, \mathbf{x} &\mapsto s , \end{aligned} \tag{6.1}$$

where $\mathbf{z} \in \mathbb{R}^C$ is a parameter vector; $\mathbf{x}$ is a 3D point; s is the Euclidean distance to a surface. Depending on the application, $\mathbf{z}$ can either represent only a latent code that parameterizes the surface or be the concatenation of such a code and the network parameters.

In Sec. 6.3.1, we propose an approach to creating a triangulated mesh $M = (V, F)$ with vertices V and facets F from the 0-level set of the scalar field $\phi(\mathbf{z}, \cdot)$. Note that it could also apply to non-learned UDFs, as shown in the appendix. In Sec. 6.3.2, we show how to make the vertex coordinates differentiable with respect to $\mathbf{z}$. This allows refinement of shape codes or network parameters with losses directly defined on the mesh.

6.3.1 From UDF to Triangulated Mesh

Surface Detection within Cells

As in standard marching cubes [LC87], we first sample a discrete regular grid G in the region of interest, typically $[-1,1]^3$. At each location $\mathbf{x}_i \in G$ we compute

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

where u_i is the unsigned distance to the implicit surface at location $\mathbf{x}_i$, and $\mathbf{g}_i \in \mathbb{R}^3$ is the gradient computed using backpropagation.

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

$$s_i = \mathrm{sgn}(\mathbf{g}_1 \cdot \mathbf{g}_i)\, u_i\ , \tag{6.2}$$

where $\mathbf{x}_1$ is one of the cell corners that we refer to as the *anchor.* $\mathbf{x}_1$ is assigned a positive pseudo-signed distance and corners where the gradient direction is opposite to that at $\mathbf{x}_1$ a negative one. When there is at least one negative s_i, we use marching cubes' disjunction cases and vertex interpolation to reconstruct a triangulated surface in the cell.

Computing pseudo-signs in this way is simple but has two shortcomings. First, it treats each cell independently, which may cause inconsistencies in the reconstructed facets orientations. Second, especially when using learned UDF fields that can be noisy [VKS+21], the above smoothness assumption may not hold within the cells. This typically results in holes in the reconstructed meshes.

To mitigate the first problem, our algorithm starts by exploring the 3D grid until it finds a cell with at least one negative pseudo-sign. It then uses it as the starting point for a breadth-first exploration of the surface. Values computed at any cell corner are stored and never recomputed, which ensures that the normal directions and interpolated vertices are consistent in adjacent cells. The process is repeated to find other non-connected surfaces, if any. To mitigate the second problem we developed a more sophisticated method to assign a sign to each cell corner. We do so as described above for the root cell of our breadth-first search, but we use the voting scheme depicted by Fig. 6.2 for the subsequent ones. Voting is used to aggregate information from neighboring nodes to estimate pseudo-sign more robustly. Each corner $\mathbf{x}_i$ of a cell under consideration receives votes from all adjacent grid points $\mathbf{x}_k$ that have already been assigned a pseudo-sign, based on the relative directions of their gradients and the pseudo-sign of $\mathbf{x}_k$. Since gradients locally point towards the greatest ascent direction, if

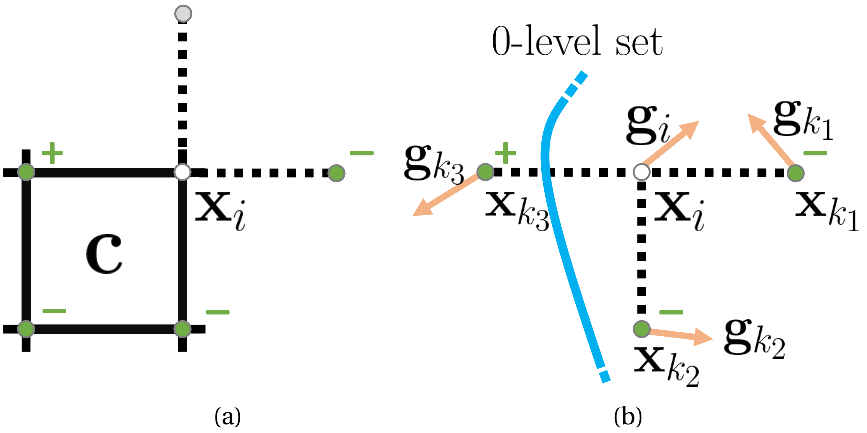


Figure 6.2 – **Voting. (a)** Corner $\mathbf{x}_i$ of cell **c** has 3 neighbors that already have a pseudo-sign and vote. **(b)** The projections of $\mathbf{g}_i$ and $\mathbf{g}_{k_1}$ on the edge connecting the two neighbors face each other. Thus $\mathbf{x}_{k_1}$ votes for $\mathbf{x}_i$ having the same sign as itself (-). The other two neighbors vote for − as well given the result of computing Eq. 6.3.

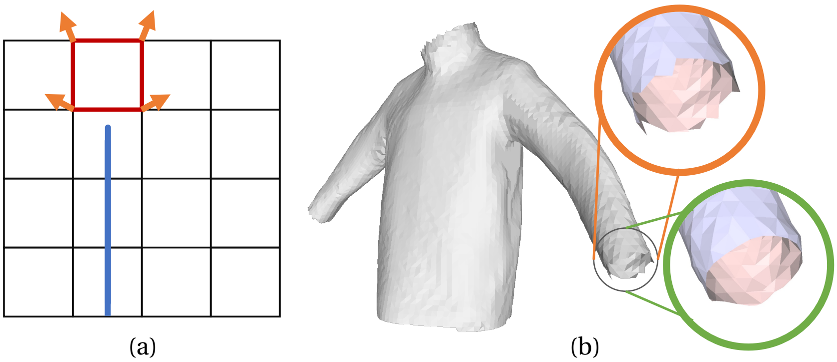


Figure 6.3 – **Removing artifacts. (a)** Given the blue 0-level surface, the red cell has gradients in opposing directions and yields an undesirable face. We prune these by evaluating the UDF on reconstructed faces. **(b)** Initially reconstructed borders are uneven (top). We smooth them during post-processing (bottom).

the projections of $\mathbf{g}_i$ and $\mathbf{g}_k$ along the edge connecting $\mathbf{x}_i$ and $\mathbf{x}_k$ face each other, there is no surface between them and the vote is in favor of them having the same sign: $v_{ik} = \text{sgn}(s_k)$. Otherwise, the vote depends on gradient directions, and we take it to be

$$v_{ik} = (\mathbf{g}_i \cdot \mathbf{g}_k)\text{sgn}(s_k) \tag{6.3}$$

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

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

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

Global Surface Triangulation

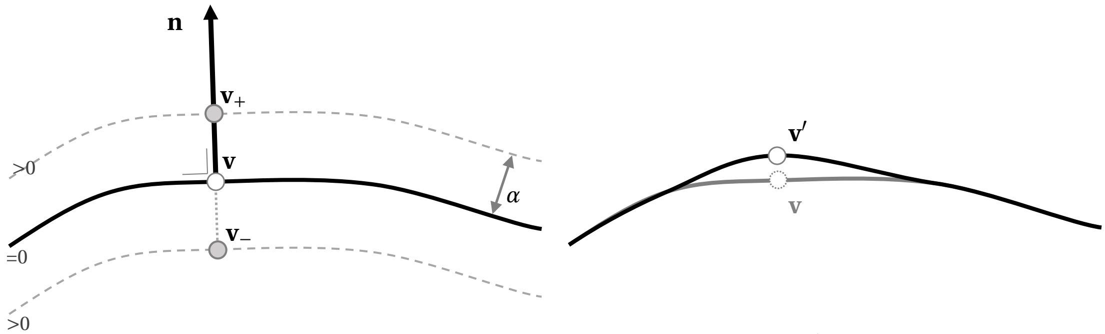
The facets that the above approach yields are experimentally consistent almost everywhere, except for a few them, which we describe below and can easily remove in a post-processing stage. Note that the gradients we derive in Sec. 6.3.2 do not require backpropagation through the iso-surface extraction. Hence, this post-processing step does not compromise differentiability.

Removing Spurious Facets. As shown in Fig. 6.3 **(a)**, facets that do not correspond to any part of the surfaces can be created in cells with gradients pointing in opposite directions without intersecting the 0-level set. This typically happens near surface borders because our approach tends to slightly extend them, or around areas with poorly approximated gradients far from the surface in the case of learned UDF fields. Such facets can be detected by re-evaluating the distance field on all vertices. If the distance field for one vertex of a face is greater than half the side-length of a cubic cell, it is then eliminated.

Smoothing Borders. Since marching cubes was designed to reconstruct watertight surfaces, it cannot handle surface borders. As a result, they appear slightly jagged on initial reconstructions. To mitigate this, we apply laplacian smoothing on the edges belonging to a single triangle. This smoothes borders and qualitatively improves reconstructions, as shown in Fig. 6.3 **(b)**.

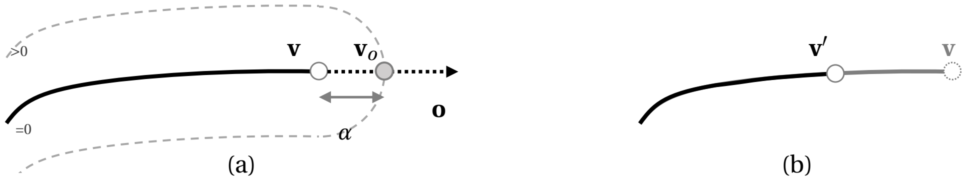
6.3.2 Differentiating through Iso-Surface Extraction

Let $\mathbf{v} \in \mathbb{R}^3$ be a point on a facet reconstructed using the method of Sec. 6.3.1. Even though differentiating $\mathbf{v}$ directly through marching cubes is not possible [LDG18, RLR$^+$20], it was shown that if ϕ were an SDF instead of an UDF, derivatives could be obtained by reasoning about surface inflation and deflation [AHY$^+$19, RLR$^+$20]. Unfortunately, for an UDF, there is no "in" or "out" and its derivative is undefined on the surface itself. Hence, this method does not directly apply. Here we extend it so that it does, first for points strictly within the surface, and then for points along its boundary.



(a) (b)

Figure 6.4 – **Isosurface deformation**: **(a)** with $\mathbf{v}$ on the 0-level set, we surround it with $\mathbf{v}_+$ and $\mathbf{v}_-$ at a distance α ; **(b)** In case the UDF decreases at $\mathbf{v}_+$ and increases at $\mathbf{v}_-$, $\mathbf{v}$ moves to $\mathbf{v}'$.



(a) (b)

Figure 6.5 – **Isosurface shrinkage or extension**: **(a)** with $\mathbf{v}$ on the border of the 0-level set, we place $\mathbf{v}_o$ at a distance α in the direction of $\mathbf{o}$; **(b)** If the UDF increases at $\mathbf{v}_o$, $\mathbf{v}$ moves to $\mathbf{v}'$.

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

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

the two closest points to $\mathbf{v}$ on the α-level set on both sides of the 0-level set. For α small enough, the outward oriented normals at these two points are close to being $\mathbf{n}$ and $-\mathbf{n}$. We can therefore use the formulation of [AHY$^+$19, RLR$^+$20] to write

$$\frac{\partial \mathbf{v}_+}{\partial \mathbf{z}} \approx -\mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_+) \quad \text{and} \quad \frac{\partial \mathbf{v}_-}{\partial \mathbf{z}} \approx \mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_-)\,. \tag{6.4}$$

Since $\mathbf{v} = \frac{1}{2}(\mathbf{v}_- + \mathbf{v}_+)$, Eq. 6.4, yields

$$\frac{\partial \mathbf{v}}{\partial \mathbf{z}} \approx \frac{\mathbf{n}}{2}\left[\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} - \alpha\mathbf{n}) - \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} + \alpha\mathbf{n})\right]\,. \tag{6.5}$$

We provide a more formal proof and discuss the validity of the approximation in appendix. Note that using $\mathbf{n}' = -\mathbf{n}$ instead of $\mathbf{n}$ yields the same result. Intuitively, this amounts to surrounding the 0-level set with α-margins where UDF values can be increased on one side and decreased on the other, which allows local deformations perpendicular to the surface. Fig. 6.4 **(a)** depicts the arrangement of $\mathbf{v}$, $\mathbf{v}_+$ and $\mathbf{v}_-$ around the surface. The derivative of Eq. 6.5 implies that infinitesimally increasing the UDF value at $\mathbf{v}_-$ and decreasing it at $\mathbf{v}_+$ would push $\mathbf{v}$ in the direction of $\mathbf{n}$, as shown in Fig. 6.4 **(b)**, and conversely. In practice, we use

$\alpha = 10^{-2}$ in all our experiments.

Derivatives at the Surface Boundaries. Let us now assume that $\mathbf{v}$ sits on the edge of a boundary facet. Mapping it to $\mathbf{v}_+$ and $\mathbf{v}_-$ and using the derivatives of Eq. 6.5 would mean that all deformations are perpendicular to that facet. Thus, it does not permit shrinking or expanding of the surface during shape optimization. In this setting, there is a whole family of closest points to $\mathbf{v}$ in the α-level set; they lay on a semicircle with radius α. To allow for shrinkage and expansion, we map $\mathbf{v}$ to the semicircle point along $\mathbf{o}$, a vector perpendicular to the edge, pointing outwards, and within the plane defined by the facet. Hence, we consider the point $\mathbf{v}_o$ which is the closest to $\mathbf{v}$ on the α-level set in the direction of $\mathbf{o}$:

$$\mathbf{v}_o = \mathbf{v} + \alpha \mathbf{o} \tag{6.6}$$

For α small enough, the outward oriented normal at $\mathbf{v}_o$ is $\mathbf{o}$ and we again use the formulation of [AHY+19, RLR+20] and Eq. 6.6 to write

$$\frac{\partial \mathbf{v}_o}{\partial \mathbf{z}} = -\mathbf{o} \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v}_o) \text{ and } \frac{\partial \mathbf{v}}{\partial \mathbf{z}} = -\mathbf{o} \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} + \alpha \mathbf{o}) , \tag{6.7}$$

which we use for all points $\mathbf{v}$ on border edges. As shown in Fig. 6.5, this implies that increasing the UDF value at $\mathbf{v}_o$ would push $\mathbf{v}$ inwards and make the surface shrink. Conversely, decreasing it extends the surface in the direction of $\mathbf{o}$.

6.4 Experiments

We demonstrate our ability to mesh UDFs created by deep neural networks. To this end, we first train a deep network to map latent vectors to UDFs representing different garments, that is, complex open surfaces with many different topologies. We then show that, given this network, our approach can be used to effectively triangulate these garments and to model previously unseen ones. Finally, we plug our triangulation scheme into existing UDF networks and show that it is a straightforward operation. Finally, the benefit of the border gradients of Sec. 6.3.2 is evaluated. The voting scheme proposed in Sec. 6.3.1 is ablated in appendix.

6.4.1 Network and Metrics

Our approach is designed to triangulate the output of networks that have been trained to produce UDF fields. To demonstrate this, we use an auto-encoding approach [PFS+19] with direct supervision on UDF samples on the MGN dataset [BTTPM19] to train a network ϕ_θ that maps latent vectors of dimension 128 to UDFs that represent garments. These UDFs can in turn be triangulated using our algorithm to produce meshes such as those of Fig. 6.1. We provide details of this training procedure in the appendix. The MGN dataset comprises 328 meshes. We use 300 to train ϕ_θ and the remaining 28 for testing.

	Garments, ϕ_θ network			**Cars, NDF network [CMPM20]**		
	BP	*Inflation*	*Ours*	*BP*	*Inflation*	*Ours*
CHD (↓)	1.62	3.00	**1.51**	6.84	11.24	**6.63**
IC (%, ↑)	92.51	88.48	**92.80**	90.50	87.09	**90.87**
NC (%, ↑)	89.50	94.16	**95.50**	61.50	**73.19**	70.38
Time (↓)	16.5s + 3000s	**1.0 sec.**	1.2 sec.	24.7s + 8400s	**4.8 sec.**	7.1 sec.

Table 6.1 – **Comparing UDF meshing methods.** Average Chamfer (CHD), image consistency (IC), normal consistency (NC) and processing time for 300 garments (left) and 300 ShapeNet cars (right). We use a single UDF network in each case and only change the meshing procedure. For BP, we decompose the time into sampling and meshing times.

For comparison purposes, we also use the publicly available pre-trained network of NDF [CMPM20] that regresses UDF from sparse input point clouds. It was trained on raw ShapeNet [CFG$^+$15] meshes, without pre-processing to remove inner components make them watertight or consistently orient facets.

To compare the meshes we obtain to the ground-truth ones, we evaluate the following three metrics:

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

We also describe them in more detail in the appendix.

6.4.2 Mesh Quality and Triangulation Speed

Fig. 6.1 was created by triangulating a UDF produced by ϕ_θ using either our meshing procedure (*Ours*) or one of two baselines:

- *BP*. It applies the ball-pivoting method [BMR$^+$99] implemented in [CCC$^+$08] on a dense surface sampling of $900k$ points, as originally proposed in [CMPM20] and also used in [ZWLS21]. Surface points are obtained by gradient descent on the UDF field.
- *Inflation* [GRL$^+$22, VKS$^+$21, CPA$^+$21]. It uses standard marching cubes to mesh the ϵ-isolevel of the field, with $\epsilon > 0$.

In Tab. 6.1 (left), we report metrics on the 300 UDF fields $\phi_\theta(\mathbf{z}_i, \cdot)$ for which we have latent codes resulting from the above-mentioned training. *Inflation* and *Ours* both use a grid size of 128^3 over the $[-1,1]^3$ bounding box, and we set *Inflation*'s ϵ to be 55% of marching cubes'

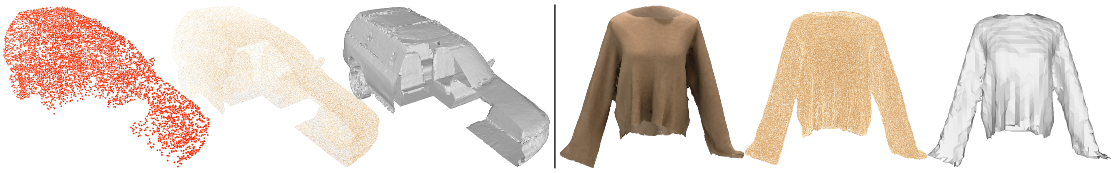


Figure 6.6 – **Using our approach to triangulate the outputs of NDF [CMPM20] (left) and AnchorUDF [ZWLS21] (right).** In both cases, we display the input to the network (a point cloud in one case and a color image in the other), the dense cloud of points that is the final output of these methods, and a triangulation of the UDF they compute generated using our method.

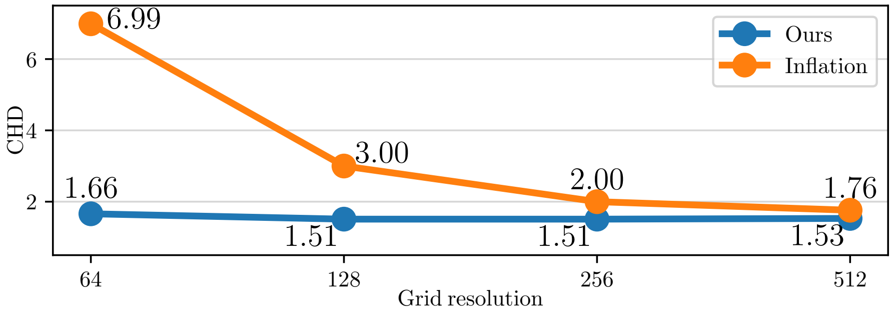


Figure 6.7 – **CHD as a function of grid resolution.** between reconstructed and ground truth meshes, averaged over the 300 training garments of MGN. *Ours* yields constantly accurate meshes, *Inflation* deforms the shapes at low resolutions.

step size. In Tab. 6.1 (right) we also report metrics for the pretrained NDF network [CMPM20] tested on 300 ShapeNet cars, in which case we increase *Inflation* and *Ours* resolution to 192^3 to account for more detailed shapes. An example is shown in Fig. 6.6. The experiments were run on a NVidia V100 GPU with an Intel Xeon 6240 CPU.

As shown on the left of Table. 6.1, *Ours* is slightly more accurate than *NDF* in terms of all three metrics, while being orders of magnitude faster. *Inflation* is even faster—this reflects the overhead our modified marching cube algorithm imposes—but far less accurate. To show that this result is not specific to garments, we repeated the same experiment on 300 cars from the ShapeNet dataset and report the results on the right side of Table. 6.1. The pattern is the same except for NC, which is slightly better for *Inflation*. We conjecture this to be a byproduct of the smoothing provided by *Inflation*, which is clearly visible in Fig. 6.1**(a,c)**.

To demonstrate that these results do not depend on the specific marching cube grid resolution we chose, we repeated the experiment for grid resolutions ranging from 64 to 512 and plot the average CHD as a function of resolution in Fig. 6.7. It remains stable over the whole range.

For comparison purposes, we also repeated the experiment with *Inflation*. Each time we increase the resolution, we take the ϵ value that defines the iso-surface to be triangulated to be 10% greater than half the grid-size, as shown in Fig. 6.8. At very high resolution, the accuracy of

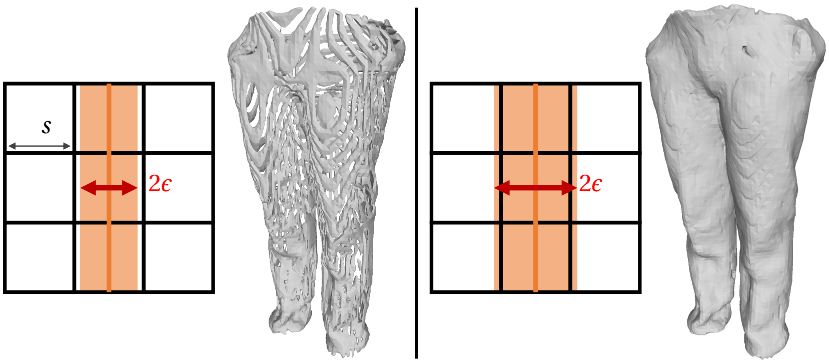


Figure 6.8 – **Choosing ϵ for *Inflation*.** when meshing a UDF's ϵ iso-level with standard marching cubes, the value of ϵ is lower bounded by half the step size s. **Left**: $2\epsilon < s$ yields many large holes. **Right**: $2\epsilon \geq s$ yields a watertight mesh.

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

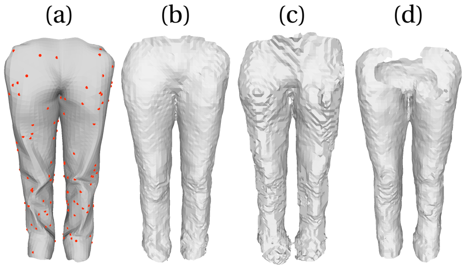


Table 6.2 – **Fitting to sparse point clouds.** The table shows average Chamfer (CHD), image consistency (IC), and normal consistency (NC) wrt. ground truth test garments. We report metrics for un-optimized latent codes (*Init.*), after optimizing ($\mathscr{L}_{PC,mesh}$) using our method, and optimizing either $\mathscr{L}_{PC,UDF}$ or $\widetilde{\mathscr{L}}_{PC,UDF}$ in the implicit domain. **(a)** A sparsely sampled ground truth mesh. **(b)** Mesh reconstructed by mimimizing $\mathscr{L}_{PC,mesh}$, **(c)** $\mathscr{L}_{PC,UDF}$, **(d)** $\widetilde{\mathscr{L}}_{PC,UDF}$.

Inflation approaches *Ours* but that also comes at a high-computational cost because operating on 512 × 512 × 512 cubes instead of 128 × 128 × 128 ones is much slower, even when using multi-resolution techniques.

6.4.3 Using Differentiability to Fit Sparse Data

Given the trained network ϕ_θ and latent codes for training shapes from Sec. 6.4.2, we now turn to recovering latent codes for the remaining 28 test garments. For each test garment G_j, given the UDF representing it, this would be a simple matter of minimizing the mean square error between it and the field $\phi_\theta(\mathbf{z}, \cdot)$ with respect to $\mathbf{z}$, which does not require triangulating. We therefore consider the more challenging and more realistic cases where we are only given eiter small set of 3D points P_j—in practice we use 200 points—or silhouettes and have to find a latent vector that generates the UDF that best approximates them.

Fitting to 3D points.

One way to do this is to remain in the implicit domain and to minimize one of the two loss functions

$$\begin{aligned}\mathscr{L}_{PC,UDF}(P_j,\mathbf{z}) &= \frac{1}{|P_j|}\sum_{p\in P_j}|\phi_\theta(\mathbf{z},p)|\,,\\ \widetilde{\mathscr{L}}_{PC,UDF}(P_j,\mathbf{z}) &= \mathscr{L}_{PC,UDF}(P_j,\mathbf{z}) + \frac{1}{|A|}\sum_{a\in A}|\phi_\theta(\mathbf{z},a) - \min_{p\in P_j}\|a-p\|_2|\,,\end{aligned} \tag{6.8}$$

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

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

$$\mathscr{L}_{PC,mesh}(P_j,\mathbf{z}) = \frac{1}{|P_j|}\sum_{p\in P_j}\min_{a\in M_\mathbf{z}}\|a-p\|_2\,, \tag{6.9}$$

where $a \in M_\mathbf{z}$ means sampling 10k points a on the triangulate surface of $M_\mathbf{z}$. Minimizing $\mathscr{L}_{PC,mesh}$ means that the chamfer distance between the triangulated surfaces and the sample points should be small. Crucially, the results of Sec. 6.3.2 guarantee that $\mathscr{L}_{PC,mesh}$ is differentiable with respect to $\mathbf{z}$, which makes minimization practical.

We tried minimizing the three loss functions defined above. In each case we started the minimization from a randomly chosen latent vector for a garment of the same type as the one we are trying to model, which corresponds to a realistic scenario if the initial estimate is provided by an upstream network. We report our results in Tab. 6.2. Minimizing $\mathscr{L}_{PC,mesh}$ clearly yields the best results, which highlights the usefulness of being able to triangulate and to differentiate the result.

Fitting to Silhouettes.

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

$$\mathscr{L}_{silh,mesh}(S_j,\mathbf{z}) = L_1(rend(M_\mathbf{z}),S_j)\,, \tag{6.10}$$

where $rend$ is a differentiable renderer [KUH18] that produces a binary image of the UDF triangulation $M_\mathbf{z}$ and $L_1(\cdot)$ is the L_1 distance. Once again, the differentiability of $M_\mathbf{z}$ with respect to $\mathbf{z}$ is key to making this minimization practical.

In theory, instead of rendering a triangulation, we could have used an UDF differential renderer. Unfortunately, we are not aware of any. Approaches such as that of [LZP+20] rely on finding

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

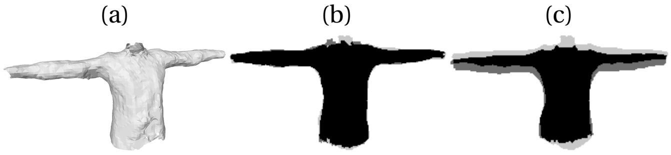


Table 6.3 – **Fitting to silhouettes.** Average Chamfer (CHD), image consistency (IC), and normal consistency (NC) wrt. ground truth test garments. We report metrics for un-optimized latent codes (*Init.*), using our method to minimize ($\mathscr{L}_{silh,mesh}$), and by minimizing ($\mathscr{L}_{silh,UDF}$) in the implicit domain. **(a)** Mesh reconstructed by minimizing $\mathscr{L}_{silh,mesh}$. **(b,c)** Superposition of a target silhouette (light gray) and of the reconstructions (dark gray) by minimizing $\mathscr{L}_{silh,UDF}$ or $\mathscr{L}_{silh,mesh}$. Black denotes perfect alignment and shows that the $\mathscr{L}_{silh,UDF}$ mesh is much better aligned.

sign changes and only work with SDFs. In contrast, CSP-Net [VKS+21] can render UDFs without meshing them but is not differentiable.

To provide a baseline, we re-implemented SMPLicit's strategy [CPA+21] for fitting a binary silhouette by directly supervising UDF values. We sample a set of points $P \subset [-1,1]^3$, and project each $p \in P$ to S_j using the front-facing camera $\mathbf{c}$ to get its projected value s_p. If $s_p = 1$, point p falls within the target silhouette, otherwise it falls into the background. SMPLicit's authors advocate optimizing $\mathbf{z}$ by summing

$$\mathscr{L}_{silh,UDF}(S_j,\mathbf{z}) = \begin{cases} |\phi_\theta(\mathbf{z},p) - d_{max}| & \text{if } s_p = 0 \\ \min\limits_{\bar{p}\text{ s.t. }\mathbf{c}(\bar{p})=\mathbf{c}(p)} |\phi_\theta(\mathbf{z},\bar{p})| & \text{if } s_p = 1 \end{cases} . \tag{6.11}$$

on $p \in P$. That is, points projecting outside the silhouette ($s_p = 0$) should have a UDF value equal to the clamping value d_{max}. For points projecting inside the silhouette, along a camera ray we only consider $\bar{p}$, the closest point to the current garment surface estimate and its predicted UDF value should be close to 0.

We report our results in Tab. 6.3. Minimizing $\mathscr{L}_{silh,mesh}$ yields the best results, which highlights the benefits of pairing our method with a differentiable mesh renderer.

Ablation Study.

We re-ran the optimizations without the border derivative term of Eq. 6.7, that is, by computing the derivatives everywhere using the expression of Eq. 6.5. As can be seen in Tab. 6.4, this reduces performance and confirms the importance of allowing for shrinkage and expansion of the garments.

6.4.4 Differentiable Topology Change

A key feature of all implicit surface representations is that they can represent surfaces whose topology can change. As shown in Fig. 6.9, our approach allows us to take advantage of this

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

Table 6.4 – **Ablation Study.** Average Chamfer (CHD), image consistency (IC), and normal consistency (NC) for test garments using either our full approach to computing gradients (*normals* + *border*) vs. computing the gradients everywhere using only the formula of Eq. 6.5. (*normals*).

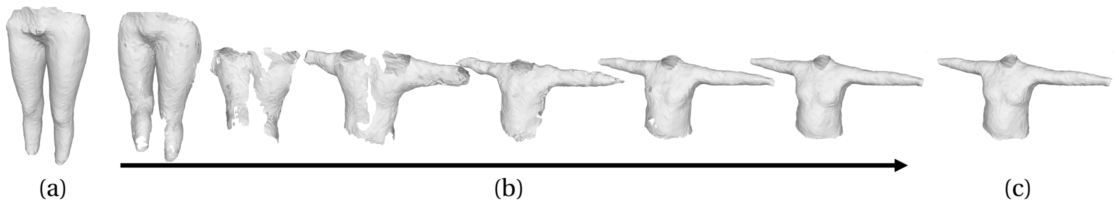


Figure 6.9 – **Optimization with a change in topology:** **(a)** Starting mesh associated to the initial latent code $\mathbf{z} = \mathbf{z}_{start}$; **(b)** Optimizing $\mathbf{z}$ with gradient descent by applying a 3D Chamfer loss between the reconstructed mesh and a target shape shown in **(c)**. During optimization, the latent code takes values that do not correspond to valid garments, hence the tears in our triangulations. Nevertheless, it eventually converges to the desired shape.

while simultaneously creating a mesh whose vertices have associated spatial derivatives. To create this example, we started from a latent code for a pair of pants and optimized with respect to it to create a new surface that approximates a sweater by minimizing the CHD loss of Eq. 6.9 over 10k 3D points on that sweater. The topology changes that occur on the mesh representing the deforming shape do not create any difficulties.

6.4.5 Generalization to other UDF Networks

To show that our meshing procedure is applicable as-is to other UDF-based methods, we use it downstream of publicly available pre-trained networks. In Fig. 6.6 (bottom) we mesh the outputs of the garment reconstruction network of AnchorUDF [ZWLS21]. In Fig. 6.6 (top) we apply it to the point cloud completion pipeline of NDF [CMPM20]. Both these methods output dense point clouds surface, which must then be meshed using the time-costly ball pivoting algorithm. Instead, our method can directly mesh the UDF and does so in a fraction of the time while preserving differentiability. That makes the whole algorithm suitable for inclusion into an end-to-end differentiable pipeline.

6.4.6 Limitations

Reliance on learned UDF fields. The proposed method can mesh the zero-surface of an unsigned distance field. In practice however, UDF fields are approximated with neural networks, and we find it difficult to learn a sharp 0-valued surface for networks with small capacities. It can for example happen that the approximate UDF field is not reaching zero, or that the zero surface thickens and becomes a volume, or that the gradients are not approximated well enough. In such cases, artifacts such as single-cell holes can appear when using our method at a high resolution. Note that applying our method to a real UDF would not exhibit such issues. By comparison however, applying marching cubes on an approximate and poorly learned SDF is more robust since it only requires the field to be continuous and to have a zero crossing to produce artifact-free surfaces. UDF networks could be made more accurate by using additional loss terms [GYH$^+$20] or an adaptive training procedure [DZW$^+$20], but this research direction is orthogonal to the method proposed in this paper.

Training limitations. Similarly to [RLR$^+$20] for SDFs, since the proposed gradients rely on the field being an UDF, they cannot be used to train a neural network from scratch. This would require network initialization or regularization strategies to ensure it regresses valid UDF fields, a topic we see as an interesting research direction.

Limitations of marching cubes. After locally detecting surface crossings via the pseudo-sign computation, we rely on standard marching cubes for meshing an open surface, which implies the need of a high resolution grid to detect high frequency details, and cubic scalability over grid resolution. Moreover, marching cubes was designed to handle watertight surfaces, and as a consequence some topological cases are missing, for example at surface borders or intersections. This could be remedied by detecting and handling such new cases with additional disjunctions.

Normal orientations. The breadth-first exploration of the surface makes the orientation of adjacent facets consistent with each other. However, non-orientable surfaces such as Möbius-strips would intrisically produce juncture points with inconsistent orientations when two different branches of the exploration reach each other. In such points, our method can produce holes. Similarly, marching cubes has geometric guarantees on the topology of reconstructed meshes, but this is not true for the proposed method since there is no concept of *inside* and *outside* in UDFs.

Sequential processing. Our strategy for obtaining consistent facet orientations and avoiding undesirable holes in the surface relies on a procedure that is sequential by nature. As they stand, the breadth-first exploration coupled with the voting mechanism are not amenable to parallelization. This limitation constrains the acceleration and optimization of our algorithm.

6.5 Conclusion

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

7 DrapeNet: Garment Generation and Self-Supervised Draping

This chapter is based on the conference paper [LLG^{+}23]:

L. De Luigi*, R. Li*, B. Guillard, M. Salzmann, and P. Fua, *DrapeNet: Generating Garments and Draping them with Self-Supervision*, at CVPR, 2023.

7.1 Introduction

Draping digital garments over differently-shaped bodies in random poses has been extensively studied due to its many applications such as fashion design, moviemaking, video gaming, virtual try-on and, nowadays, virtual and augmented reality. Physics-based simulation (PBS) [BW98, LBK17, P^{+}95, Pro97, TTN^{+}13, VSC01, Zel05, Nvi18a, Sof18, Nvi18b, Des18, SZZ^{+}18] can produce outstanding results, but at a high computational cost.

Recent years have witnessed the emergence of deep neural networks aiming to achieve the quality of PBS draping while being much faster, easily differentiable, and offering new speed vs. accuracy tradeoffs [GCS^{+}19, GCP^{+}22, MYR^{+}20, PLPM20, SOC19, SLL20, TBTPM20, VSGC20, WCPM18]. These networks are often trained to produce garments that resemble ground-truth ones. While effective, this requires building training datasets, consisting of ground-truth meshes obtained either from computationally expensive simulations [NSO12] or using complex 3D scanning setups [PMPHB17]. Moreover, to generalize to unseen garments and poses, these supervised approaches require training databases encompassing a great variety of samples depicting many combinations of garments, bodies and poses.

The recent PBNS and SNUG approaches [SOC22, BME21] address this by casting the physical models adopted in PBS into constraints used for self-supervision of deep learning models. This makes it possible to train the network on a multitude of body shapes and poses without ground-truth draped garments. Instead, the predicted garments are constrained to obey physics-based rules. However, both PBNS and SNUG, require training a separate network for each garment. They rely on mesh templates for garment representation and feature one output per mesh vertex. Thus, they cannot handle meshes with different topologies, even for

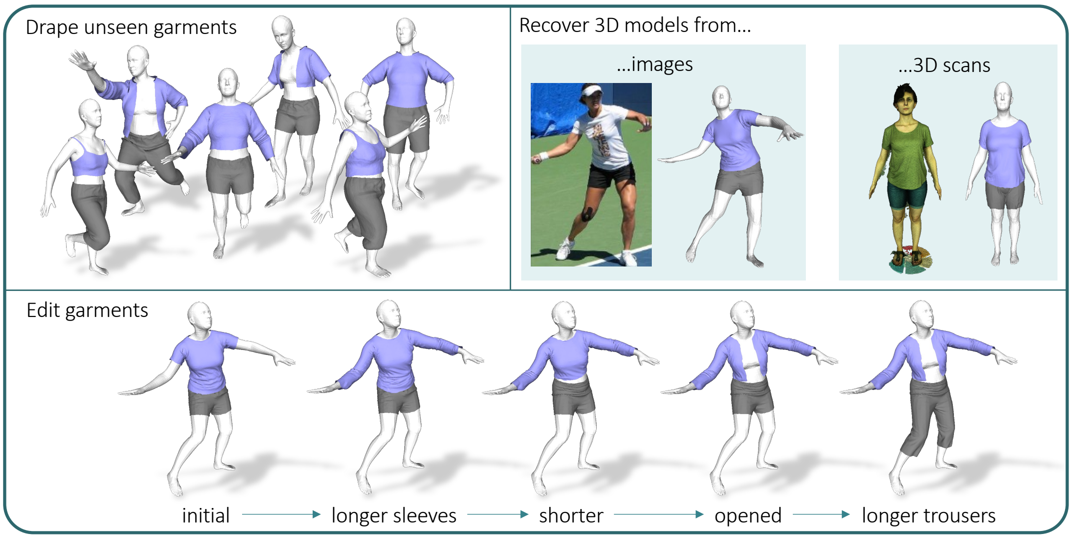


Figure 7.1 – **Overview of DrapeNet.** Our network can drape garments over bodies of different shapes in various poses. To minimize the required amount of supervision, our draping network is trained with physics-based self-supervision and generalizes to multiple garments by being conditioned on latent codes. These can be manipulated to edit specific features of the corresponding garments. Being fully differentiable, our pipeline makes it possible to recover 3D models of garments and bodies from observations such as images and 3D scans.

the same garment. This makes them very specialized and limits their applicability to large garment collections as a new network must be trained for each new clothing item.

In this work, we introduce `DrapeNet`, an approach that also relies on physics-based constraints to provide self-supervision but can handle generic garments by conditioning a *single* draping network with a latent code describing the garment to be draped. We achieve this by coupling the draping network with a garment *generative* network, composed of an encoder and a decoder. The encoder is trained to compress input garments into compact latent codes that are used as input condition for the draping network. The decoder, instead, reconstructs a 3D garment model from its latent code, thus allowing us to sample and edit new garments from the learned latent space.

Specifically, we model the output of the garment decoder as an unsigned distance function (UDF), which were demonstrated [GSF22] to yield better accuracy and fewer interpenetrations than the inflated signed distance functions often used for this purpose [CPA⁺21, LGRF22]. Moreover, UDFs can be triangulated in a differentiable way [GSF22] to produce explicit surfaces that can easily be post-processed, making our pipeline fully differentiable. Hence, `DrapeNet` can not only drape garments over given body shapes but can also perform gradient-based optimization to fit garments, along with body shapes and poses, to partial observations of clothed people, such as images or 3D scans.

Our contributions are as follows:

- We introduce a *single* garment draping network conditioned on a latent code to handle generic garments from a large collection (e.g. *top* or *bottom* garments);
- By exploiting physics-based self-supervision, our pipeline only requires a few hundred garment meshes in a canonical pose for training;
- Our framework enables the fast draping of new garments with high fidelity, as well as the sampling and editing of new garments from the learned latent space;
- Being fully differentiable, our method can be used to recover accurate 3D models of clothed people from images and 3D scans.

7.2 Related Work: Draping Garments over 3D Bodies

Two main classes of methods coexist, physics-based algorithms [BW98, LHT$^+$21, LLK19, NSO12, NPO13, SZZ$^+$18] that produce high-quality drapings but at a high computational cost, and data-driven approaches that are faster but often at the cost of realism.

Among the latter, template-based approaches [BME21, BTTPM19, JZH$^+$20, PLPM20, STOC21, SOC22, TBTPM20, PMJ$^+$22] are dominant. Each garment is modeled by a specific triangulated mesh and a draping function is learned for each one. In other words, they do not generalize. There are however a number of exceptions. In [GCP$^+$22, BMTE21] the mesh is replaced by 3D point clouds that can represent generic garments. This enables deforming garments with arbitrary topology and geometric complexity, by estimating the deformation separately for each point. [ZMGL21] goes further and allows differentiable changes in garment topology by sampling a fixed number of points from the body mesh. Unfortunately, this point cloud representation severely limits possible downstream applications.

In recent approaches [CPA$^+$21, LGRF22], a space of garments is learned with clothing items modeled as inflated SDFs and one single shared network to predict their deformations as a 3D displacement field. This makes deployment in real-world scenarios easier and allows the reconstruction of garments from images and 3D scans. However, the inflated SDF scheme reduces realism and precludes post-processing using standard physics-based simulators or other cloth-specific downstream applications. Furthermore, both models are fully supervised and require a dataset of draped garments whose collection is extremely time-consuming.

Alleviating the need for costly ground-truth draped garments is tackled in [SOC22, BME21], by introducing physics-based losses to train draping networks in a self-supervised manner. The approach of [SOC22] relies on a mass spring model to enforce the physical consistency of static garments deformed by different body poses. The method of [BME21] also accounts for variable body shapes and dynamic effects; furthermore, it incorporates a more realistic and

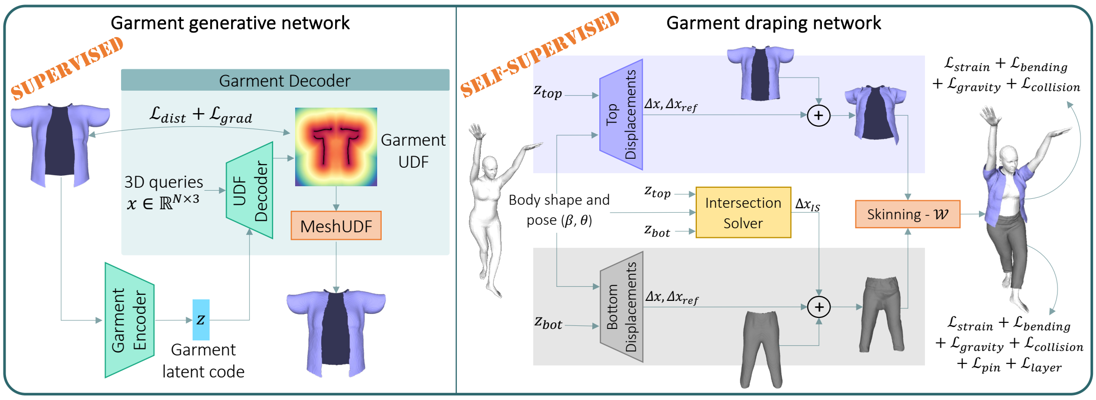

Garment generative network
SUPERVISED
Garment Decoder
$\mathcal{L}_{dist} + \mathcal{L}_{grad}$
Garment UDF
3D queries $x \in \mathbb{R}^{N\times 3}$
UDF Decoder
MeshUDF
Garment Encoder
z
Garment latent code
Garment draping network
SELF-SUPERVISED
z_{top}
Top Displacements
$\Delta x, \Delta x_{ref}$
$\mathcal{L}_{strain} + \mathcal{L}_{bending} + \mathcal{L}_{gravity} + \mathcal{L}_{collision}$
Body shape and pose (β, θ)
z_{top}
z_{bot}
Intersection Solver
Δx_{IS}
Skinning - $\mathcal{W}$
Bottom Displacements
$\Delta x, \Delta x_{ref}$
z_{bot}
$\mathcal{L}_{strain} + \mathcal{L}_{bending} + \mathcal{L}_{gravity} + \mathcal{L}_{collision} + \mathcal{L}_{pin} + \mathcal{L}_{layer}$

Figure 7.2 – **Overview of our framework. Left:** Garment generative network, trained to embed garments into compact latent codes and predict their unsigned distance field (UDF) from such vectors. UDFs are then meshed using [GSF22]. **Right:** Garment draping network, conditioned on the latent codes of the generative network. It is trained in a self-supervised way to predict the displacements Δx and Δx_{ref} to be applied to the vertices of given garments, before skinning them according to body shape and pose (β, θ) with the predicted blending weights $\mathcal{W}$. It includes an Intersection Solver module to prevent intersection between top and bottom garments.

expressive material model. Both methods, however, require training one network per garment, a limitation we remove.

7.3 Method

We aim to realistically deform and drape generic garments over human bodies of various shapes and poses. To this end, we introduce the `DrapeNet` framework, presented in Fig. 7.2. It comprises a generative network shown on the left and a draping network shown on the right. Only the first is trained in a supervised manner, but using only static unposed garments meshes. This is key to avoiding having to run physics-based simulations to generate ground-truth data. Furthermore, we condition the draping network on latent vectors representing the input garments, which allows us to use the same network for very different garments, something that competing methods [BME21, SOC22] cannot do.

The generative network is a decoder trained using an encoder that turns a garment into a latent code **z** that can then be decoded to an Unsigned Distance Function (UDF), from which a triangulated mesh can be extracted in a differentiable manner [GSF22]. The UDF representation allows us to accurately represent open surfaces and the many openings that garments typically feature. Since the top and bottom garments – shirts and trousers – have different patterns, we train one generative model for each. Both networks have the same architecture but different weights.

The resulting *garment generative network* is only trained to output garments in a canonical shape, pose, and size that fit a neutral SMPL [LMR+15] body. Draping the resulting garments to bodies in non-canonical poses is then entrusted to a *draping network*, again one for the top and one for the bottom. As in [BME21, SOC22, LGRF22], this network predicts vertex displacements w.r.t. the neutral position. The deformed garment is then skinned onto the articulated body model. To enable generalization to different tops and bottoms, we condition the draping process on the garment latent codes of the generative network, shown as $\mathbf{z}_{\text{top}}$ and $\mathbf{z}_{\text{bot}}$ in Fig. 7.2.

We use a small database of static unposed garments loosely aligned with bodies in the canonical position to train the two garment generating networks. This being done, we exploit physics-based constraints to train in a fully self-supervised manner the top and bottom draping networks for realism, without interpenetrations with the body and between the garments themselves.

7.3.1 Garment Generative Network

To encode garments into latent codes that can then be decoded into UDFs, we rely on a point cloud encoder that embeds points sampled from the unposed garment surface into a compact vector. This lets us obtain latent codes for previously unseen garments in a single inference pass from points sampled from its surface. This can be done given any arbitrary surface triangulation. Hence, it gives us the flexibility to operate on any given garment mesh.

We use DGCNN [WSL+19] as the encoder. It first propagates the features of points within the same local region at multiple scales and then aggregates them into a single global embedding by max pooling. We pair it with a decoder that takes as input a latent vector, along with a point in 3D space, and returns its (unsigned) distance to the garment. The decoder is a multi-layer perceptron (MLP) that relies on Conditional Batch Normalization [VSM+17] for conditioning on the input latent vector.

We train the encoder and the decoder by encouraging them to jointly predict distances that are small near the training garments' surface and large elsewhere. Because the algorithm we use to compute triangulated meshes from the predicted distances [GSF22] relies on the gradient vectors of the UDF field, we also want these gradients to be as accurate as possible [AL20b, ZWLS21]. We therefore minimize the loss

$$L_{garm} = L_{dist} + \lambda_g L_{grad}\ , \tag{7.1}$$

where L_{dist} encodes our distance requirements, L_{grad} the gradient ones, and λ_g is a weight balancing their influence.

More formally, at training time and given a mini-batch comprising B garments, we sample a fixed number P of points from the surface of each one. For each resulting point cloud $\mathbf{p}_i$

$(1 \le i \le B)$, we use the garment encoder E_G to compute the latent code

$$\mathbf{z}_i = E_G(\mathbf{p}_i) \tag{7.2}$$

and use it as input to the decoder D_G. It predicts an UDF field supervised with Eq. (7.1), whose terms we define below.

Distance Loss. Having experimented with many different formulations of this loss, we found the following one both simple and effective. Given N points $\{\mathbf{x}_{ij}\}_{j\le N}$ sampled from the space surrounding the i-th garment, we pick a distance threshold δ, clip all the ground-truth distance values $\{y_{ij}\}$ to it, and linearly normalize the clipped values to the range $[0,1]$. This yields normalized ground-truth values $\bar{y}_{ij} = \min(y_{ij}, \delta)/\delta$. Similarly, we pass the output of the final layer of D_G through a sigmoid function $\sigma(\cdot)$ to produce a prediction in the same range for point $\mathbf{x}_{ij}$

$$\widetilde{y}_{ij} = \sigma(D_G(\mathbf{x}_{ij}, \mathbf{z}_i)) \,. \tag{7.3}$$

Finally, we take the loss to be

$$\mathscr{L}_{dist} = \text{BCE}\left[(\bar{y}_{ij})_{j\le N}^{i\le B} \,,\, (\widetilde{y}_{ij})_{j\le N}^{i\le B} \right], \tag{7.4}$$

where $\text{BCE}[\cdot,\cdot]$ stands for binary cross-entropy. As observed in [DZW$^+$20], the sampling strategy used for points $\mathbf{x}_{ij}$ strongly impacts training effectiveness. We describe ours in the appendix. In our experiments, we set $\delta = 0.1$, being the top and bottom garments normalized respectively into the upper and lower halves of the $[-1,1]^3$ cube.

Gradient Loss. Given the same sample points as before, we take the gradient loss to be

$$\mathscr{L}_{grad} = \frac{1}{BN} \sum_{i,j} \left\| \mathbf{g}_{ij} - \widehat{\mathbf{g}_{ij}} \right\|_2^2 \,, \tag{7.5}$$

where $\mathbf{g}_{ij} = \nabla_{\mathbf{x}} y_{ij} \in \mathbb{R}^3$ is the ground-truth gradient of the i-th garment's UDF at $\mathbf{x}_{ij}$ and $\widehat{\mathbf{g}_{ij}} = \nabla_{\mathbf{x}} D_G(\mathbf{x}_{ij}, \mathbf{z}_i)$ the one of the predicted UDF, computed by backpropagation.

7.3.2 Garment Draping Network

We describe our approach to draping generic garments as opposed to specific ones and our self-supervised scheme. We assume that all garments are made of a single common fabric material, and we drape them in a quasi-static manner.

Draping Generic Garments

We rely on SMPL [LMR+15] to parameterize the body in terms of shape (β) and pose (θ) parameters. It uses Linear Blend Skinning to deform a body template. Since garments generally follow the pose of the underlying body, we extend the SMPL skinning procedure to the 3D volume around the body for garment draping. Given a point $\mathbf{x} \in \mathbb{R}^3$ in the garment space, its position $D(\mathbf{x}, \beta, \theta, \mathbf{z})$ after draping becomes

$$\begin{aligned} D(\mathbf{x}, \beta, \theta, \mathbf{z}) &= W(\mathbf{x}_{(\beta,\theta,\mathbf{z})}, \beta, \theta, \mathscr{W}(\mathbf{x}))\,, \\ \mathbf{x}_{(\beta,\theta,\mathbf{z})} &= \mathbf{x} + \Delta x(\mathbf{x}, \beta) + \Delta x_{\text{ref}}(\mathbf{x}, \beta, \theta, \mathbf{z})\,, \\ \Delta x_{\text{ref}}(\mathbf{x}, \beta, \theta, \mathbf{z}) &= \mathscr{B}(\beta, \theta) \cdot \mathscr{M}(x, \mathbf{z})\,, \end{aligned} \tag{7.6}$$

where $W(\cdot)$ is the SMPL skinning function, applied with blending weights $\mathscr{W}(\mathbf{x})$, over the point displaced by $\Delta x(\mathbf{x}, \beta)$ and $\Delta x_{\text{ref}}(\mathbf{x}, \beta, \theta, \mathbf{z})$. $\mathscr{W}(\mathbf{x})$ and $\Delta x(\mathbf{x}, \beta)$ are computed as in [STOC21, LGRF22]. However, they only give an initial deformation for garments that roughly fits the underlying body. To refine it, we introduce a new term, $\Delta x_{\text{ref}}(\mathbf{x}, \beta, \theta, \mathbf{z})$. It is a deformation field conditioned on body parameters β and θ, and on the garment latent code $\mathbf{z}$ from the generative network. Following the linear decomposition of displacements in SMPL, it is the composition of an embedding $\mathscr{B}(\beta, \theta) \in \mathbb{R}^{N_{\mathscr{B}}}$ of body parameters and a displacement matrix $\mathscr{M}(x, \mathbf{z}) \in \mathbb{R}^{N_{\mathscr{B}} \times 3}$ conditioned on $\mathbf{z}$. Being conditioned on the latent code $\mathbf{z}$, Δx_{ref} can deform different garments differently, unlike the methods of [BME21, SOC22]. The number of vertices does not need to be fixed, since displacements are predicted separately for each vertex.

Since we have distinct encodings for the top and bottom garments, for each one we train two MLPs ($\mathscr{B}$, $\mathscr{M}$) to predict Δx_{ref}. The other MLPs for $\mathscr{W}(\cdot)$ and $\Delta x(\cdot)$ are shared.

Self-Supervised Training

We first learn the weights of $\mathscr{W}(\cdot)$ and $\Delta x(\cdot)$ as in [STOC21, LGRF22], which does not require any annotation or simulation data but only the blending weights and shape displacements of SMPL. We then train our deformation fields Δx_{ref} in a fully self-supervised fashion by minimizing the physics-based losses introduced below. In this way, we completely eliminate the huge cost that extensive simulations would entail.

Top Garments. For upper body garments – shirts, t-shirts, vests, tank tops, etc. – the deformation field is trained using the loss from [SOC22], expressed as

$$\mathscr{L}_{top} = \mathscr{L}_{strain} + \mathscr{L}_{bend} + \mathscr{L}_{gravity} + \mathscr{L}_{col}\,, \tag{7.7}$$

where $\mathscr{L}_{strain}$ is the membrane strain energy of the deformed garment, $\mathscr{L}_{bend}$ the bending energy caused by the folding of adjacent faces, $\mathscr{L}_{gravity}$ the gravitational potential energy, and $\mathscr{L}_{col}$ a penalty for collisions between body and garment. Unlike in [SOC22], we only consider the quasi-static state after draping, that is, without acceleration.

Bottom Garments. Due to gravity, bottom garments, such as trousers, would drop onto the floors if we used only the loss terms of Eq. (7.7). We thus introduce an extra loss term to constrain the deformation of vertices around the waist and hips. The loss becomes

$$\begin{aligned} \mathcal{L}_{bottom} &= \mathcal{L}_{strain} + \mathcal{L}_{bend} + \mathcal{L}_{gravity} + \mathcal{L}_{col} + \mathcal{L}_{pin}, \\ \mathcal{L}_{pin} &= \sum_{v \in V} |\Delta x_y|^2 + \lambda(|\Delta x_x|^2 + |\Delta x_z|^2) , \end{aligned} \tag{7.8}$$

where V is the set of garment vertices whose closest body vertices are located in the region of the waist and hips. See the appendix for details. The terms Δx_x, Δx_y and Δx_z are the deformations along the X, Y and Z axes, respectively. λ is a positive value smaller than 1 that penalizes deformations along the vertical direction (Y axis) and produces natural deformations along the other directions.

Top-Bottom Intersection. To ensure that the top and bottom garments do not intersect with each other when we drape them on the same body, we define a loss $\mathcal{L}_{IS}$ that ensures that when the top and the bottom garments overlap, the bottom garment vertices are closer to the body mesh than the top ones, which prevents them from intersecting – this is arbitrary, and the following could be formulated the other way around. To this end, we introduce an Intersection Solver (IS) network. It predicts a displacement correction Δx_{IS}, added only when draping bottom garments as

$$\tilde{\mathbf{x}}_{(\mathbf{z}_{top},\mathbf{z}_{bot})} = \mathbf{x}_{(\mathbf{z}_{bot})} + \Delta x_{IS}(\mathbf{x}, \mathbf{z}_{top}, \mathbf{z}_{bot}) , \tag{7.9}$$

where we omit the dependency of $\tilde{\mathbf{x}}$, $\mathbf{x}$ and Δx_{IS} on the body parameters (β, θ) for simplicity. $\mathbf{z}_{top}$ and $\mathbf{z}_{bot}$ are the latent codes of the top and bottom garments, and $\mathbf{x}_{(\mathbf{z}_{bot})}$ is the input point displaced according to Eq. (7.6). The skinning function of Eq. (7.6) is then applied to $\tilde{\mathbf{x}}_{(\mathbf{z}_{top},\mathbf{z}_{bot})}$ for draping. $\Delta x_{IS}(\cdot)$ is implemented as a simple MLP and trained with

$$\mathcal{L}_{IS} = \mathcal{L}_{bottom} + \mathcal{L}_{layer}, \tag{7.10}$$

where $\mathcal{L}_{layer}$ is a loss whose minimization requires the top and bottom garments to be separated from each other. We formulate it as

$$\mathcal{L}_{layer} = \sum_{v_B \in C} max(d_{bot}(v_B) - \gamma d_{top}(v_B), 0) , \tag{7.11}$$

where C is the set of body vertices covered by both the top and bottom garments, $d_{top}(\cdot)$ and $d_{bot}(\cdot)$ the distance to the top and the bottom garments respectively, and γ a positive value smaller than 1 (more details in the appendix).

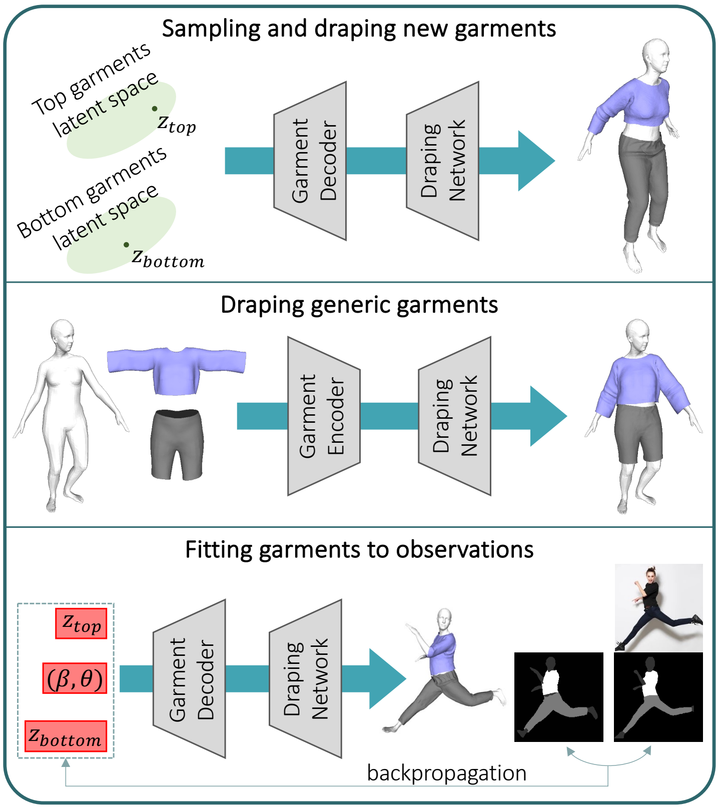


Figure 7.3 – **Overview of `DrapeNet` applications. Top:** New garments can be sampled from the latent spaces of the generative networks, and deformed by the draping networks to fit to a given body. **Center:** The garment encoders and the draping networks form a general purpose framework to drape any garment with a single forward pass. **Bottom:** Being a differentiable parametric model, our framework can reconstruct 3D garments by fitting observations such as images or 3D scans. The red boxes indicate the parameters optimized in this process.

7.4 Experiments

We first describe our experimental setup and test `DrapeNet` for the different purposes depicted by Fig. 7.3. They include reconstructing different kinds of garments and editing them by manipulating their latent codes. We then gauge the draping network both qualitatively and quantitatively. Finally, we use `DrapeNet` to reconstruct garments from images and 3D scans.

7.4.1 Settings, Datasets and Metrics

Datasets. Both our generative and draping networks are trained with garments from CLOTH3D [BME20], a synthetic dataset that contains over 7K sequences of animated 3D humans parametrized used the SMPL model and wearing different garments. Each sequence comprises up to 300 frames and features garments coming from different templates. For training, we randomly selected 600 top garments (t-shirts, shirts, tank tops, etc.) and 300 bottom garments (both long and short trousers). Neither for the generative nor for the draping networks did we use the simulated deformations of the selected garments. Instead, we trained the networks using only garment meshes on average body shapes in T-pose. By contrast, for testing purposes, we selected random clothing items – 30 for top garments and 30 bottom ones – and considered *whole* simulated sequences.

Training. We train two different models for top and bottom garments, both for the generative and for the draping parts of our framework. First, the generative models are trained on the 600/300 neutral garments Then, with the generative networks weights frozen, we train the draping networks by following [SOC22]: body poses θ are sampled randomly from the AMASS [MGT$^+$19] dataset, and shapes β uniformly from $[-3,3]^{10}$ at each step. The other hyperparameters are given in the appendix.

Metrics. We report the Euclidean distance (ED), interpenetration ratio between body and garment (B2G), and intersection between top and bottom garments (G2G). ED is computed between corresponding vertices of the considered meshes. B2G is the area ratio between the garment faces inside the body and the whole surface as in [LGRF22]. Since CLOTH3D exclusively features pairs of top/bottom garments with the bottom one closer to the body, G2G is computed by detecting faces of the bottom garment that are outside of the top one, and taking the area ratio between those and the overall bottom garment surface.

7.4.2 Garment Paramerization

We first test the encoding-decoding scheme of Section 7.3.1.

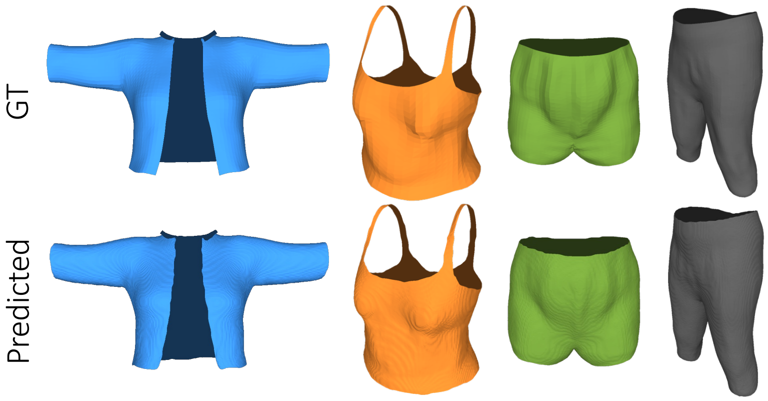


Figure 7.4 – **Generative network: reconstruction of unseen garments in neutral pose/shape.** The latent codes are obtained with the garment encoder, then decoded into open surface meshes.

Encoding-Decoding Previously Unseen Garments. The generative network of Fig. 7.2 is designed to project garments into a latent space and to reconstruct them from the resulting latent vectors. In Fig. 7.4, we visualize reconstructed previously-unseen garments from CLOTH3D. The reconstructions are faithful to the input garments, including fine-grained details such as the shirt collar on the left or the shoulder straps of the tank top.

Semantic Manipulation of Latent Codes. Our framework enables us to edit a garment by manipulating its latent code. For the resulting edits to have a semantic meaning, we assigned binary labels corresponding to features of interest to 100 training garments. For instance, we labeled garments as having “short sleeves” (label = 0) or “long sleeves” (label = 1). Then, we fit a linear logistic regressor to the garment latent codes. After training, the regressor weights indicate which dimensions of the latent space control the feature of interest. To this end, we first apply min-max normalization to the absolute weight values and then zero out the ones below a certain threshold, empirically set to 0.5. The remaining non-zero weights indicate which dimensions of the latent codes should be increased or decreased to edit the studied feature. To create Fig. 7.5, we applied this simple procedure to control the sleeve length and the front opening for top garments along with the length for bottom garments. As can be seen from the figure, our latent representations give us the ability to edit a specific garment feature while leaving other aspects of the garment geometry unchanged.

7.4.3 Garment Draping

We now turn to the evaluation of the draping network and compare its performance to those of DeePSD [BMTE21] or DIG [LGRF22], two *fully supervised* learning methods trained on CLOTH3D. DeePSD takes the point cloud of the garment mesh as input and predicts blending weights and pose displacements for each point; DIG drapes garments with a learned skinning

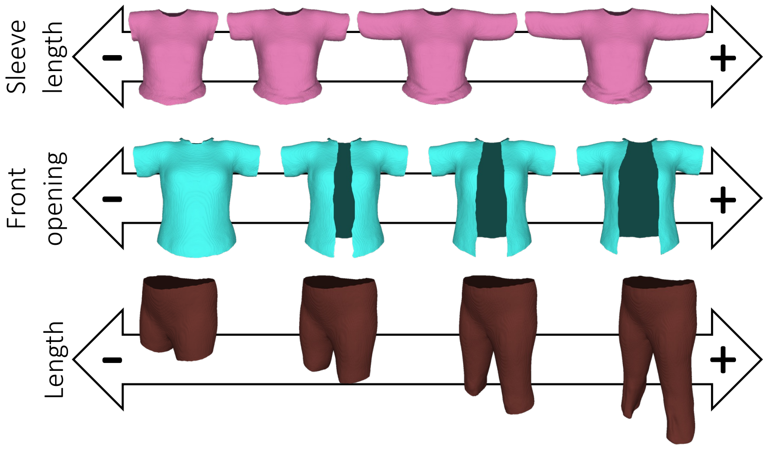


Figure 7.5 – **Garment editing.** The latent codes produced by the garment encoder can be manipulated to edit specific features of the corresponding garments, without altering the overall geometry.

	DeePSD	DIG	Ours
ED-top (mm)	28.1	29.6	47.9
ED-bottom (mm)	18.3	20.0	27.3
B2G-top (%) ↓	7.2	1.8	**0.9**
B2G-bottom (%) ↓	3.4	0.8	**0.3**
G2G (%) ↓	2.0	4.0	**0.5**

Table 7.1 – **Draping unseen garment meshes.** Comparison between DeePSD, DIG and our method, for top and bottom garments: Euclidean distance (ED), intersections with the body (B2G) and between garments (G2G) as ratio of intersection areas.

field that can be applied to generic 3D points, but is similar for all garments. We chose those because, like `DrapeNet`, they both can deform garments of arbitrary geometry and topology.

Draping Unseen Meshes. We drape previously unseen garments on different bodies in random poses. We first encode the garments and use the resulting latent codes to condition the draping network, whose inference takes ~5ms.

We provide qualitative results in Fig. 7.6 and report quantitative ones in Table 7.1. Despite being completely self-supervised, `DrapeNet` delivers the lowest ratio of body-garment interpenetrations (B2G) for both top and bottom garments and the least intersections between them (G2G).

However, `DrapeNet` also yields higher ED values, which makes sense because there is more than one way to satisfy the physical constraints and to achieve realism. Hence, in the absence of explicit supervision, there is no reason for the answer picked by `DrapeNet` to be exactly

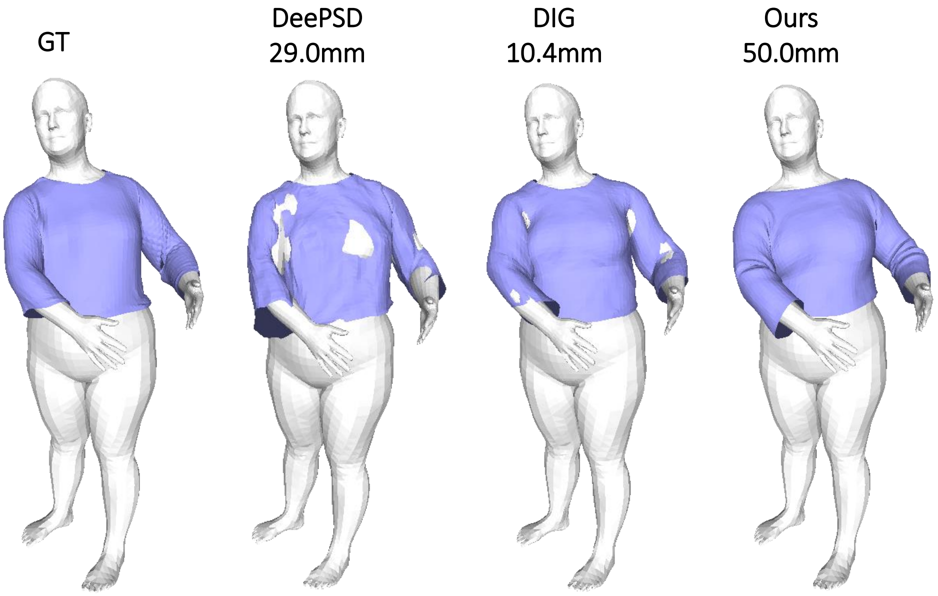


Figure 7.6 – **Comparison between DeePSD, DIG and our method.** Ours is more realistic despite having the highest Euclidean distance (ED) error (**left**), and has less intersection between garments (**right**). **Left** also shows that Δx_{ref} is necessary for realistic deformations.

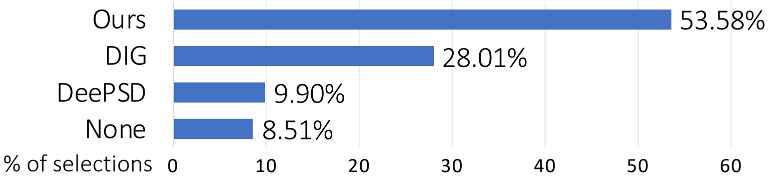


Figure 7.7 – **Human evaluation of draping results.** When shown draping results of our method, DIG and DeePSD, evaluators selected ours as the most realistic one in more than half of the cases. *None* refers to the case when they had no clear preference.

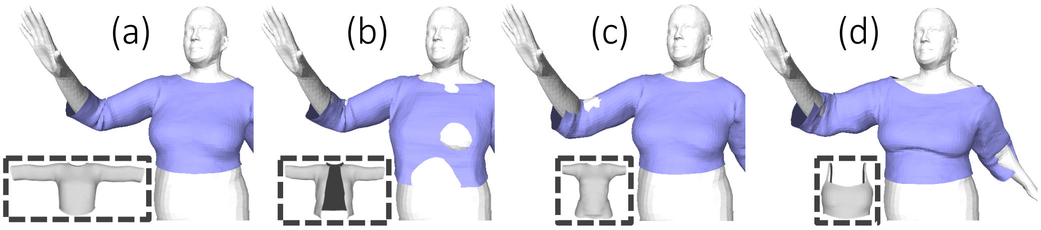


Figure 7.8 – **Switching input latent codes of the draping network.** Draping the same shirt by conditioning the draping network with **(a)** the corresponding latent code, **(b)** the code of an open vest, **(c)** of a t-shirt and **(d)** of a tank top. Gray meshes in dashed boxed are the garments corresponding to the input latent codes.

the same as the one picked by the simulator. In fact, as argued in [BME21] and illustrated by Fig. 7.6, which is representative in terms of ED, a low ED value does not necessarily correspond to a realistic draping. To confirm this, we conducted a human evaluation study by sharing a link to a website on friends groupchats. We gave no further instructions or details besides those given on the site and reproduced in the appendix. The website displays 3 drapings of the same garment over the same posed body, one computed using our method and the others using the other two. The users were asked to select which one of the three seemed more realistic and more pleasant, with a fourth potential response being “none of them". We obtained feedback from 187 different people. A total of 1258 individual examples were rated and we collected 3738 user opinions. In other words, each user expressed 20 opinions on average. The chart in Fig. 7.7 shows that our method was selected more than 50% of the times, with a large gap over the second best, DIG [LGRF22], selected less than 30% of the time. This result confirms that `DrapeNet` can drape garments with better perceptual quality than the competing methods.

Ablation Study. In Fig. 7.8, we show what happens when the draping network is conditioned with a latent code of a garment that does not match the input one. This creates unnatural deformations on the front when using the code of a shirt with a front opening to deform a shirt without an opening. Similarly, the sleeves penetrate the arms when conditioning with the code of a short sleeves shirt. This demonstrates that the draping network truly exploits the latent codes to predict garment-dependent deformation fields.

In Fig. 7.6 **left** we show that removing our novel displacement term $\Delta x_{\text{ref}}(\cdot)$ from Eq. (7.6) leads to unrealistic results.

We also ablate the influence of our Intersection Solver and observe that G2G increases from 0.5% to 1.1% without it. This demonstrates the effectiveness of this component at reducing collisions between top and bottom garments.

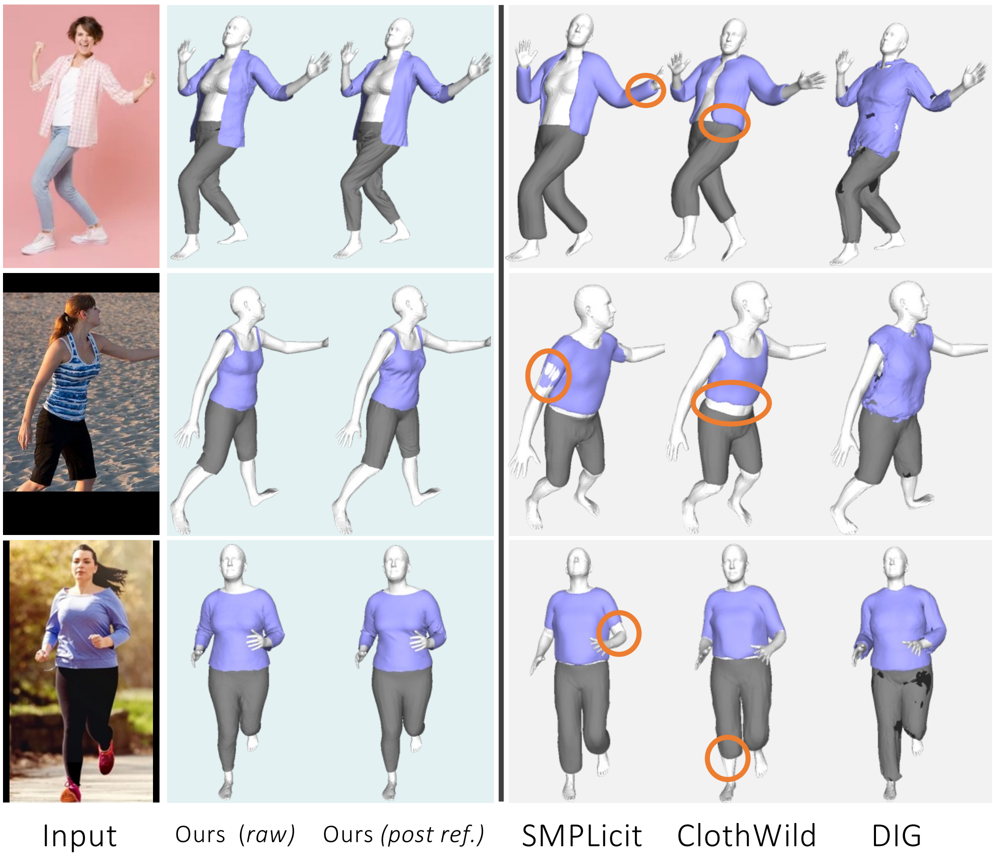


Figure 7.9 – **Recovering garments and bodies from images.** From left to right we show the input image and the 3D models recovered with our method (without and with post-refinement), and competitors methods: SMPLicit [CPA+21], ClothWild [MNSL22], DIG [LGRF22].

7.4.4 Fitting Observations

Since our method is end-to-end differentiable, it can be used to reconstruct 3D models of people and their garments from partial observations, such as 2D images and 3D scans.

Fitting Images. Given an image of a clothed person, we use the algorithm of [YRSJ21, YSW+20] to get initial estimates for the body parameters (β,θ) and a segmentation mask $\mathbf{S}$. Then, starting with the mean of the learned codes $\mathbf{z}$, we reconstruct a mesh for the body and its garments by minimizing

$$\begin{aligned} L(\beta,\theta,\mathbf{z}) &= L_{\mathrm{IoU}}(R(D(\mathbf{G},\beta,\theta,\mathbf{z}),\mathrm{SMPL}(\beta,\theta)),\ \mathbf{S})\,, \\ \mathbf{G} &= \mathrm{MeshUDF}(D_G(\mathbf{z}))\,, \end{aligned} \tag{7.12}$$

w.r.t. $\mathbf{z}$, β and θ, where L_{IoU} is the IoU loss [LZK+21] in pixel space penalizing discrepancies between 2D masks, $R(\cdot)$ is a differentiable mesh renderer [RRN+20], and $\mathbf{G}$ is the set of vertices of the garment mesh reconstructed with our garment decoder using $\mathbf{z}$. $D(\cdot)$ and $\mathrm{SMPL}(\cdot)$ are

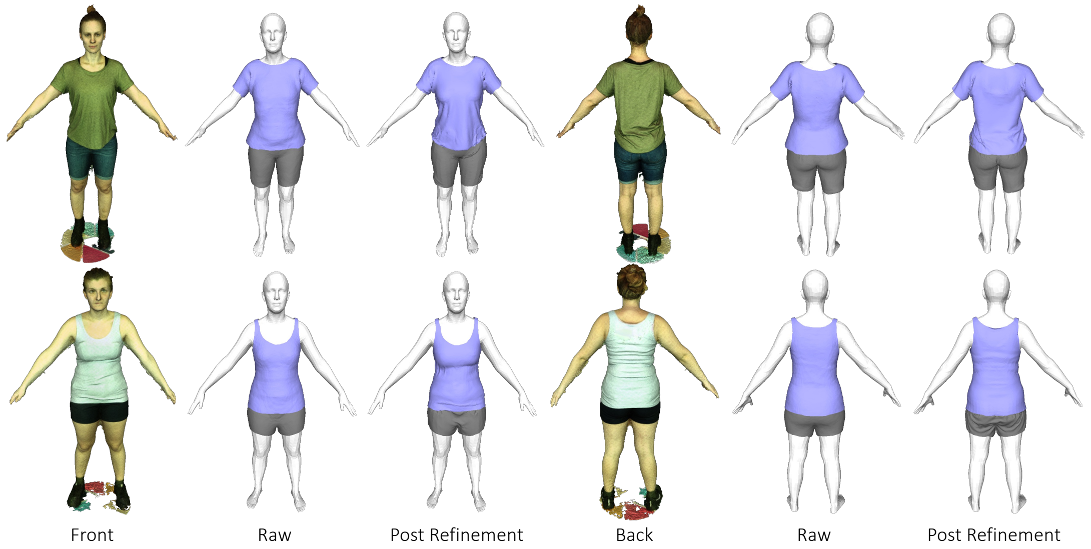


Figure 7.10 – **Recovering garments and bodies from 3D scans.** We show 3D models recovered with our method from scans of the SIZER dataset [TBTPM20]. *Raw* indicates the model recovered with Eq. (7.13) from the 3D scan. *Post Refinement* refers to the models further refined with the physics-based losses.

the garment and body skinning functions defined in Eq. (7.6) and in [LMR⁺15], respectively. To ensure pose plausibility, θ is constrained by an adversarial pose prior [DRC⁺22].

For the sake of simplicity, Eq. (7.12) formulates the reconstruction of a single garment $\mathbf{G}$. In practice, we extend this formulation to both the top and the bottom garments shown in the target image. Fig. 7.9 depicts the results of minimizing this loss. It outperforms the state-of-the-art methods SMPLicit [CPA⁺21], ClothWild [MNSL22] and DIG [LGRF22]. The garments we recover follow the ones in the input image with higher fidelity and visual quality, without interpenetration between the body and the garments or between the two garments.

After this optimization, we can further refine the result by minimizing the physics-based objectives of Eq. (7.7) w.r.t. the per-vertex displacements of the reconstructed garments, as opposed to w.r.t. the latent vectors. We describe this procedure in the appendix. As shown in the third column of Fig. 7.9, this further boosts the realism of the reconstructed garments. Note that this refinement is feasible thanks to the open surface representation allowed by our UDF model. Applying these physically inspired losses to an inflated garment, as produced by SMPLicit, ClothWild and DIG, yields poor results with many self-intersections, as shown in the appendix.

Fitting 3D scans. Given a 3D scan of a clothed person and segmentation information, we apply a strategy similar to the one presented above and minimize

$$L(\beta,\theta,\mathbf{z}) = d(D(\mathbf{G},\beta,\theta,\mathbf{z}),\ \mathbf{S_G}) + \vec{d}(\mathrm{SMPL}(\beta,\theta),\ \mathbf{S_B}), \tag{7.13}$$

w.r.t. $\mathbf{z}$, β and θ, where $\mathbf{S_G}$ and $\mathbf{S_B}$ denote the segmented garment and body scan points, and $d(a,b)$ and $\vec{d}(a,b)$ are the bidirectional and the one-directional Chamfer distance from b to a. Similarly to Eq. (7.12), we apply Eq. (7.13) to recover both the top and bottom garments. Fig. 7.10 shows our fitting results for some scans of the SIZER dataset [TBTPM20]. The recovered 3D models closely match the input scans. Moreover, we can also apply a post-refinement procedure similar to the one described above, by minimizing both the physics-based losses from Eq. (7.7) and the Chamfer distance to the input scan w.r.t. the 3D coordinates of the vertices of the reconstructed models. This leads to even more realistic results, with fine wrinkles aligning to the input scans.

7.5 Conclusion

We have shown that physics-based self-supervision can be leveraged to learn a single parameterization for many different garments to be draped on human bodies in arbitrary poses. Our approach relies on UDFs to represent garment surfaces and on a displacement field to drape them, which enables us to handle a continuous manifold of garments without restrictions on their topology. Our whole pipeline is differentiable, which makes it suitable for solving inverse problems and for modeling clothed people from image data.

Future work will focus on modeling dynamic poses instead of only static ones. This is of particular relevance for loose clothes, where our reliance on the SMPL skinning prior should be relaxed. Moreover, we will investigate replacing our current global latent code by a set of local ones to yield finer-grained control both for garment editing and draping.

8 Conclusions

8.1 Summary

We have introduced three key technical innovations in this work:

1. a hybrid shape decoder that combines a voxel-based representation with 2D atlases,
2. a differentiable parameterization of watertight meshes using deep implicit surfaces,
3. an extension of this approach to handle open surfaces through a novel meshing procedure.

These foundational components enable the exploration of novel applications for generating 3D surfaces using neural networks. We have demonstrated their effectiveness and versatility in various contexts, including single view reconstruction from images and sketches, as well as gradient-based optimization tasks. Our experiments encompassed reconstruction and manipulation of 3D surfaces from synthetic and real images of common objects and garments, as well as sketches.

Our hybrid representation provides a systematic and principled approach to elevate 2D features to 3D, which has proven advantageous for single and multi view reconstruction. However, using the multiple patches it produces in downstream applications remains challenging, and the fusion of voxels and atlases lacks generalizability to other 3D applications. Nonetheless, the concept of structuring 3D space with a grid as support for other primitives was used in more recent applications [RBS^{+}22, HCJS20, MESK22] and remains a valid approach.

Implicit shape representations with SDFs and Occupancy fields address this problem by yielding smooth watertight meshes that lend themselves well to a great variety of applications and downstream tasks. The structure of this thesis therefore aligns with a prevalent trend in the research community, where implicit shape representations have gained significant traction over the past four years. These implicit representations have demonstrated their superiority in terms of performance, simplicity, generality, and robustness compared to explicitly deforming mesh templates or atlases. The success of implicit representations has been allowed by their utilization of smooth multi-layer perceptrons, which readily facilitates gradient descent and enables the construction of high-dimensional features. One notable application that gained significant popularity is the representation of 5D radiance fields through overfitting a single scene [MPT$^+$20], which differs from the focus of this thesis. We ourselves have embraced this shift towards implicit representations, transitioning from the utilization of explicit representations like our own hybrid shape decoder, to predominantly employing purely implicit surfaces, particularly signed distance functions (SDFs). This shift aligns with the broader trend of embracing implicit representations within the research community.

The data-driven watertight mesh parameterization we introduced next thus offers increased versatility, as we demonstrated on with a wide range of applications. Within the same framework, we employed it for single view reconstruction in a conventional feed-forward manner, as well as for refining initial reconstructions and tackling various other tasks, including physical shape optimization and fitting sparse observations such as 3D point clouds or sketches. This approach gained significant attention and has since sparked further research and subsequent developments.

Unsigned Distance Fields (UDFs) offer an even greater level of generality as they can effectively model both open and closed surfaces, alleviating the need for watertight shapes and simplifying data pre-processing. Notably, our contribution lies in the introduction of a differentiable meshing procedure, thus enhancing the practicality of UDFs. While our primary demonstration focuses on a data-driven parameterization for garments, it is important to note that our meshing procedure finds utility beyond latent models. Specifically, it extends to the meshing of generic UDF fields, such as entire implicit scenes stored as augmented radiance fields [LLL$^+$22]. However, it is important to acknowledge that UDFs present inherent challenges. They are more intricate to train and mesh compared to their signed distance function (SDF) counterparts, rendering them relatively less robust and established in comparison. Additionally, due to their relative novelty, UDFs remain less mature and less prevalent within the field. Furthermore, while our main application of UDFs has been focused on garment parameterization, it is worth noting that they contend with other specific parameterization approaches for such specialized cases [LGF23, KL21].

Furthermore, this thesis has underscored the great potential of the render and compare approach, a technique also known as *analysis by synthesis*, that we employed by combining differentiable rendering with data-driven shape priors. The efficacy of this iterative refinement process has been extensively demonstrated across multiple applications discussed in Chapters 4 to 7. Through this approach, the precise fitting of sparse observations has been achieved, yielding results that surpass those obtained with simple feed-forward neural networks. Of particular significance is the application of the render and compare approach to interactive shape editing, as presented in Chapter 5. This practical implementation further exemplifies the versatility and potential of this technique in facilitating intuitive user interactions. In summary, the render and compare approach serves as a powerful tool for enhancing shape refinement, enabling the accurate alignment of sparse observations, and maximizing the potential of data-driven shape priors across various applications explored in this thesis.

8.2 Limitations and Future work

One limitation of our image and sketch fitting approaches is their focus solely on silhouettes and external contours, disregarding other visual cues like shading and color. Addressing this limitation in the future could involve learning a prior for surface appearance. However, we identify two challenges in doing so. Firstly, it would necessitate the development of a representation and model that jointly learns shape and appearance. This is non-trivial as appearance is inherently a 2D surface property, while shapes are typically represented as 3D volumetric fields. Secondly, such models would require careful consideration to ensure robust generalization. By contrast, considering only binary silhouettes reduces domain gaps and facilitates robust generalization.

Another limitation of the render and compare strategy is its reliance on known camera parameters, which we assume to be known in most cases. For instance, in our sketching interface developed in Chapter 5, proper initialization to an example shape is required to orient the drawing space. As a result, it is not applicable to non-oriented sketches. A potential solution to address this limitation would involve jointly regressing both the 3D shape and its pose.

Furthermore, although our shape decoders have shown promising results, there is still room for improvement in terms of the quality of the generated meshes. Currently, the generated meshes may exhibit discernible differences compared to handcrafted ones, in terms of fine details, surface regularity, or connectivity of the different shape components. Initial experiments with an adversarial training paradigm did not yield any noticeable improvement in this regard. One potential approach to enhance the level of detail is to deviate slightly from a global model that is conditioned on a single latent code and instead adopt local latent codes to model different parts of the shape. However, achieving this while maintaining the ability to fit sparse observations with large occlusions poses a challenge. To address this, we envision the use of an attention model that allows the local codes to attend to other regions, thereby capturing a global prior.

Deep implicit shape representations can benefit from further improvements in the learning process, particularly for Uniform Distance Fields (UDFs) that are susceptible to artifacts due to the localization of the surface at a singularity of the field. Currently, our latent models require pre-processing of meshes and the generation of supervision samples throughout the entire 3D space, even though the level set of interest is confined to a 2D manifold within the ambient 3D space. The selection of optimal supervision samples for implicit networks remains an underexplored area of research [DZW$^+$20]. An alternative approach that could hold promise is the utilization of active learning methods to effectively identify and select the most informative supervision samples. This avenue warrants further investigation to enhance the learning process and address the challenges associated with training deep implicit shape representations.

A Appendix

A.1 Supplementary material for UCLID-Net

A.1.1 Metrics

This subsection defines the metrics and loss functions used in the corresponding chapter.

Chamfer-L1

The Chamfer-L1 (CD–L_1) pseudo distance d_{CD_1} between point clouds $X = \left\{x_i | 1 \leq i \leq N, x_i \in \mathbb{R}^3\right\}$ and $Y = \left\{y_j | 1 \leq j \leq M, y_j \in \mathbb{R}^3\right\}$ is the following:

$$d_{CD_1}(X,Y) = \frac{1}{|X|} \cdot \sum_{x \in X} \min_{y \in Y} \left\|x - y\right\|_2 + \frac{1}{|Y|} \cdot \sum_{y \in Y} \min_{x \in X} \left\|x - y\right\|_2 , \tag{A.1}$$

where $\|.\|_2$ is the Euclidean distance. We use CD–L_1 as a validation metric on the Pix3D dataset, according to the original procedure. It is applied on shapes normalized to bounding box $[-0.5, 0.5]^3$, and sampled with 1024 points.

Chamfer-L2

The Chamfer-L2 (CD–L_2) pseudo distance d_{CD_2} between point clouds X and Y is the following:

$$d_{CD_2}(X,Y) = \frac{1}{|X|} \cdot \sum_{x \in X} \min_{y \in Y} \left\|x - y\right\|_2^2 + \frac{1}{|Y|} \cdot \sum_{y \in Y} \min_{x \in X} \left\|x - y\right\|_2^2 \tag{A.2}$$

i.e. CD–L_2 is the average of the *squares* of closest neighbors matching distances. We use CD–L_2 as a validation metric on the ShapeNet dataset. It is applied on shapes normalized to unit radius sphere, and sampled with 2048 points.

Earth Mover's distance

The Earth Mover's Distance (EMD) is a distance that can be used to compare point clouds as well. It is defined as

$$d_{EMD}(X,Y) = \min_{T \in \wp(N,M)} \sum_{1 \leq i \leq N, 1 \leq j \leq M} T_{i,j} \times \left\| x_i - y_j \right\|_2 \tag{A.3}$$

where $\wp(N,M)$ is the set of all possible uniform *transport plans* from a point cloud of N points to one of M points, i.e. $\wp(N,M)$ is the set of all $N \times M$ matrices with real coefficients larger than or equal to 0, such that the sum of each line equals $1/N$ and the sum of each column equals $1/M$.

The high computational cost of EMD implies that it is mostly used for validation only, and in an approximated form. On ShapeNet, we use the implementation from [Qi18] on point clouds normalized to unit radius sphere, and sampled with 2048 points. On Pix3D, we use the implementation from [Sun18] on point clouds normalized to bounding box $[-0.5, 0.5]^3$, and sampled with 1024 points.

F-score

The F-Score is introduced in [TRR+19], as an evaluation of distance between two object surfaces sampled as point clouds. Given a ground truth and a reconstructed surface, the F-Score at a given threshold distance d is the harmonic mean of precision and recall, with:

- **precision** being the percentage of reconstructed points lying within distance d to a point of the ground truth;
- **recall** being the percentage of ground truth points lying within distance d to a point of the reconstructed surface.

We use the F-Score as a validation metric on the ShapeNet dataset. It is applied on shapes normalized to unit radius sphere, and sampled with 10000 points. The distance threshold is fixed at 5% side-length of bounding box $[-1,1]^3$, i.e. $d = 0.1$.

Shell Intersection over Union

We introduce shell-Intersection over Union (sIoU). It is the intersection over union computed on voxelized surfaces, obtained as the binary occupancy grids of reconstructed and ground truth shapes. As opposed to volumetric-IoU which is dominated by the interior parts of the objects, sIoU accounts only for the overlap between object surfaces instead of volumes.

We use the sIoU as a validation metric on the ShapeNet dataset. The occupancy grid divides the $[-1,1]^3$ bounding box at resolution $50 \times 50 \times 50$, and is populated by shapes normalized to unit radius sphere.

A.1.2 Network details

We here present some details of the architecture and training procedure for UCLID-Net. We will make our entire code base publicly available.

3D CNN UCLID-Net uses $S = 4$ scales, and feature map F_s is the output of the s-th residual layer of the ResNet18 [HZRS16] encoder, passed through a 2D convolution with kernel size 1 to reduce its feature channel dimension before being back-projected. In the 3D CNN, $layer_4$, $layer_3$, and $layer_2$ are composed of 3D convolutional blocks, mirroring the composition of a residual layer in the ResNet18 image encoder, with:

- 2D convolutions replaced by 3D convolutions;
- 2D downsampling layers replaced by 3D transposed convolutions.

Final $layer_1$ is a single 3D convolution. Each $concat$ operation repeats depth grids twice along their single binary feature dimension before concatenating them to feature grids. Tab. A.1 summarizes the size of feature maps and grids appearing on Fig. 3.1.

Local shape regressors The last feature grid H_0 produced byt the 3D CNN is passed to two downstream Multi Layer Perceptrons (MLPs). First, a coarse voxel shape is predicted by MLP occ. Then, within each predicted occupied voxel, a local patch is folded in the manner of AtlasNet [GFK$^+$18], by MLP $fold$. Both MLPs locally process each voxel of H_0 independently.

First, MLP occ outputs a surface occupancy grid $\widetilde{O}$ such that

$$\widetilde{O}_{xyz} = occ((H_0)_{xyz}) \tag{A.4}$$

at every voxel location (x, y, z). $\widetilde{O}$ is compared against ground truth occupancy grid O using binary cross-entropy:

$$\mathscr{L}_{BCE}(\widetilde{O}, O) = -\sum_{xyz} \left[O_{xyz} \cdot log(\widetilde{O}_{xyz}) + (1 - O_{xyz}) \cdot log(1 - \widetilde{O}_{xyz}) \right] \tag{A.5}$$

$\mathscr{L}_{BCE}$ provides supervision for training the 2D image encoder convolutions, the 3D decoder convolutions and MLP occ.

Then $fold$, the second MLP learns a 2D parametrization of 3D surfaces within voxels whose predicted occupancy is larger than a threshold τ. As in [GFK$^+$18, YFST18], such learned parametrization is physically explained by folding a flat sheet of paper (or a patch) in space. It continuously maps a discrete set of 2D parameters $(u, v) \in \Lambda$ to 3D points in space. A patch

Nature	Name	Spatial resolution	Number of features
input image	I	224×224	3
2D feature maps	F_1	56×56	30
	F_2	28×28	30
	F_3	14×14	30
	F_4	7×7	290
2D feature grids	G^{F_1}	28×28×28	30
	G^{F_2}	28×28×28	30
	G^{F_3}	14×14×14	30
	G^{F_4}	7×7×7	290
3D depth grids	G_1^D	28×28×28	1 (binary)
	G_2^D	28×28×28	
	G_3^D	14×14×14	
	G_4^D	7×7×7	
3D CNN outputs	H_0	28×28×28	40
	H_1	28×28×28	73
	H_2	28×28×28	73
	H_3	14×14×14	146

Table A.1 – **UCLID-Net architecture:** tensor sizes, names according to Fig. 3.1 in the main chapter.

can be sampled at arbitrary resolution. In our case, we use a single MLP whose input is locally conditioned on the value of $(H_0)_{xyz}$. The predicted point cloud $\widetilde{X}$ is defined as the union of all point samples over all folded patches:

$$\widetilde{X} = \bigcup_{\substack{xyz \\ \widetilde{O}_{xyz} > \tau}} \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} + fold(u, v | (H_0)_{xyz}) \mid (u, v) \in \Lambda \right\} \tag{A.6}$$

Notice that 3D points are expressed relatively to the coordinate of their voxel. As a result, we can explicitly restrict the spatial extent of a patch to the voxel it belongs to. We use the Chamfer-L2 pseudo-distance to compare $\widetilde{X}$ to a ground truth point cloud sampling of the shape X: $\mathscr{L}_{CD}(\widetilde{X}, X) = d_{CD_2}(\widetilde{X}, X)$.

$\mathscr{L}_{CD}$ provides supervision for training the 2D image encoder convolutions, the 3D decoder convolutions and MLP $fold$. The total loss function is a weighted combination of the two losses $\mathscr{L}_{BCE}$ and $\mathscr{L}_{CD}$. Practically, for training each patch of $\widetilde{X}$ is sampled with $|\Lambda| = 10$ uniformly sampled parameters, and X is composed of 5000 points.

Pre-training UCLID-Net is first trained for one epoch using the occupancy loss $\mathscr{L}_{BCE}$ only.

Normalization layers In the ResNet18 that serves as our image encoder, we replace the batch-normalization layers by instance normalization ones. We empirically found out this provides greater stability during training, and improves final performance.

Regressing depth maps We slightly adapt the off-the-shelf network architecture used for regressing depth maps [Che18]. We modify the backbone CNN to be a ResNet18 with instance normalization layers. Additionally, we perform less down-sampling by removing the initial pooling layer. As a result the input size is 224 × 224 and the output size is 112 × 112.

Regressing cameras We similarly adapt the off-the-shelf network architecture used for regressing cameras in [XWC$^+$19]: the backbone VGG is replaced by a ResNet18 with instance normalization layers.

A.1.3 Per-category results on ShapeNet

We here report per-category validation metrics for UCLID-Net and baseline methods: AtlasNet [GFK$^+$18] (AN), Pixel2Mesh$^+$ [WZL$^+$18] (P2M$^+$), Mesh R-CNN [GMJ19] (MRC), DISN [XWC$^+$19] and UCLID-Net (ours).

Tab. A.2 reports Chamfer-L2 validation metric, Tab. A.3 the Earth Mover's Distance, Tab. A.4 the Shell Intersection over Union and Tab. A.5 the F-Score at 5% distance threshold (ie. $d = 0.1$).

	category													
method	plane	bench	box	car	chair	display	lamp	speaker	rifle	sofa	table	phone	boat	mean
AN	10.6	15.0	30.7	10.0	11.6	17.3	17.0	22.0	6.4	11.9	12.3	12.2	10.7	13.0
P2M$^+$	11.0	4.6	**6.8**	5.3	6.1	8.0	11.4	**10.3**	**4.3**	6.5	**6.3**	**5.0**	7.2	7.0
MRC	12.1	7.5	9.7	6.5	8.9	9.3	14.0	13.5	5.7	7.7	8.1	6.9	8.6	9.0
DISN	6.3	6.6	11.3	5.3	9.6	8.6	23.6	14.5	4.4	6.0	12.5	5.2	7.8	9.7
Ours	**5.3**	**4.2**	7.4	**4.1**	**4.7**	**6.9**	**10.9**	13.8	5.8	**5.7**	6.9	6.0	**5.0**	**6.3**

Table A.2 – **Chamfer-L2 Distance** (CD, $\times 10^3$) for single view reconstructions on ShapeNet Core, with various methods, computed on shapes scaled to fit unit radius sphere, sampled with 2048 points. The lower the better.

	category													
method	plane	bench	box	car	chair	display	lamp	speaker	rifle	sofa	table	phone	boat	mean
AN	6.3	7.9	9.5	8.3	7.8	8.8	9.8	10.2	6.6	8.2	7.8	9.9	7.1	8.0
P2M$^+$	4.4	3.2	3.4	3.4	3.7	3.7	5.5	4.2	3.5	3.4	3.8	2.7	3.4	3.8
MRC	5.0	4.1	5.1	4.1	4.7	4.9	5.6	5.7	4.1	4.6	4.5	4.6	4.2	4.7
DISN	**2.2**	2.3	3.2	2.4	2.8	**2.5**	3.9	**3.1**	**1.9**	**2.3**	2.9	**1.9**	2.3	2.6
Ours	2.5	**2.2**	**3.0**	**2.2**	**2.3**	**2.5**	**3.2**	3.4	2.0	2.4	**2.7**	2.2	**2.2**	**2.5**

Table A.3 – **Earth Mover's Distance** (EMD, $\times 10^2$) for single view reconstructions on ShapeNet Core, with various methods, computed on shapes scaled to fit unit radius sphere, sampled with 2048 points. The lower the better.

	category													
method	plane	bench	box	car	chair	display	lamp	speaker	rifle	sofa	table	phone	boat	mean
AN	20	13	7	16	13	12	14	8	28	11	15	14	17	15
P2M$^+$	31	34	23	26	28	28	28	20	42	24	33	35	34	30
MRC	24	26	18	22	21	23	21	16	33	19	27	28	27	24
DISN	40	33	20	31	25	**33**	21	19	**60**	29	25	**44**	34	30
Ours	**41**	**41**	**29**	**34**	**36**	**33**	**37**	**24**	51	**31**	**38**	43	**37**	**37**

Table A.4 – **Shell-Intersection over Union** (IoU, %) for single view reconstructions on ShapeNet Core, with various methods, computed on voxelized surfaces scaled to fit unit radius sphere. The higher the better.

	category													
method	plane	bench	box	car	chair	display	lamp	speaker	rifle	sofa	table	phone	boat	mean
AN	91.2	85.9	73.8	94.4	90.5	84.3	81.4	79.7	95.6	91.1	90.8	90.4	90.3	89.3
P2M$^+$	90.3	97.1	**96.0**	97.9	95.7	93.1	90.2	**91.3**	96.8	96.5	**95.8**	**97.6**	94.4	95.0
MRC	88.4	93.3	92.1	96.4	92.0	91.4	85.8	88.3	94.9	95.0	93.9	95.9	92.8	92.5
DISN	94.4	94.3	88.8	96.2	90.2	91.8	77.9	85.4	96.3	95.7	86.6	96.4	93.0	90.7
Ours	**96.1**	**97.5**	94.3	**98.5**	**97.4**	**95.8**	**92.7**	90.6	**98.0**	**97.0**	95.5	96.4	**97.1**	**96.2**

Table A.5 – **F-Score** (%) at threshold $d = 0.1$ for single view reconstructions on ShapeNet Core, with various methods, computed on shapes scaled to fit unit radius sphere, sampled with 10000 points. The higher the better.

A.2 Supplementary material for DeepMesh

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.

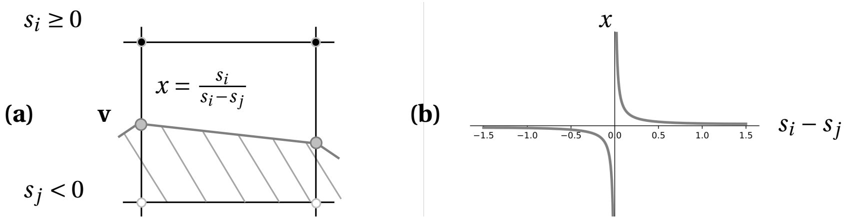


Figure A.1 – **Marching cubes differentiation.** **(a)** Marching Cubes determines the relative position x of a vertex $\mathbf{v}$ along an edge via linear interpolation. This does not allow for effective back-propagation when topology changes because of a singularity when $s_i = s_j$. **(b)** We plot x, relative vertex position along an edge. Note the infinite discontinuity for $s_i = s_j$.

A.2.1 Non-differentiability of Marching Cubes

The Marching Cubes (MC) algorithm [LC87] 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 is determined as the zero crossing of the interpolant of s_i, s_j, that is,

$$x = \frac{s_i}{s_i - s_j}, \tag{A.7}$$

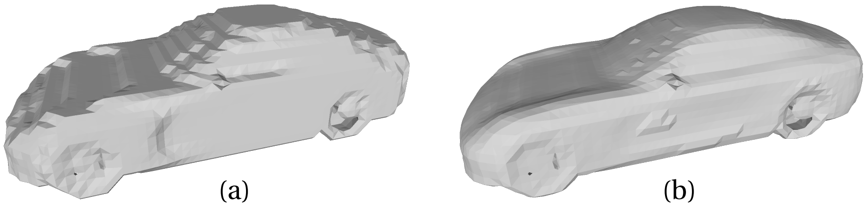

(a) (b)

Figure A.2 – **Marching Cubes on an occupancy field.** **(a)** running Marching Cubes on the occupancy field predicted by f_Θ yields artefacts. **(b)** Using an inverse sigmoid function to amplify the predicted occupancy values yields smoother shapes.

as shown in Fig. A.1(a). The vertex location is then taken to be

$$\mathbf{v} = \mathbf{G}_i + x(\mathbf{G}_j - \mathbf{G}_i). \tag{A.8}$$

Unfortunately, this function is discontinuous for $s_i = s_j$, as illustrated in Fig A.1(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 [LDG18].

A.2.2 Meshing an occupancy field

As shown in Fig. A.2(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 [MON+19]. 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. A.2(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.

A.2.3 Failure Case: Vanishing Surface

As explained in the main chapter, end-to-end training cannot be done from scratch, because our gradients require the implicit field to already represent a valid surface. Here, we examine another failure case that arises when training end-to-end with unsufficient regularization.

We start with the network f_{Θ_1} of Sec 4.4.1, 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 $\mathscr{B}$. We do so by using our gradients and applying a surface to surface distance directly on the output

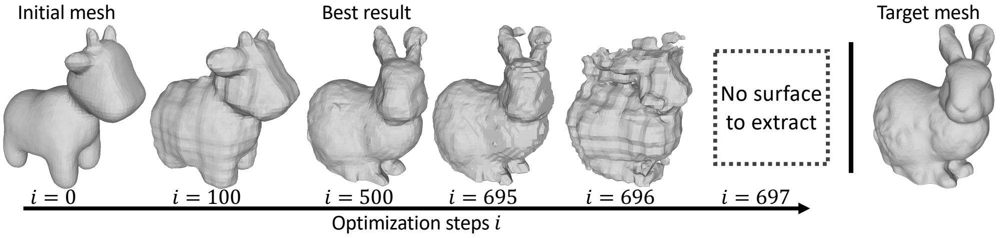


Figure A.3 – **Limitation of end-to-end training:** In the absence of regularization, optimizing the network weights Θ and latent code $\mathbf{z}$ using our gradients for an extended number of iterations to fit an unseen shape can lead to divergence in the optimization process.

mesh $\mathcal{M}$, with no other form of regularization or supervision of the implicit field. We minimize

$$\mathcal{L}_{\text{task1}}(\mathcal{M}_{\Theta_1}(\mathbf{z})) = \min_{m \in \mathcal{M}} d(m, \mathcal{B}) + \min_{b \in \mathcal{B}} d(\mathcal{M}, b)\ . \tag{A.9}$$

with respect to $\mathbf{z}$ and Θ_1 with Adam [KB15a] for 800 steps, where d is the point-to-surface distance in 3D.

Figure A.3 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 useen shape with a weak prior turned the field into an invalid signed distance, and failed. To avoid this issue, regularization techniques can be employed, including:

- early stopping (Fig. A.3 for $i = 500$) ;
- initializing the network weights to some valid implicit field by pretraining on a dataset and keeping them frozen (as in Sec. 4.4.1 to 4.4.5), or initializing the network weights to match a 3D sphere (as in [MCR22]) ;
- simultaneously supervising both the implicit field and the mesh surface, as in Sec. 4.4.6 ;
- adding consistency terms for the implicit field, such an eikonal regularization on the gradients (as in [AL20a]).

A.2.4 Comparing against Deep Marching Cubes

Deep Marching Cubes (DMC) [LDG18] 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. A.4, 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

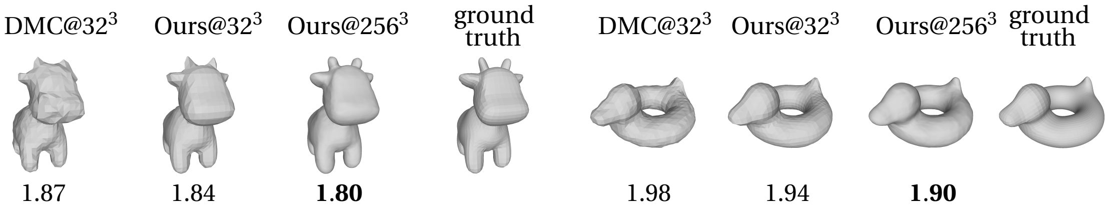


Figure A.4 – **Comparison to Deep Marching Cubes** considering a latent space of size 2. The metric we report is the Chamfer distance (CHD$\cdot 10^2 (\downarrow)$), evaluated on 5000 samples for unit sphere normalized shapes.

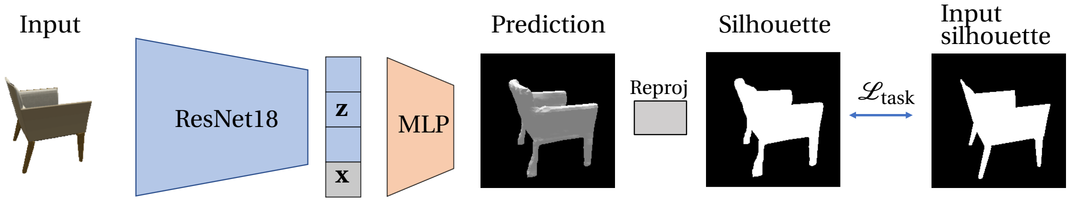


Figure A.5 – **Shilouette-driven refinement.** At inference time, given an input image, we exploit the differentiability of *DeepMesh* to refine the predicted surface so that to match input silhouette in image space through Differentiable Rasterization [KUH18] or contour matching [GRYF21].

the grid. By contrast, deep implicit fields are not resolution limited. Hence, they can better capture high frequency details.

A.2.5 Single view 3D Reconstruction

We first provide additional details on the Single view 3D Reconstruction pipeline presented in the main chapter. Then, for each experimental evaluation of the main chapter, we first introduce metrics in details, and then provide additional qualitative results.

Architecture

Fig A.5 depicts our full pipeline. As in earlier work [MON+19, CZ19], we condition our deep implicit field architecture on the input images via a residual image encoder [HZRS16], which maps input images to latent code vectors **z**. Specifically, our encoder consists of a ResNet18 network, where we replace batch-normalization layers with instance normalization ones [UVL16] 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 [PFS+19]. We train this architecture, which we dub *DeepMesh* (raw), by minimizing $\mathscr{L}_{\text{imp}}$ (Eq.1 on the main manuscript) wrt. Θ on a training set of image-surface

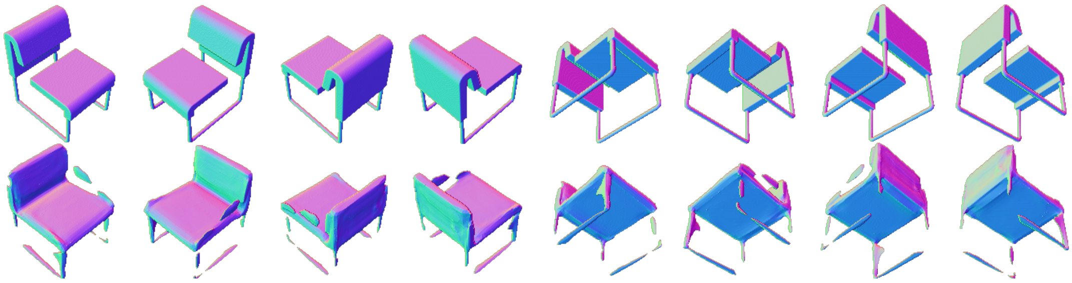


Figure A.6 – **Normal consistency score:** we render target (top row) and reconstructed shapes (bottom row) normal maps from 8 fixed viewpoints, and take the average of pixelwise cosine similarity to compute an image based normal consistency score.

pairs.

At inference time, we exploit end-to-end differentiability to refine predictions as depicted in Fig A.5.

Evaluation on ShapeNet.

Recent work [TRR+19] 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 [XWC+19] 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

$$\mathrm{CHD}(P,Q) = \sum_{\mathbf{p}\in P} \min_{\mathbf{q}\in Q} \|\mathbf{p}-\mathbf{q}\|_2^2 + \sum_{\mathbf{q}\in Q} \min_{\mathbf{p}\in P} \|\mathbf{p}-\mathbf{q}\|_2^2. \tag{A.10}$$

  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. A.6, 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 $\widetilde{\mathbf{n}}_i$, we have

$$\mathrm{NC} = \frac{1}{8}\sum_{i=1}^{i=8} \frac{1}{|\mathbf{n}_i \cap \widetilde{\mathbf{n}}_i|} \sum_{(u,v)\in \mathbf{n}_i \cap \widetilde{\mathbf{n}}_i} \frac{\mathbf{n}_i[u,v]\cdot \widetilde{\mathbf{n}}_i[u,v]}{\|\mathbf{n}_i[u,v]\|\,\|\widetilde{\mathbf{n}}_i[u,v]\|}. \tag{A.11}$$

Additional Qualitative Results

We provide additional qualitative comparative results for ShapeNet in Fig. A.7. Fig A.8 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 [MMNKF18, OKC+21].

A.2.6 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. A.9, and additional optimization results are shown in Fig. A.12.

Dataset

As described in the main manuscript, we consider the car split of the ShapeNet [CFG+15] dataset for this experiment. Since aerodynamic simulators typically require high quality surface triangulations to perform CFD simulations reliably, we (1) follow [SSB13] 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 [JJT+07], setting an *inflow velocity* of 15 meters per second and airflow *density* equal 1.18. Each simulation was run 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. A.10.

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

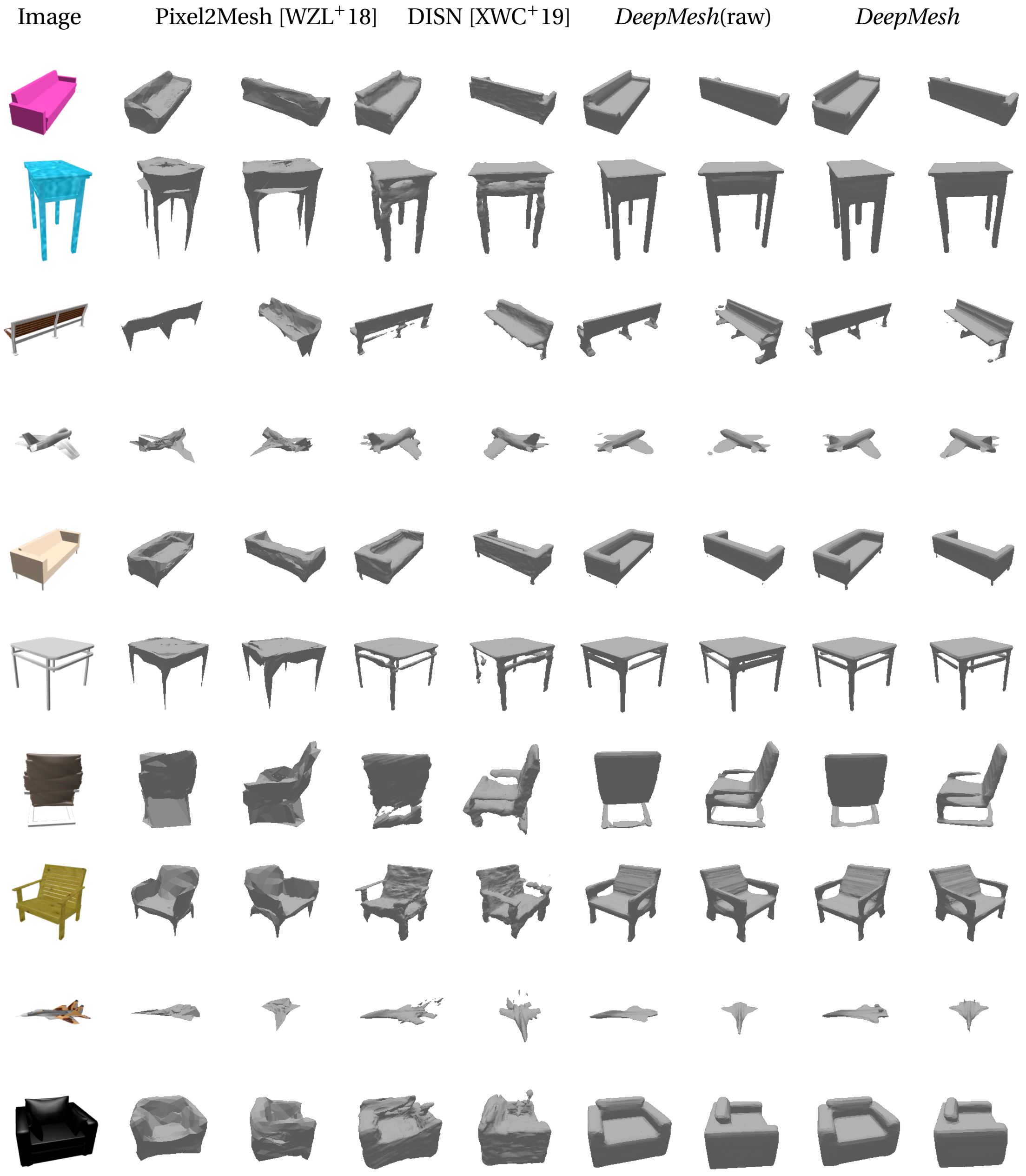


Figure A.7 – **Comparative results for SVR on ShapeNet.**

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 [BRFF18] and replaced Geodesic Convolutions [MBM+17] by Spline ones [FLWM18] for efficiency. The predicted and simulated pressure values are compared in Fig. A.10.

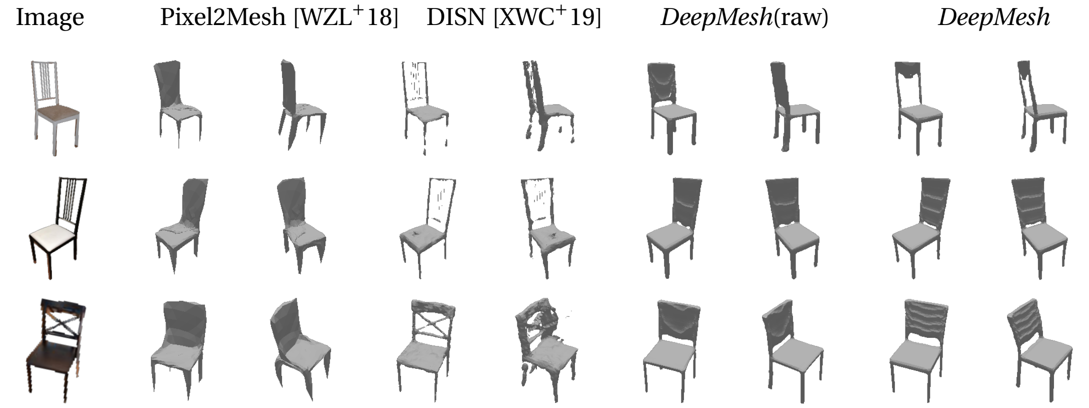


Figure A.8 – **Failure cases for SVR on Pix3D.** Reconstruction refinement based on L_1 silhouette distance or chamfer matching fails to capture fine topological details for challenging samples.

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 [BRFF18], where it led to good design performances. In our experiments we are using the parameterization described in [BRFF18] with only a small modification: to enforce the car left and right sides to be symmetrical we square sinuses in the corresponding terms. We also add l_2-norm of the parameterization vector to the loss as a regularization.
- **PolyCube** Inspired by [UB18] we create a grid of control points to change the mesh. The grid size is 8 × 8 × 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 ($\sigma = 1$,

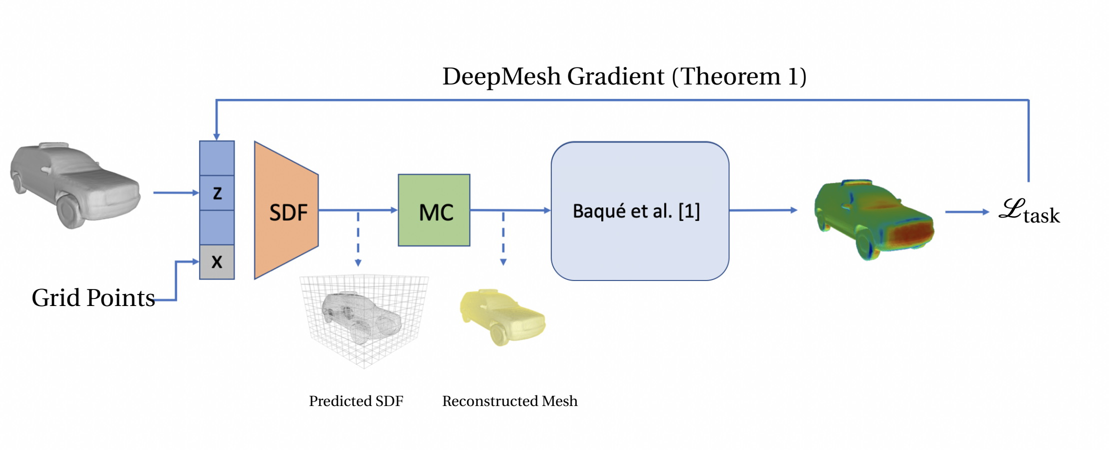


Figure A.9 – **Aerodynamic optimization pipeline.** We encode a shape we want to optimize using *DeepSDF* (denoted as **SDF** block on the figure) and obtain latent code $\mathbf{z}$. Then we start our iterative process. First, we assemble an Euclidean grid and predict SDF values for each node of the 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 z and do an optimization step. The loop is repeated until convergence.

kernel size = 3). This results in a parameterization that allows for deformations that are very similar to the one of [UB18].

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. A.11 to preserve space for the driver and engine and define a loss term $\mathscr{L}_{\text{constraint}}$. To avoid generating unrealistic designs, we also introduce an additional regularization term $\mathscr{L}_{\text{reg}}$ that prevents $\mathbf{z}$ from straying to far away from known designs. We take it to be

$$\mathscr{L}_{\text{reg}} = \alpha \sum_{\mathbf{z}' \in \mathcal{Z}_k} \frac{||\mathbf{z} - \mathbf{z}'||_2^2}{|\mathcal{Z}_k|}, \tag{A.12}$$

where $\mathcal{Z}_k = \mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_k$ denote the k closest latent vectors to $\mathbf{z}$ from the training set. In our experiments we set $k = 10$, $\alpha = 0.2$. Minimizing $\mathscr{L}_{\text{reg}}$ 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 MeshSDF, even though we use the Adam optimizer [KB15b] , which is known for its ability

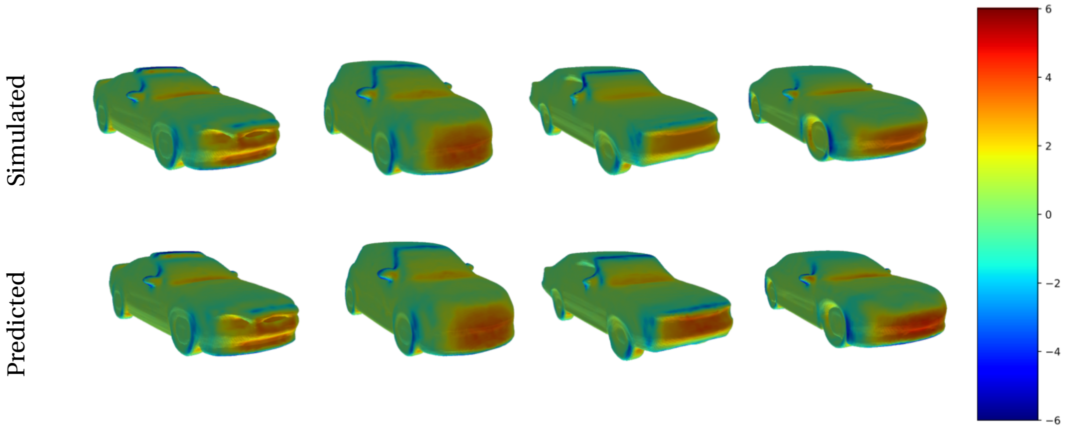


Figure A.10 – **Simulated and predicted pressure fields.** Pressure fields for different shape simulated with OpenFoam (top) and predicted by a Convolutional Neural Network (bottom).

to escape some of them. We are planning to address this problem more thoroughly in future.

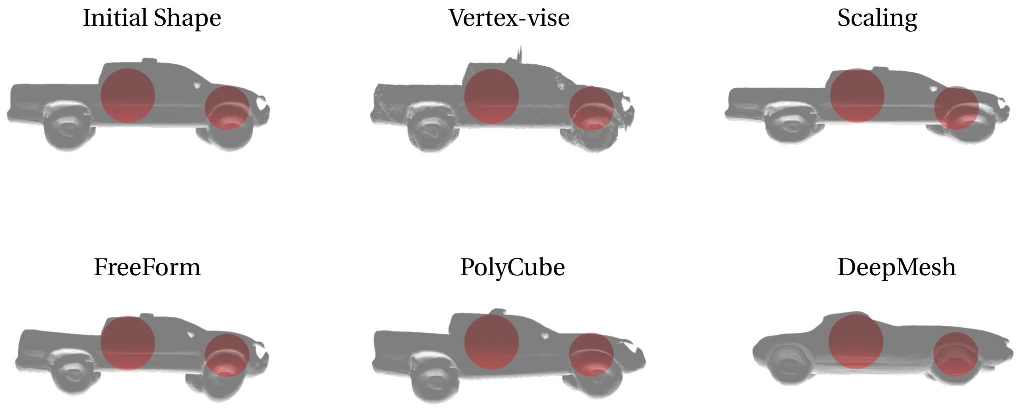


Figure A.11 – **Preserving space for the driver and engine.** We define a loss function $\mathscr{L}_{\text{constraint}}$ that forces the reconstructed shape to contain the red spheres. The spheres are shown overlaid on the initial shape and then on the various results. Because the constraints are soft, they can be slightly violated.

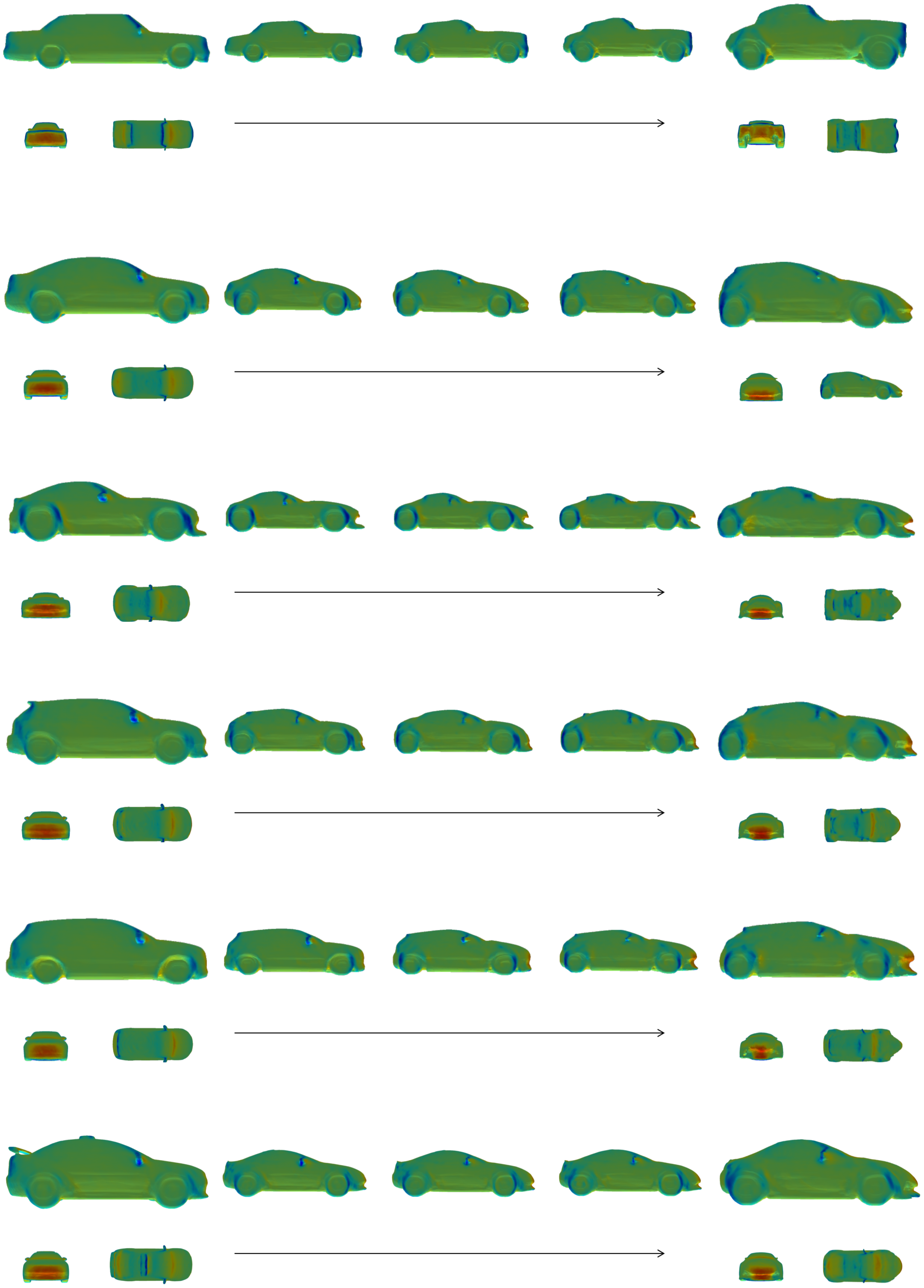


Figure A.12 – **DeepMesh aerodynamic optimizations.**

A.3 Supplementary material for Sketch2Mesh

A.3.1 External Contours

Our 2D Chamfer refinement objective for matching external contours requires an estimation of these contours. We here describe the simple algorithms we use to get them for the reconstructed mesh, and for the full input sketch.

External Contours of Reconstructed Shapes

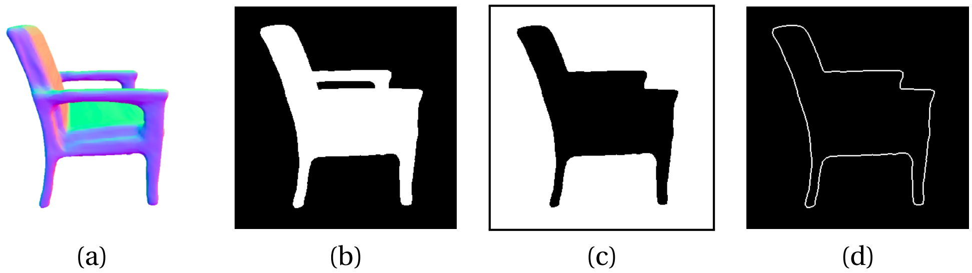


Figure A.13 – **External contours of reconstructed shape: (a)** Initially reconstructed shape **(b)** Rendered foreground/background mask, **(c)** Flood-filling (b) from one image corner, and performing 1 pixel dilation of the flood-filled background **(d)** Taking the pixel-wise multiplication of (b) and (c) yields an exterior contour image, in which internal holes are ignored (the armrest for example here).

Given mesh $\mathcal{M}_\Theta$ and projection Λ, we render a $H \times W$ foreground/background mask. Then we flood fill the background, starting from one image corner, and apply morphological dilation to the result. As depicted on Fig. A.13, taking the pixel-wise multiplication of this dilated flood filled background with the original foreground/background mask yields an external contour that ignores inner holes. We use this image for $\widetilde{\mathbf{F}}$ in Section 5.3.3.

External Contours of Input Sketches

Given an input sketch, we cannot apply the above method since line drawings might not be watertight. Instead, we apply an image-space only algorithm that extracts external contours, which can then be matched against the ones of the initial reconstruction.

As pictured in Fig. 5.2(c), we propose to do this by shooting rays from the image borders, at multiple angles, and only preserve the first encountered stroke for each ray. For the ease of implementation, in practice we shoot rays that are perpendicular to the image borders, but rotate the input image of $\pm \{0, 10, 20, 30, 35, 40, 45\}$ degrees and aggregate the resulting pixels at each angle. This is depicted in Fig. A.14.

To achieve a relative invariance to pen size (free choice in our interface), we extract both the entry and exit pixels of the first pen stroke a ray encounters. In case the average distance over

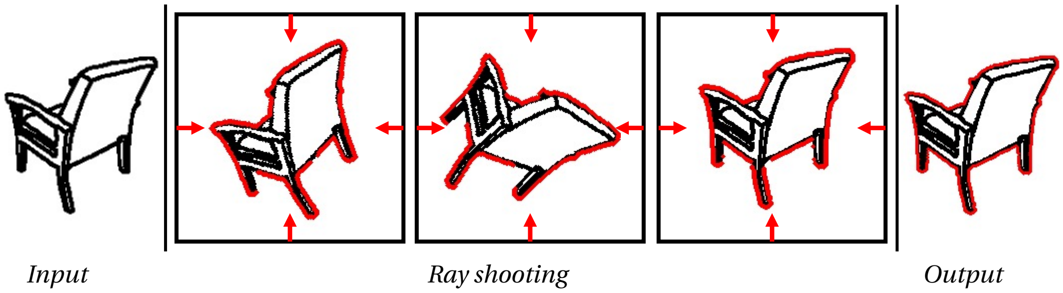


Figure A.14 – **External contours of input sketch:** We rotate the input sketch at various angles and shoot vertical and horizontal rays to only keep the first encountered pen stroke (red pixels). These pixels obtained at different rotation angles are then aggregated to yield the full external contour (shown in red on the last panel, superposed to the sketch).

the whole image between the entry and exit pixels is greater than a threshold, we heuristically consider the line as thick and only keep the exit pixels - this corresponds to the inner shell of the external contour. Otherwise, we consider the line as thin, and keep the entry pixel.

A.3.2 Comparison of the two Refinement Approaches

Training on `SketchFD`

Metric	Method	**Cars**, test drawing style:			**Chairs**, test drawing style:		
		Suggestive	SketchFD	Hand-drawn	Suggestive	SketchFD	Hand-drawn
CD-$l_2 \cdot 10^3$ ↓	*Initial*	3.231	1.815	2.534	12.290	7.770	17.395
	Sketch2Mesh/Render	2.538	**1.515**	2.054	10.761	**6.517**	16.091
	Sketch2Mesh/Chamfer	**2.419**	1.516	**2.047**	**9.524**	6.737	**12.585**
Normal Consistency ↑	*Initial*	89.67	90.94	89.06	76.76	80.49	63.11
	Sketch2Mesh/Render	90.92	**92.34**	91.02	80.39	**84.43**	68.67
	Sketch2Mesh/Chamfer	**91.23**	92.09	**91.03**	**81.00**	83.10	**70.49**

Table A.6 – **Cars and Chairs.** Reconstruction metrics when using the encoding/decoding network trained on `SketchFD` synthetic sketches of cars and of chairs, and tested on all 3 datasets. We show *initial* results before refinement and then using our two refinement methods. Note that *Sketch2Mesh/Chamfer* does better than *Sketch2Mesh/Render* on the styles it has *not* been trained for, indicating a greater robustness to style changes.

In the main paper, in Sec. 5.4.3 we present a comparison of *Sketch2Mesh/Render* and *Sketch2Mesh/Chamfer* approaches when applied on networks trained on `Suggestive` synthetic sketches, and tested on all 3 datasets. In Tab. A.6 we present the same comparison, but this time for encoder/decoder pairs trained on `SketchFD`. Again, *Sketch2Mesh/Chamfer* appears to be more robust to style change and performs better than *Sketch2Mesh/Render* on datasets the latter has not been trained on.

Gradients and Sensitivity to Thin Components

In Fig. A.15, we demonstrate how *Sketch2Mesh/Chamfer* is more sensitive to thin shape components such as chair legs. Indeed, removing a thin component only affects $\mathscr{L}_{F/B}$ of a few pixels, whereas $\mathscr{L}_{CD}$ takes into account the distance and spatial extent of the deformation.

Differentiable Rasterization Hyperparameters

In Figure A.16, we show that the the choice of hyper-parameters in the differentiable rendering process can deeply influence the refinement behavior of *Sketch2Mesh/Render*, particularly when the predicted binary masks are not accurate. Specifically, we investigate the importance of parameter `faces_per_pixel`, controlling how many triangles are used for backpropagation within each pixel: using a higher number of triangles will result in smoother gradients, as binary mask information is back-propagated to more faces. As depicted in Figure A.16 this is particularly beneficial when predicted binary masks are not accurate or noisy, and results in more accurate reconstructions. In practice, we set `faces_per_pixel=25` in our experiments.

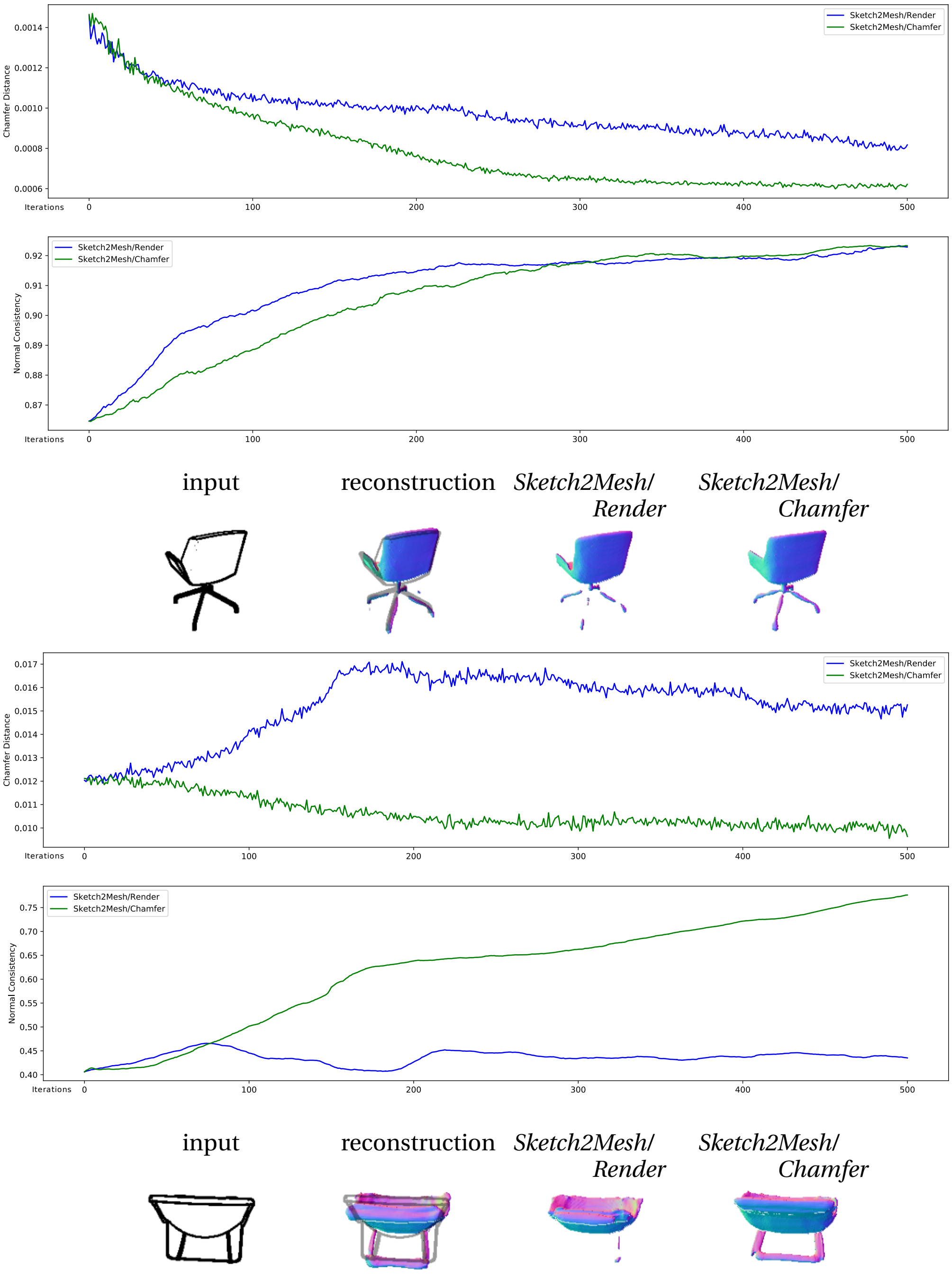


Figure A.15 – **Refinement.** *Sketch2Mesh/Chamfer* is more sensitive to thin shape components such as chair legs with respect to *Sketch2Mesh/Render* : this is due to the nature of the loss, penalizing chamfer distance between silhouettes, rather than per-pixel discrepancies. Best seen digitally, zoomed in.

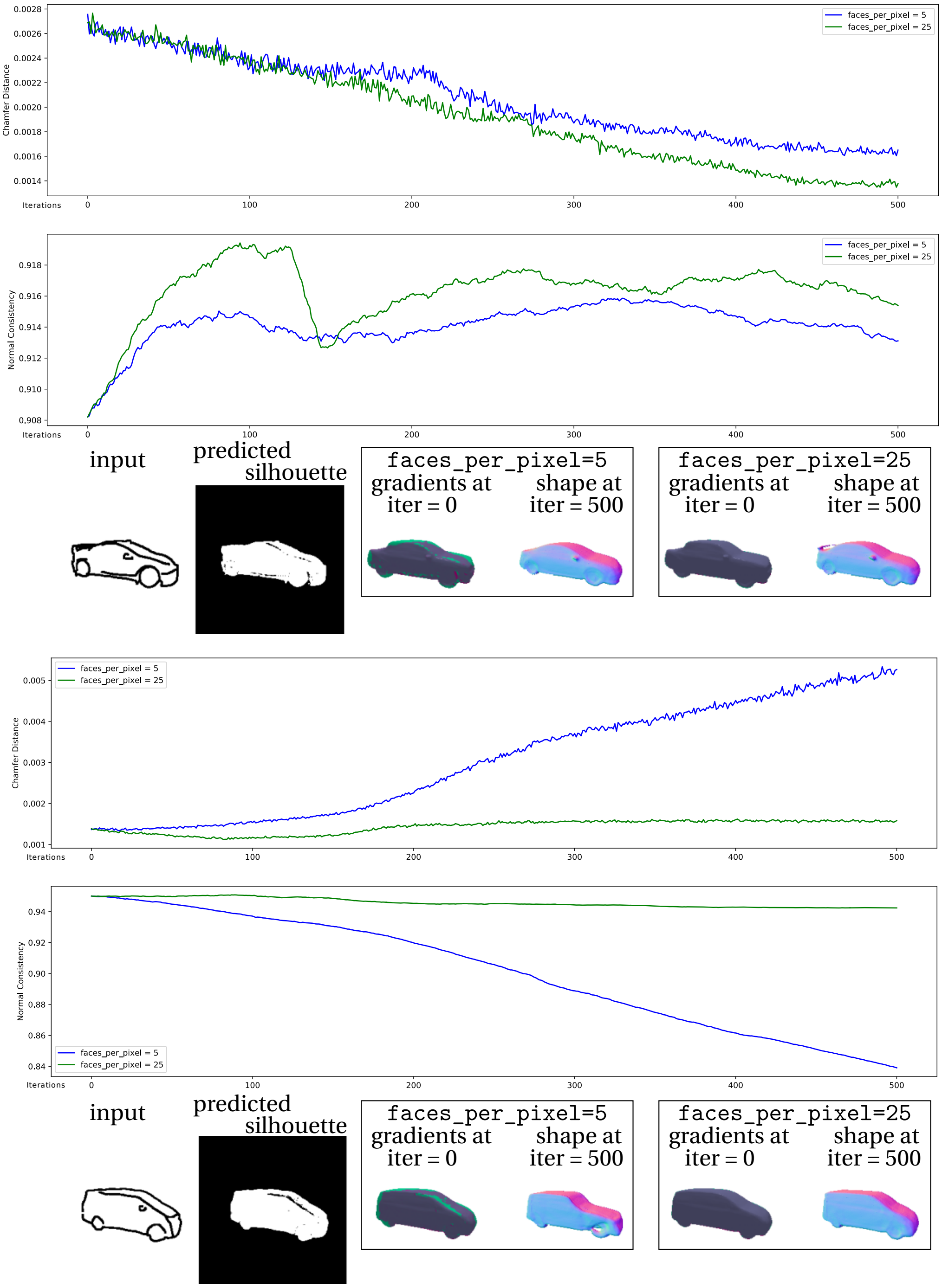


Figure A.16 – **Silhouette Alignment.** We compare different settings of pytorch3D [RRN+20] on two human-drawn car samples. Surface gradients are shown in color (green = intrusion along the surface normal, purple = extrusion).When considering a lower number of triangles for backpropagation of the rasterization process, gradients are more influenced by erroneous silhouette predictions, making refinement less effective (top) or even detrimental (bottom). Best seen digitally, zoomed in.

A.4 Supplementary material for MeshUDF

In Sec. A.4.1 we describe the procedure used to train the UDF network ϕ_θ on MGN garments. In Sec. A.4.2 we explain the metrics used in the experiment section of the corresponding chapter. In Sec. A.4.3 we explain in more details the gradients introduced in the main paper. In Sec. A.4.4 we show experimental evidence that artifacts appearing at high resolution are caused by the UDF field being approximated. In Sec. A.4.5 we demonstrate the benefits of the voting scheme for establishing pseudo-signs. In Sec. A.4.6, we show the fitting to sparse pointclouds of Sec. 6.4.3 can be initialized from random latent codes.

A.4.1 Network Training

We train one auto-decoder [PFS$^+$19] network ϕ_θ to approximate the UDF field of a garment collection. We use the dataset from MGN [BTTPM19] consisting of 328 meshes from which we keep 300 instances for training and 28 for testing.

To generate UDF supervision sample points and values, meshes are scaled to fit a sphere of radius 0.8, and for each mesh i we generate N training samples $(\mathbf{p}_{i,j}, d_{i,j}) \in \mathbb{R}^3 \times \mathbb{R}^+$ where $d_{i,j}$ is the minimum between d_{max} and the distance from 3D point $\mathbf{p}_{i,j}$ to the i-th shape. We clamp UDF values at $d_{max} = 0.1$ to avoid wasting the network's capacity on learning a precise field away from the surface as in [PFS$^+$19, CMPM20]. We pick $N = 30000$ and sample 6000 points uniformly on the surface, 12000 within a distance 0.05, 8000 within a distance 0.3 and 4000 within the bounding box of side length 2. As opposed to SDF values, unsigned distances $d_{i,j}$ can be computed directly from raw triangle soups with standard software [Daw], and do not require any pre-processing of the meshes.

ϕ_θ is implemented by a 9-layer MLP with 512 hidden dimensions and ReLU activation functions. It uses Fourier positional encoding of fifth order on the 3D coordinate inputs [SMB$^+$20]. We jointly optimize the network's weights θ with one latent vector embedding $\mathbf{z}_i \in \mathbb{R}^{128}$ per training shape i by minimizing the L_1 loss between the predicted and target UDF values, with a regularization of strength $\lambda = 10^{-4}$ on the norm of the latent codes. With $\mathcal{T}$ as the training set, the full loss is

$$\mathcal{L} = \frac{1}{|\mathcal{T}| \cdot N} \sum_{i \in \mathcal{T}} \left[\sum_{j=1}^{N} \left| \phi_\theta(\mathbf{z}_i, \mathbf{p}_{i,j}) - d_{i,j} \right| + \lambda \, \|\mathbf{z}_i\|_2 \right]$$

and is minimized using Adam [KB15b] for 2000 epochs.

A.4.2 Metrics

Given a reconstructed mesh $\widetilde{M}$ and a set A of points on its surface, along with a ground-truth mesh M and a set B of points on its surface, we compute the following metrics:

- **Chamfer distance.** We take it to be

$$
\begin{aligned}
\mathrm{CHD}(\widetilde{M}, M) = {} & \frac{1}{|A|} \sum_{a \in A} \min_{b \in B} \|a - b\|^2 \\
& + \frac{1}{|B|} \sum_{b \in B} \min_{a \in A} \|a - b\|^2 \,,
\end{aligned}
\tag{A.13}
$$

  the sum of the average distance of each point in A to B and the average distance of each point in B to A.

- **Image consistency.** Let K be a set of 8 cameras located at the vertices of a cuboid encompassing the garments looking at its centroid. For each $k \in K$ we render the corresponding binary silhouette $S_k \in \{0,1\}^{256 \times 256}$ (respectively $\widetilde{S}_k$) and normal map $N_k \in \mathbb{R}^{256 \times 256 \times 3}$ (resp. $\widetilde{N}_k$) of mesh M (resp. $\widetilde{M}$). Then we define the image consistency between $\widetilde{M}$ and M as

$$
\mathrm{IC}(\widetilde{M}, M) = \tfrac{1}{|K|} \sum_{k \in K} \mathrm{IoU}(\widetilde{S}_k, S_k) * \mathrm{COS}(\widetilde{N}_k, N_k)\,,
\tag{A.14}
$$

  where IoU is the intersection-over-union of two binary silhouettes and COS is the average cosine-similarity between two normal maps. Both can be written as

$$
\mathrm{IoU}(\widetilde{S}, S) = \sum_{u=1}^{H} \sum_{v=1}^{W} \widetilde{S}_{u,v} S_{u,v}
\tag{A.15}
$$

$$
\cdot \left[\sum_{u=1}^{H} \sum_{v=1}^{W} max(\widetilde{S}_{u,v} + S_{u,v}, 1) \right]^{-1} \,,
\tag{A.16}
$$

$$
\mathrm{COS}(\widetilde{N}, N) = \tfrac{1}{HW} \sum_{u=1}^{H} \sum_{v=1}^{W} \frac{\widetilde{N}_{u,v} \cdot N_{u,v}}{\left\|\widetilde{N}_{u,v}\right\| \left\|N_{u,v}\right\|} \,,
$$

  with H and W the image height and width, $S_{u,v} \in \{0,1\}$ the binary pixel value at coordinate (u,v) of S and $N_{u,v} \in \mathbb{R}^3$ the color pixel value at coordinate (u,v) of N.

- **Normal consistency.** We take it to be

$$
\begin{aligned}
\mathrm{NC}(\widetilde{M}, M) = {} & \tfrac{1}{|A|} \sum_{a \in A} \left| \cos[\tilde{\mathbf{n}}(a), \mathbf{n}(\operatorname*{argmin}_{b \in B} \|a - b\|^2)] \right| \\
& + \tfrac{1}{|B|} \sum_{b \in B} \left| \cos[\mathbf{n}(b), \tilde{\mathbf{n}}(\operatorname*{argmin}_{a \in A} \|a - b\|^2)] \right| \,,
\end{aligned}
\tag{A.17}
$$

  the average *unsigned* cosine-similarity between the normals of pairs of closest point in A and B, where $\tilde{\mathbf{n}}(x)$ denotes the normal at point x.

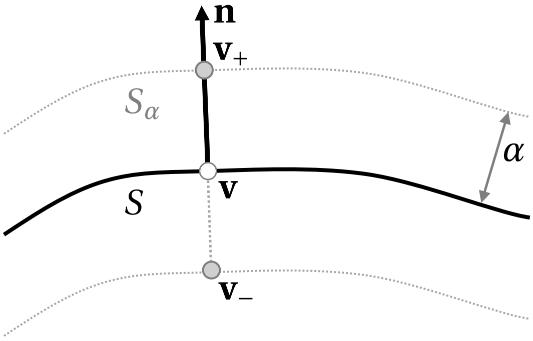


Figure A.17 – **Iso-surface differentiation**: S is the minimum-level set of UDF $\phi(\mathbf{z},\cdot)$ and S_α its α-level set ($\alpha > 0$). By using already established differentiability results on $\mathbf{v}_+ \in S_\alpha$ and $\mathbf{v}_- \in S_\alpha$, we derive new derivatives for $\mathbf{v} \in S$.

A.4.3 Differentiating through Iso-Surface Extraction

In Sec. 6.3.2, we derived gradients for surface points with respect to the latent code $\mathbf{z}$. We here expand on the underlying assumptions and justify our choices.

Using the α-isolevel. Let $\alpha > 0$ be a small scalar and $\mathbf{z} \in \mathbb{R}^C$ a latent code parametrizing the UDF field $\phi(\mathbf{z},\cdot)$. We consider $\mathbf{v} \in \mathbb{R}^3$, a surface point lying within a facet of the mesh $M_\mathbf{z}$, and $\mathbf{n}$ its surface normal –defined up to its orientation. $\mathbf{v}$ lies on the 0-level set of the field, and we choose to formulate it as the following linear combination

$$\mathbf{v} = \tfrac{1}{2}(\mathbf{v}_- + \mathbf{v}_+)\,, \tag{A.18}$$

where

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

The arrangement of $\mathbf{v}$, $\mathbf{v}_+$ and $\mathbf{v}_-$ is depicted in Fig. A.17.

Both $\mathbf{v}_+$ and $\mathbf{v}_-$ are at a distance α from $\mathbf{v}$. Assuming such points to belong to the α-level set, the outwards pointing normal of $\mathbf{v}_+$ on the α-level set is $\mathbf{n}$, and the one of $\mathbf{v}_-$ is $-\mathbf{n}$, and we can use [AHY+19, RLR+20] to write

$$\frac{\partial \mathbf{v}_+}{\partial \mathbf{z}} = -\mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z},\mathbf{v}_+) \quad \text{and} \quad \frac{\partial \mathbf{v}_-}{\partial \mathbf{z}} = \mathbf{n}\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z},\mathbf{v}_-)\,. \tag{A.19}$$

and differentiating the mapping of Eq. A.18 –which we consider as fixed– yields

$$\frac{\partial \mathbf{v}}{\partial \mathbf{z}} = \frac{\mathbf{n}}{2}\left[\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z},\mathbf{v}-\alpha\mathbf{n}) - \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z},\mathbf{v}+\alpha\mathbf{n})\right]\,. \tag{A.20}$$

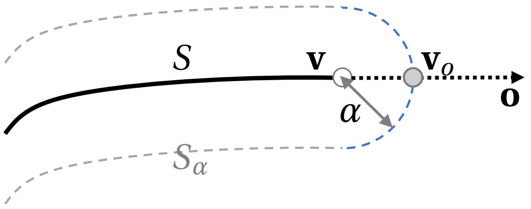


Figure A.18 – **Iso-surface differentiation at borders**: S is the minimum-level set of UDF $\phi(\mathbf{z}, \cdot)$ and S_α its α-level set ($\alpha > 0$). By using already established differentiability results on $\mathbf{v}_o \in S_\alpha$, we derive a new derivative for $\mathbf{v} \in S$.

Approximate gradients. In practice however, $\mathbf{v}_+$ and $\mathbf{v}_-$ are not guaranteed to lie on the α-level set, but can be on a β-level set with $\beta < \alpha$, in which case their normals differ from $\mathbf{n}$ and $-\mathbf{n}$. For our assumption to hold, $\mathbf{v}$ needs to be the closest point to $\mathbf{v}_+$ on the 0-level set, and similarly for $\mathbf{v}_-$, which is true when α is small compared to the surface curvature. We thus use Eqs. A.19, A.20 as approximations only.

Eq. A.20 is only flawed for points with high curvature, and still holds true for most of the points lying on unwrinkled regions of the surface. Since gradients backpropagated to the latent code are averaged over the entire surface (as in [MESK22]), a minority of them being noisy is not an issue. Sec. 6.4.3 empirically shows that using $\alpha = 0.01$ works in practice for a wide range of shapes.

Uniqueness of the mapping. Eq. A.18 is an arbitrary choice of a mapping. It is not unique, and one could instead pair $\mathbf{v}$ to other points on the α-level set, leading to a different result in Eq. A.20. We deliberately chose the 2 closest points to naturally surround $\mathbf{v}$ with its closest neighbors.

Minimizing a downstream loss. Eq. A.20 can be used to minimize downstream loss functions directly defined on mesh vertices with gradient descent. Given such a loss function $\mathscr{L}$, we use the chain rule to write

$$\frac{\partial \mathscr{L}}{\partial \mathbf{z}} = \sum_{(\mathbf{v},\mathbf{n}) \in M_{\mathbf{z}}} \frac{\partial \mathscr{L}}{\partial \mathbf{v}} \frac{\mathbf{n}}{2} \left[\frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} - \alpha \mathbf{n}) - \frac{\partial \phi}{\partial \mathbf{z}}(\mathbf{z}, \mathbf{v} + \alpha \mathbf{n}) \right] \quad .$$

We rely on the field being an UDF and move its zero level set. This is in practice enforced by freezing the network weights, which is thus acting as a strong prior on the field, and only optimize the latent code.

The case of border points. Border points do not only have 2 closest neighbors on the α-level set, but an entire semicircle as depicted in blue on Fig. A.18. In this case, we pair $\mathbf{v}$ with the

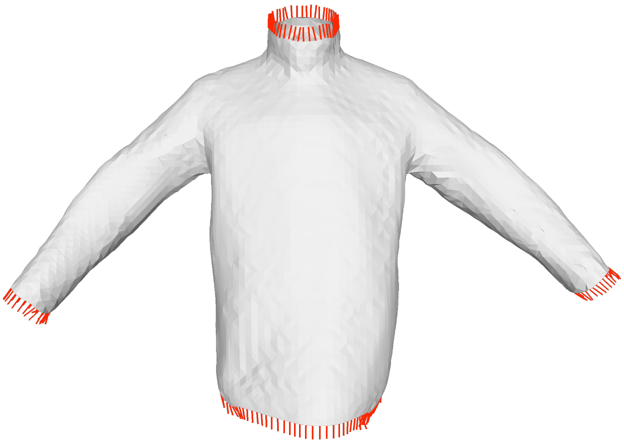


Figure A.19 – **Outwards pointing vectors**: for border vertices we define outwards pointing vectors $\mathbf{o}$ to construct derivatives allowing the surface to shrink or extend along them.

outmost point on the α-level set with

$$\mathbf{v} = \mathbf{v}_o - \alpha \mathbf{o} \,, \tag{A.21}$$

and follow the same reasoning as above. We consider $\mathbf{o}$ as a mapping direction, and thus locally fixed.

Constructing the o vectors. Fig. A.19 depicts the outwards pointing vectors $\mathbf{o}$ for one reconstructed garment. They are computed as follows. Let $\mathbf{v}$ be a vertex lying on the border, $\mathbf{n}$ be the normal vector of the facet it belongs to, and $\mathbf{e}$ be the border edge it is on. We take $\mathbf{o}$ to be

$$\mathbf{o} = \omega \frac{\mathbf{n} \times \mathbf{e}}{\|\mathbf{n} \times \mathbf{e}\|} \quad \text{with} \quad \omega = \pm 1 \,, \tag{A.22}$$

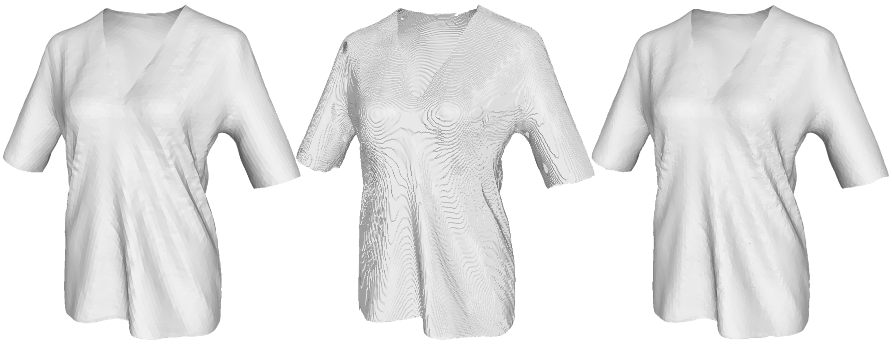
the unit vector colinear to the cross product of $\mathbf{n}$ and $\mathbf{e}$. This way, $\mathbf{o}$ is both in the tangent plane of the surface and perpendicular to the border. We choose the sign ω to orient $\mathbf{o}$ outwards. We write

$$\omega = \underset{\{-1,1\}}{\operatorname{argmax}} \; u\Big(\mathbf{v} + \omega \frac{\mathbf{n} \times \mathbf{e}}{\|\mathbf{n} \times \mathbf{e}\|}\Big) \,, \tag{A.23}$$

that is, we evaluate the UDF in both directions and pick the one that yields the highest value.

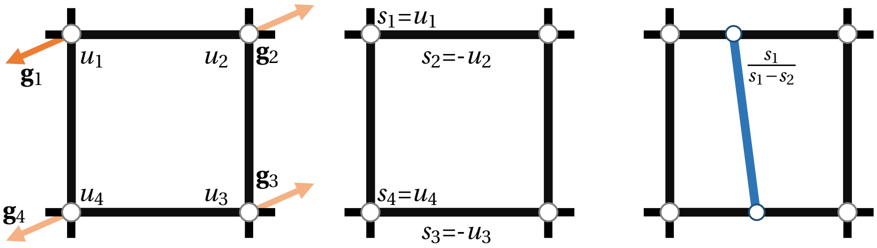
A.4.4 Meshing approximate or real UDFs

In Sec. 6.4.6 and Fig. 6.7 of the main chapter, we mention artifacts of our meshing procedure when applied to approximate UDFs and at a high resolution. This is depicted in Fig. A.20(**b**), where meshing a UDF represented by a shallow network (4 layers) with a grid resolution of



(a) (b) (c)

Figure A.20 – **Meshing UDFs: (a)** Ground truth mesh; **(b)** Our meshing procedure applied to a shallow UDF neural network yields staircase artifacts at a very high resolution (512); **(c)** Our method applied to the exact UDF at the same resolution reconstructs a smooth surface.



(a) (b) (c)

Figure A.21 – **Detecting surface crossings**: **(a)** all corners of the grid's cell are annotated with unsigned distance values u_i and gradients $\mathbf{g}_i$; **(b)** we locally approximate signed distances with $s_i = \mathrm{sgn}(\mathbf{g}_1 \cdot \mathbf{g}_i) u_i$; **(c)** marching cubes processes these pseudo-signed distances and produces a surface element accordingly.

512 yields a mesh that is not smooth.

We hypothesized that this is due to the 0-level set of the field being slightly inflated into a volume, with many grid locations evaluating to a 0 distance near the surface. This impedes Marching Cube's interpolation step and produces this staircase artifact. To validate this hypothesis, in Fig. A.20**(c)** we apply our meshing procedure to the exact UDF grid, numerically computed from the ground truth mesh of Fig. A.20**(a)**. This results in a smooth surface, thus indicating that the staircase artifact is indeed a consequence of meshing approximate UDFs.

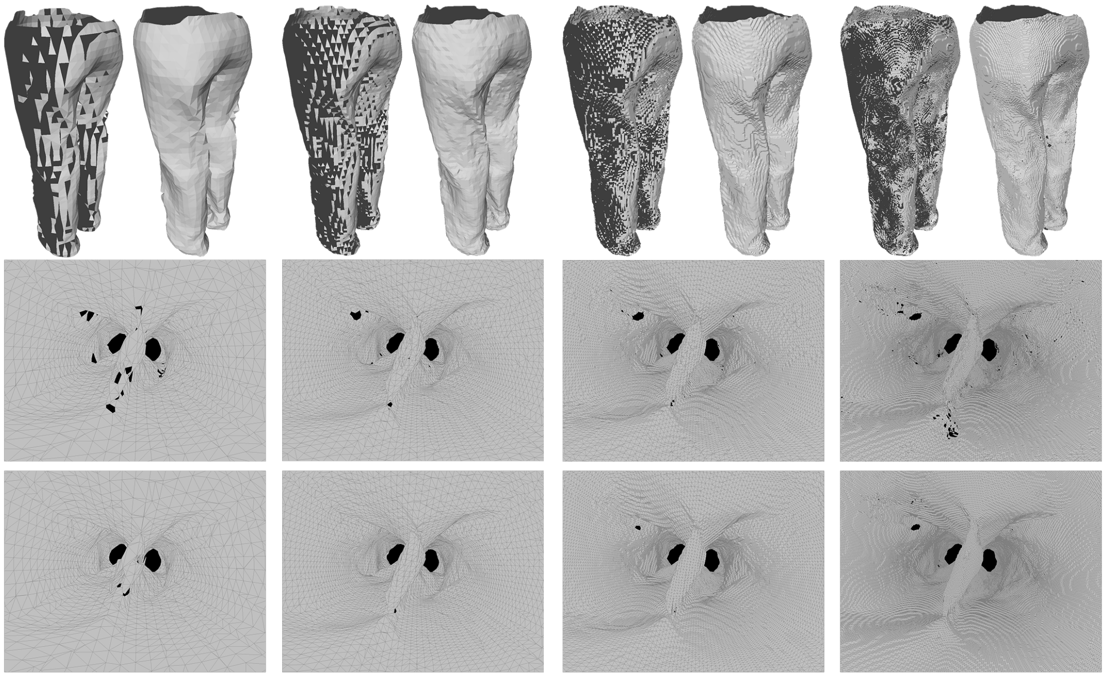
A.4.5 Ablation study: pseudo-sign and breadth-first exploration

In Sec. 6.3 we described a way to locally compute the pseudo-signed distance using gradient orientations (*PSD*), that is described in more details in Fig. A.21. The *PSD* method has two shortcomings. First, the choice of the anchor corner implies that the anchor will have a positive pseudo-sign, and thus choosing a different anchor might invert all the signs of the cell. Since the choice is arbitrary, adjacent cells might have opposing sign choices: they will produce meaningful facets, but with opposing orientations. This can be partially fixed in a post-processing step that scans the mesh trying to consistently reorient the facets, but this proved to be a time-consuming operation, and it does not always find a consistent orientation. Second, if the surface in the cell or in the immediate proximity is not smooth enough, the gradients of the field can have ambiguous orientations (i.e. they do not clearly oppose each other, for example at a 45° angle). In this setting, two different anchors can produce different pseudo-signs for the corners of the cell, and thus nearby cells that use a different anchor can assign different pseudo-signs to the same corner. This inconsistency creates an unwanted hole in the mesh and happens especially with learned UDF fields, which have noisy gradients.

The breadth-first exploration (*BFE*) method with a voting scheme that we propose has the purpose of improving these shortcomings: produce consistent normal orientations in adjacent facets and increase the robustness of the method on learned UDF fields with noisy gradients. The first objective is reached thanks to the breadth-first exploration itself, which is implemented using queues: following the surface makes it possible to store values of previously computed pseudo-signs, ensuring that corners have the same pseudo-sign in adjacent cells. This also reduces the number of dot products required to complete the meshing procedure, since corners are only computed once instead of being recomputed in every cell. However, simply plugging the pseudo-signed distance computation in this breadth-first exploration can cause even more artifacts due to anchor choice, as they can propagate in nearby cells since the cells are not treated independently anymore.

To solve this problem and at the same time address the second objective, we use the voting scheme described in Sec. 6.3. This voting scheme has been experimentally inferred by looking at artifacts of the previous procedure, and has three motivations. First, it avoids an explicit and arbitrary anchor choice, which is the main cause of inconsistencies, and it increases the robustness by making multiple neighbors vote for a single corner. Second, it prevents votes to be computed along diagonals in a cell, because the underlying interpolation algorithm of marching cubes does not create vertices along cell diagonals. Third, it prevents gradients facing each other along an edge to vote for having an opposing sign –when they indicate a local maximum of the field instead.

Moreover, we notice that in corners with possible ambiguities the absolute value of the sum of received votes will be low. Some neighbors will vote positively and some others negatively, and the weight itself of the votes can be low when gradients are not clearly facing or opposing each other. We detect these cases that get a sum of votes below a threshold, fixed to $\cos(\pi/4)$, and we put them into a separate queue with a lower priority, to be re-evaluated later. The threshold has been set by noticing that, in a single-vote scenario, gradients at a $[45°, 135°]$ angle have a high ambiguity, since a 45° variation in the angle would flip the sign of their dot product. This queue is explored when the main exploration is over, thus increasing the number of neighbors that can vote and making the sign decision more robust. We also employ a third queue, which is explored with the lowest priority, that contains cells with multiple non-adjacent facets. These cells can potentially start the exploration of a non-contiguous



(a) 64 (b) 128 (c) 256 (d) 512

Figure A.22 – **Comparing qualitative results of PSD and BFE.** Each of the 4 columns corresponds to a meshing resolution, as indicated in the labels. In each column, top row left is the result of PSD, top row right is the result of BFE. Center and bottom rows show an above view of the same mesh, with holes colored in black. The two bigger holes correspond to the legs. Center row is PSD, bottom row is BFE.

surface, and are thus explored at the very end.

To validate this algorithm, we compare *BFE* with the simple application of *PSD* using the same garment network and dataset described in Sec. 6.4.2 (and Tab. 6.1 left), at different resolutions. The post-processing steps (Fig. 6.3) applied to the two methods are the same except for the parameters used. Since *PSD* produces slightly less precise borders, we apply a coarser filtering of spurious facets and remove those whose UDF value is larger than 1/6 of the side-length of a cubic cell instead of half. In *PSD* we also apply 5 steps of laplacian smoothing on the borders instead of 1 for *BFE*.

In Fig A.22 we see that *BFE* produces consistent facets orientations, while *PSD* does not. Moreover, one can notice small holes in the garments reconstructed with *PSD* (center row), which tend to increase in number and decrease in dimension as the resolution increases, whereas *BFE* is able to close most of them (bottom row), proving to be more robust. Tab. A.7 shows that the *BFE* method produces meshes with a slightly lower Chamfer distance, except at resolution 64. Since the size of the holes produced by *PSD* is very small, they do not significatively impact the CHD of this method. They however produce artifacts that are detrimental to the quality of the reconstructed mesh. To have a quantitative measure of this, given a ground-truth mesh M and a reconstructed mesh $\widetilde{M}$, we define the *number of excess*

Resolution	**64**		**128**		**256**		**512**	
Meshing procedure	*PSD*	*BFE*	*PSD*	*BFE*	*PSD*	*BFE*	*PSD*	*BFE*
CHD (↓)	**1.63**	1.66	**1.51**	**1.51**	1.52	**1.51**	1.61	**1.53**
EH (↓)	21	**1.6**	153	**7.8**	1566	**38**	11526	**478**
Time (↓)	0.35s	**0.24s**	1.4s	**1.2s**	10.0s	**9.1s**	105s	**69s**

Table A.7 – **Comparing UDF meshing methods: pseudo sign (*PSD*) versus breadth-first exploration with voting strategy (*BFE*).** Average Chamfer distance (CHD), average number of excess holes (EH) and average processing time on 300 garments. We use a single UDF network and only change the meshing procedure.

holes as:

$$EH(\widetilde{M}, M) = ||\widetilde{H}| - |H|| \,, \tag{A.24}$$

where H and $\widetilde{H}$ are the sets of holes of M and $\widetilde{M}$, computed as closed loops of edges that belong to a single triangle. This amounts to computing the number of holes in excess that are in $\widetilde{M}$ compared to M, or vice versa.

Tab. A.7 shows that *BFE* has a consistent advantage over *PSD* in this metric across all tested resolutions. In both methods, the EH tends to increase with resolution, as the limits of the learned field are approached and the gradients become noisier. The same experiment with a network trained on only 4 garments yields better results on such garments, with the *BFE* producing no excess holes at all 64-512 resolutions, and *PSD* producing a similar amount to that shown in the table.

Finally, the *BFE* method is also slightly faster than *PSD*. This is mainly due to the reduced number of dot products computed. In *PSD* we compute 8 dot products per cell –which amounts to an average of 4 dot products per corner, since every corner belongs to 4 different cells. In *BFE* each corner receives votes from a maximum of 6 neighbours with existing pseudo-signs. Since the exploration starts from one cell and proceeds breadth-first, for the vast majority of corners only a smaller number of neighbours will actually vote, decreasing the total number of dot products.

A.4.6 Optimization from random initial latent codes

In Tab. A.8 we reproduce the experiment from Sec. 6.4.3 and fit latent codes using sparse point clouds, but start from random latent codes instead of codes from a similar semantic class. This shows that the latter is not even a requirement because, despite starting from much worse initializations, our approach still succeeds better than direct supervision on the UDF values. Starting with latent codes of the same object category remains a plausible scenario because such codes could be provided by a regressor.

	Initialization: same class				**Initialization: random**			
	Init.	$\mathscr{L}_{PC,mesh}$	$\mathscr{L}_{PC,UDF}$	$\widetilde{\mathscr{L}}_{PC,UDF}$	*Init.*	$\mathscr{L}_{PC,mesh}$	$\mathscr{L}_{PC,UDF}$	$\widetilde{\mathscr{L}}_{PC,UDF}$
CHD (↓)	20.45	**3.54**	4.54	4.69	129.51	**3.64**	4.59	4.60
IC (%,↑)	69.54	**84.84**	82.80	82.31	49.08	**84.70**	83.22	82.94
NC (%,↑)	74.54	**86.85**	80.68	86.35	56.74	**86.96**	84.20	86.62

Table A.8 – **Fitting to sparse point clouds, with different latent code initializations:** either from a code of the same garment type (left), or from a random code (right). The table shows average Chamfer (CHD), image consistency (IC), and normal consistency (NC) wrt. ground truth test garments. We report metrics for un-optimized latent codes (*Init.*), after optimizing ($\mathscr{L}_{PC,mesh}$) using our method, and optimizing either $\mathscr{L}_{PC,UDF}$ or $\widetilde{\mathscr{L}}_{PC,UDF}$ in the implicit domain.

A.4.7 Additional results

In Fig. A.23 we show additional results of our method applied to mesh the UDF regressed from NDF [CMPM20] from sparse input point clouds. In Fig. A.24, we mesh the UDF predicted by AnchorUDF [ZWLS21] from input images.

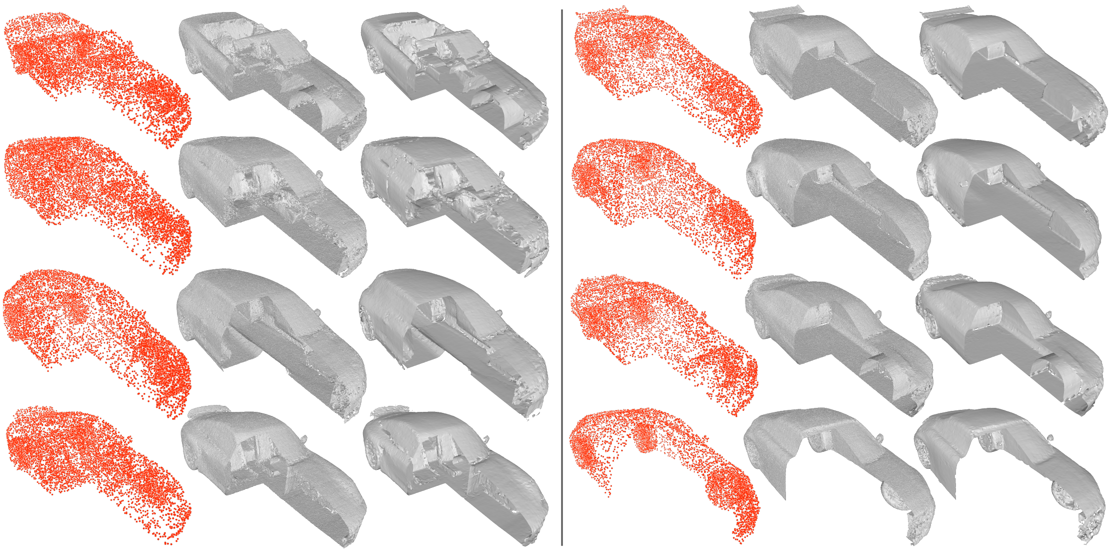


Figure A.23 – **Using our approach to triangulate the outputs of NDF [CMPM20].** For 8 examples we display the input to the network (a sparse point cloud), a mesh of the predicted UDF mesh reconstructed by the ball pivoting method in more than 2 hours, and a triangulation of the UDF generated using our method in less than 10 seconds.

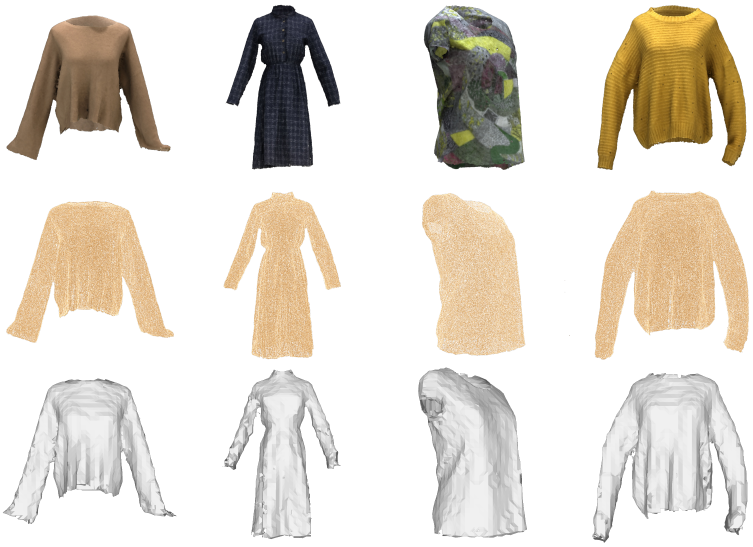


Figure A.24 – **Using our approach to triangulate the outputs of AnchorUDF** [**ZWLS21**], For 4 examples we display the input to the network (a color image), a point cloud of the predicted UDF as originally provided by this network, and a triangulation of the UDF generated using our method.

A.5 Supplementary material for DrapeNet

In this appendix we first provide more details about our networks and their architectures in Appendix A.5.1. In Appendix A.5.2 we expand on the choice and formulations of some loss terms we use. Importantly, in Appendix A.5.2 we explain the physics-based refinement procedure used in the main chapter, and show that *modelling garments as open surfaces is necessary for it.* Then in Appendix A.5.3 we report additional quantitative and qualitative results of our pipeline and the runtime of its components. Finally, in Appendix A.5.4 we describe how human ratings were collected.

A.5.1 Network Architectures and Training

Garment Generative Network

Garment Encoder To encode a given garment into a compact latent code, we first sample P points from its surface and then we feed them to a DGCNN [WSL⁺19] encoder.The input point cloud is processed by four *edge convolution* layers, which project the input 3D points into features with increasing dimensionality – *i.e.*,, 64, 64, 128 and finally 256.

Each edge convolution layer works as follows. For each input point, the features from its K neighbours are collected and used to prepare a matrix with K rows. Each row is the concatenation of two vectors: $\mathbf{f}_i - \mathbf{f}_0$ and $\mathbf{f}_0$, $\mathbf{f}_i$ and $\mathbf{f}_0$ being respectively the feature vector of the i-th neighbour and the feature vector of the considered point. Each row of the resulting matrix is then transformed independently to the desired output dimension. The output feature vector for the considered point is finally obtained by applying max pooling along the rows of the produced matrix.

The original DGCNN implementation recomputes the neighborhoods in each edge convolution layer, using the distance between the feature vectors as metric. This can be explained by the original purposes of DGCNN, *i.e.*,, point cloud classification and part segmentation. Since we are interested in encoding the geometric details of the input point cloud, we compute neighborhoods only once based on the euclidean distance of the points in the 3D space and reuse this information in every edge convolution layer. We set $K = 16$ in our experiments.

The feature vectors from the four edge convolutions are then concatenated to form a single vector with 512 elements, that is fed to a final linear layer paired with batch normalization and leaky ReLU. Such layer projects the 512 sized vectors into the final desired dimension, which is 32 in our case. The final latent code is obtained by compressing the feature matrix with shape $P \times 32$ along the first dimension with max pooling.

Garment Decoder The garment generative network features an implicit decoder that can predict the unsigned distance field of a garment starting from its latent code. More specifically, the decoder is a coordinate-based MLP that takes as inputs the garment latent code and a 3D query. Using the latent code as condition, the decoder predicts the unsigned distance from the query to the garment surface.

Our UDF decoder is inspired by [MON+19]. The input 3D query is first mapped to a higher dimensional space ($\mathbb{R}^{63}$) with the positional encoding proposed in [MPT+20], which is known to improve the capability of the network to approximate high frequency functions. The encoded query is then mapped with a linear layer to $\mathbb{R}^{512}$ and then goes through 5 residual blocks. The output of each block is computed as $\mathbf{f}_{out} = \mathbf{f}_{in} + \Delta\mathbf{f}$, where $\mathbf{f}_{in}$ is the input vector and $\Delta\mathbf{f}$ is a residual term predicted by two consecutive linear layers starting from $\mathbf{f}_{in}$. The size of the feature vector is 512 across the whole sequence of residual blocks. The output of the last block is mapped to the scalar output $out \in \mathbb{R}$ with a final linear layer.

All the linear layers but the output one are paired with Conditional Batch Normalization (CBN) [VSM+17] and ReLU activation function. CBN is used to condition the MLP with the input latent code $\mathbf{z}$. In more details, each CBN module applies standard batch normalization [IS15] to the input vectors, with the difference that the parameters of the final affine transformation are not learned during the training but are instead predicted at each inference step by dedicated linear layers starting from $\mathbf{z}$.

Finally, recall that our generative network is trained with the binary cross-entropy loss. Thus, the output of the decoder must be converted to the corresponding UDF value by first applying the sigmoid function and then scaling the result with the UDF clipping distance δ, which we set to 0.1 in our experiments. Such procedure is indeed the dual of the one applied on the UDF ground-truth labels during training to normalize them in the range $[0,1]$.

Surface Sampling We sample supervision points with a probability inversely proportional to the distance to the surface: 30% of the points are sampled directly on the input surface, 30% are sampled by adding gaussian noise with ϵ variance to surface points, 30% are obtained with gaussian noise with 3ϵ variance, and the remaining 10% are gathered by sampling uniformly the bounding box in which the garment is contained. Since in our experiments, the top and bottom garments are normalized respectively into the upper and lower halves of the $[-1,1]^3$ cube, we set $\epsilon = 0.003$.

Draping Network

The networks $\mathcal{W}(\mathbf{x}) \in \mathbb{R}^{24}$ and $\Delta x(\mathbf{x}, \beta) \in \mathbb{R}^3$ that predict blending weights and coarse displacements are implemented by a 9-layer multilayer perceptron (MLP) with a skip connection from the input layer to the middle. Each layer has 256 nodes except the middle and the last ones. ReLU is used as the activation function. The body-parameter-embedding module $\mathcal{B}(\beta, \theta) \in \mathbb{R}^{128}$ and the displacement-matrix module $\mathcal{M}(x, \mathbf{z}) \in \mathbb{R}^{128\times 3}$ for Δx_{ref} are implemented by a 5-layer MLP with LeakyReLU activation in-between. Each layer has 512 nodes except the last one. Δx_{IS} uses the same architecture as Δx_{ref}.

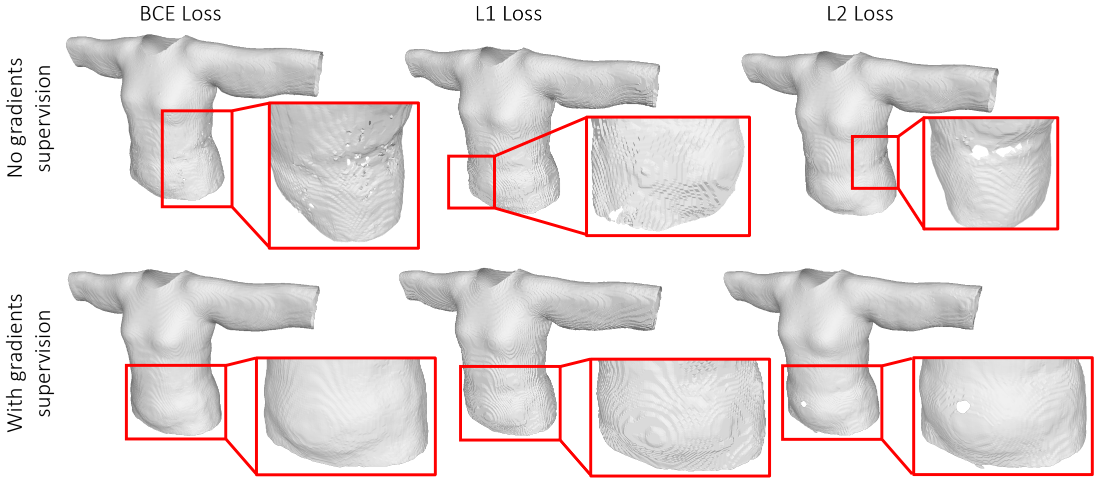


Figure A.25 – **Comparison between different loss functions for the garment generative network.** We present the same garment reconstructed by our generative network after being trained for 48 hours with six different alternatives of loss functions.

Training Hyperparameters

The generative models (top/bottom ones) are trained on the 600/300 neutral garments for 4000 epochs, using mini-batches of size $B = 4$. Each item of the mini-batch contains an input point cloud with $P = 10,000$ points and $N = 20,000$ random UDF 3D queries. The dimension of the latent codes is set to 32 for both top and bottom garments, and we set $\lambda_g = 0.1$ in

$$\mathcal{L}_{garm} = \mathcal{L}_{dist} + \lambda_g \mathcal{L}_{grad} \, . \tag{A.25}$$

The draping networks are trained for 250K iterations with mini-batches of size 18, where each item is composed of the vertices of one garment paired with one body shape and pose. We set $\lambda = 0.1$ for $\mathcal{L}_{pin}$ and $\gamma = 0.5$ for $\mathcal{L}_{layer}$.

Both the generative and the draping networks are trained with Adam optimizer [KB15b] and learning rates set to 0.0001 and 0.001 respectively.

A.5.2 Loss Terms and Ablation Studies

$\mathcal{L}_{garm}$ for Garment Reconstruction

We report here an ablation study that we conducted to determine the best formulation for $\mathcal{L}_{garm}$, the loss function presented in Eq. (7.1) of the main chapter, that we use to train our garment generative network.

In particular, we consider three variants for $\mathcal{L}_{dist}$, the term of the supervision signal that guides the network to predict accurate values for the garments UDF. In addition to the binary cross-entropy loss (BCE) presented in Eq. (7.4) of the main chapter, we study the possibility of using more traditional regression losses, such as L1 and L2 losses. Adopting the notation introduced in Section 7.3.1 of the main chapter, the L1 loss is defined as $\frac{1}{BN}\sum_{i,j}|min(y_{ij},\delta) - \tilde{y}_{ij}|$, while the L2 loss is computed as $\frac{1}{BN}\sum_{i,j}(min(y_{ij},\delta) - \tilde{y}_{ij})^2$.

On top of the three variants for $\mathscr{L}_{dist}$, we also consider for each one the possibility of removing the gradients supervision from $\mathscr{L}_{garm}$, *i.e.*,, setting $\lambda_g = 0$.

We trained our generative network for 48 hours with the resulting six loss function variants and then compared the quality of the garments reconstructed with the garment decoder. Fig. A.25 presents a significant example of what we observed on the test set. Without gradients supervision (top row of the figure), none of the considered loss functions (BCE, L1 or L2) can guide the network to predict smooth surfaces without artifacts or holes. Adding the gradients supervision (bottom row) induces a strong regularization on the predicted distance fields, helping the network to predict surfaces without holes in most of the cases. However, using the L1 loss leads to rough surfaces, as one can observe in the center column of the bottom row of the figure. The BCE and the L2 losses (first and third columns of the bottom row), instead, produce smooth surfaces that are pleasant to see. We finally opted for the BCE loss over the L2 loss, since the network trained with the latter occasionally predicts surfaces with small holes, as in the example shown in the figure.

$\mathscr{L}_{pin}$ for Bottom Garments

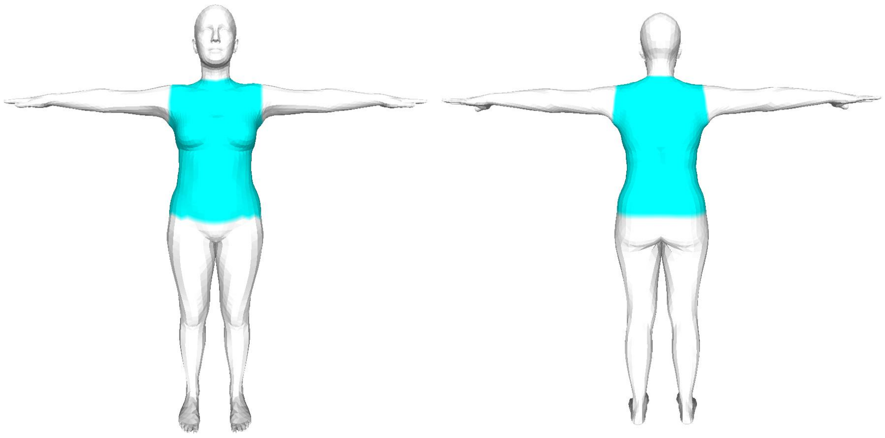


Figure A.26 – Body region (marked in cyan) used to compute $\mathscr{L}_{pin}$.

To determine V, the set of bottom garment vertices that need to be constrained by $\mathscr{L}_{pin}$, we first find the closest body vertex v_B for each bottom garment vertex v. If v_B locates in the body trunk (cyan region as shown in Fig. A.26), v is added to V.

In Fig. A.27, we show the draping results of bottom garments by using different values for λ in $\mathscr{L}_{pin}$. When λ equals 0 or 1, the deformations along the X and Z axes are not natural because no constraints or too strong constraints are applied, while it is not the case when

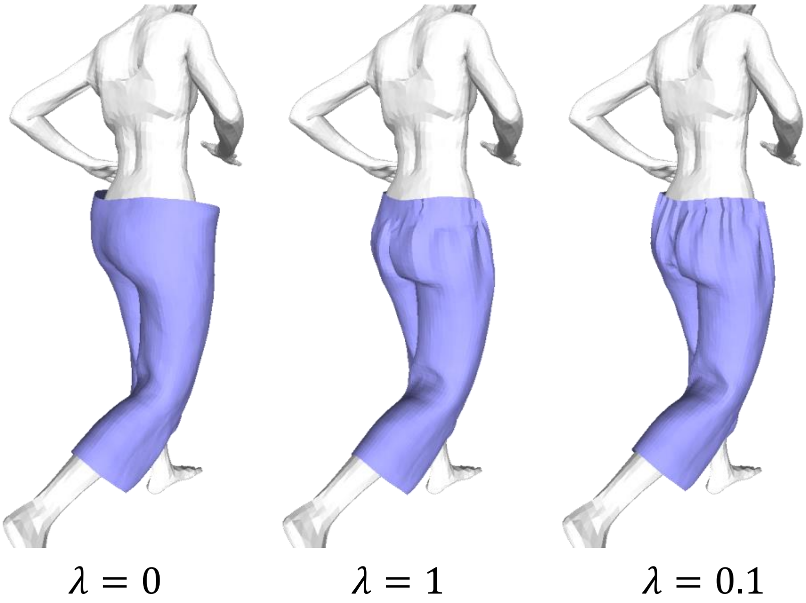


Figure A.27 – **Comparison between different values for λ of $\mathscr{L}_{pin}$.** To restrict the deformation mainly along the vertical direction (Y axis) and produce natural deformations along other directions, λ has to be a positive value smaller than 1. We use $\lambda = 0.1$ for our training.

$\lambda = 0.1$, which is our setting.

$\mathscr{L}_{layer}$ for Top-bottom Intersection

To determine C, the set of body vertices covered by both the top and bottom garments, we first subdivide the SMPL body mesh for a higher resolution, and then we compute C_{top} the set of closest body vertices for the given top garment, and C_{bottom} the set of closest body vertices for the bottom. C is derived as the intersection of C_{top} and C_{bottom}.

In Fig. A.28 we compare the results of models trained without and with $\mathscr{L}_{layer}$. We can observe that without $\mathscr{L}_{layer}$, the top tank can intersect with the bottom trousers, while it is not the case when using $\mathscr{L}_{layer}$. This indicates the efficacy of $\mathscr{L}_{layer}$ to avoid intersections between garments.

Physics-based Refinement

After recovering the draped garment $\mathbf{G}_D$ from images by the optimization of Eq. (7.12) of the main chapter, we can apply the physics-based objectives of Eq. (7.7) (main chapter) to increase its level of realism

$$\begin{aligned} L(\Delta_\mathbf{G}) =& \mathscr{L}_{strain}(\mathbf{G}_D + \Delta_\mathbf{G}) + \mathscr{L}_{bend}(\mathbf{G}_D + \Delta_\mathbf{G}) \\ &+ \mathscr{L}_{gravity}(\mathbf{G}_D + \Delta_\mathbf{G}) + \mathscr{L}_{col}(\mathbf{G}_D + \Delta_\mathbf{G}) , \end{aligned} \tag{A.26}$$

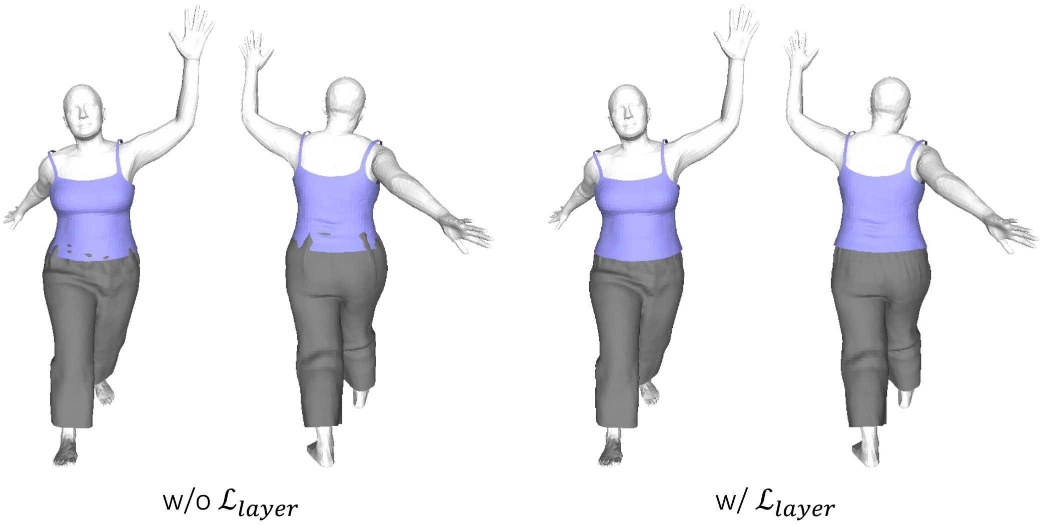


Figure A.28 – **Comparison: draping without and with $\mathscr{L}_{layer}$.** Without it, the top and bottom garments intersect with each other.

where $\Delta_{\mathbf{G}}$ is the per-vertex-displacement initialized to zero. For the recovery from 3D scans, we apply the following optimization which minimizes both the above physics-based objectives and the Chamfer Distance $d(\cdot)$ to the input scan $\mathbf{S_G}$

$$\begin{aligned} L(\Delta_{\mathbf{G}}) =& \mathscr{L}_{strain}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathscr{L}_{bend}(\mathbf{G}_D + \Delta_{\mathbf{G}}) \\ &+ \mathscr{L}_{gravity}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathscr{L}_{col}(\mathbf{G}_D + \Delta_{\mathbf{G}}) \\ &+ d(\mathbf{G}_D + \Delta_{\mathbf{G}}, \mathbf{S_G}) \,. \end{aligned} \tag{A.27}$$

This refinement procedure is only applicable to open surface meshes, and our UDF model is thus key to enabling it. Applying Eq. (A.26) or Eq. (A.27) to an inflated garment (as recovered by SMPLicit [CPA$^+$21], ClothWild [MNSL22] and DIG [LGRF22]) indeed yields poor results with many self-intersections as illustrated in Fig. A.29. This is because inflated garments modelled as SDFs have a non-zero thickness, with distinct inner and outer surfaces whose interactions are not taken into account in this fabric model. The physics model we apply on garment meshes indeed considers collisions of the garment with the body, but not with itself, which is what happens with the inner and outer surfaces in Fig. A.29. Adding a physics term to prevent self intersections would not be trivial, and is related to the complex task of untangling layered garments [SOTC22, BRB$^+$19].

Note that this is also the case for most garment draping softwares [NSO12, NPO13, PNJO14, TWL$^+$18, GCP$^+$22] to expect single layer garments. Modeling garment with UDFs is thus a key feature of our pipeline for its integration in downstream tasks.

Both the optimizations of Eqs. (7.12) and (7.13) of the main chapter and Eqs. (A.26) and (A.27) are done with Adam [FSG17] but with different learning rates set to 0.01 and 0.001 respectively.

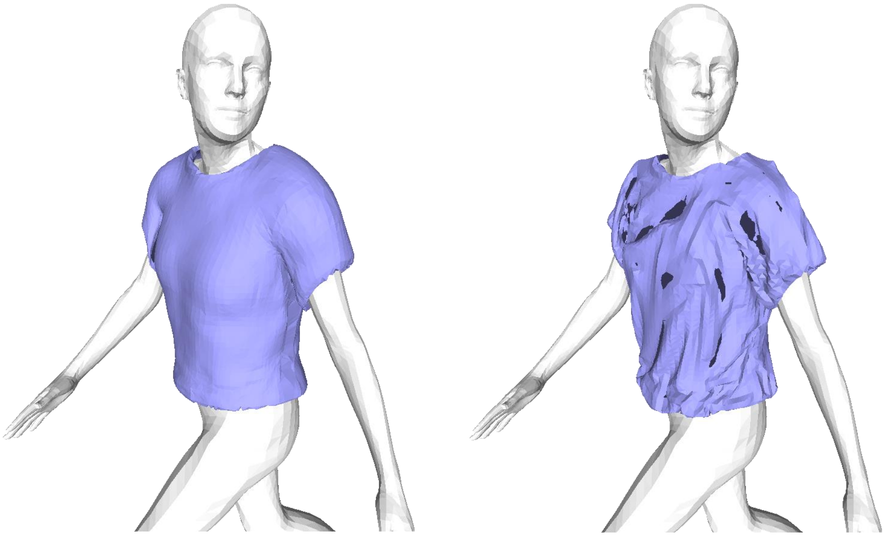

The watertight mesh reconstructed by SDF.
The mesh refined by physics-based objectives.

Figure A.29 – **Applying post-refinement procedure to watertight mesh**. Left: the watertight mesh reconstructed by DIG [LGRF22]. Right: the same mesh after being refined with physics-based objectives (Eq. (A.26)). Physics-based refinement is not compatible with inflated garment meshes, and leads to many self-intersections.

A.5.3 Additional Results

Garment Encoder/Decoder

Additional Qualitative Results Fig. A.30 and Fig. A.31 show the encoding-decoding capabilities of our garment generative network for top and bottom *test* garments, respectively. The ground-truth garments are passed through the garment encoder, which produces a compact latent code for each clothing item. Then, our garment decoder reconstructs the input garments surface from the latent codes. It is possible to notice how the output garments closely match the input ones, both in terms of geometry and topology.

Latent Space Optimization (LSO). After training the garment generative network, we obtain a latent space that allows us to sample a garment latent code and to feed it to the implicit decoder to reconstruct the explicit surface. We study here the possibility of exploring the garment latent space by the means of LSO. To do that, given a target 2D silhouette or a sparse 3D point cloud of a garment, we optimize with gradient descent a latent code – initialized to the training codes average – so that the frozen decoder conditioned on it can produce a garment which fits the target image or point cloud.

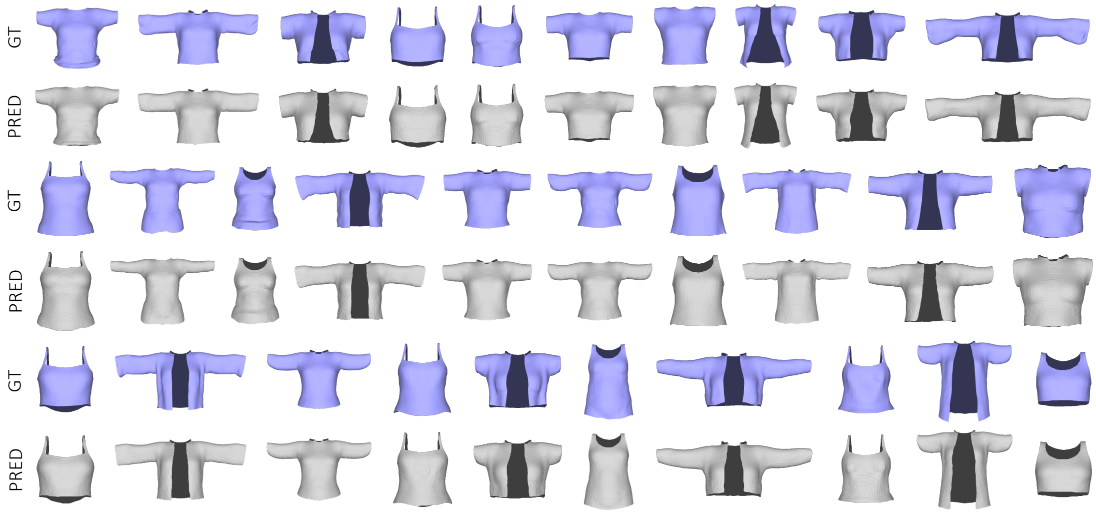


Figure A.30 – **Generative network: reconstruction of unseen garments in neutral pose/shape (top garments).** Latent codes for unseen garments can be obtained with our garment encoder. These codes are then used by the garment decoder to reconstruct open surface meshes. Input garments are colored in purple, while the reconstructed meshes are colored in gray.

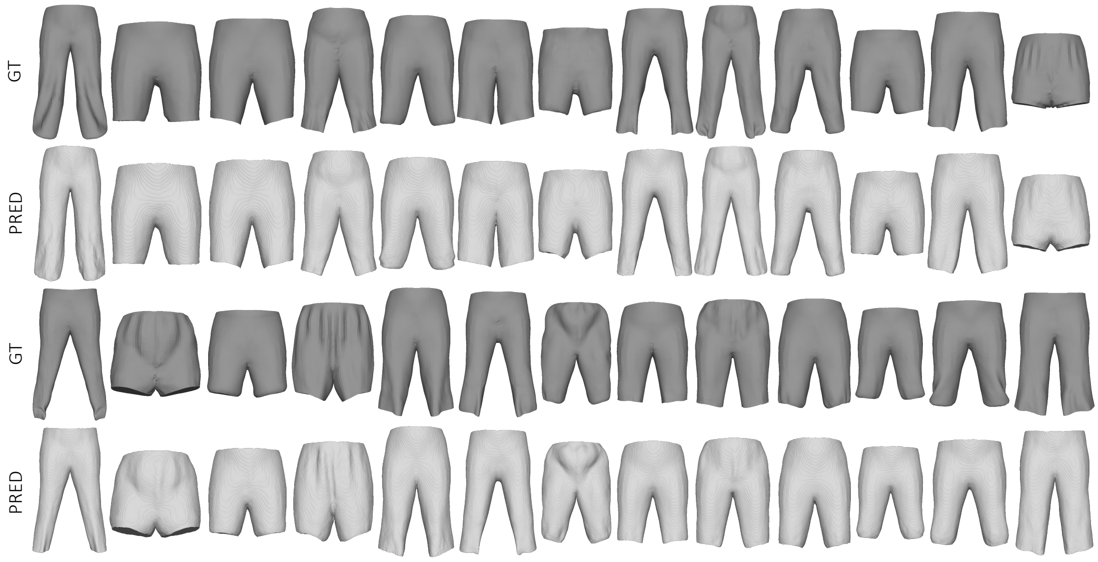


Figure A.31 – **Generative network: reconstruction of unseen garments in neutral pose/shape (bottom garments).** Latent codes for unseen garments can be obtained with our garment encoder. These codes are then used by the garment decoder to reconstruct open surface meshes. Input garments are colored in dark gray, while the reconstructed meshes are colored in light gray.

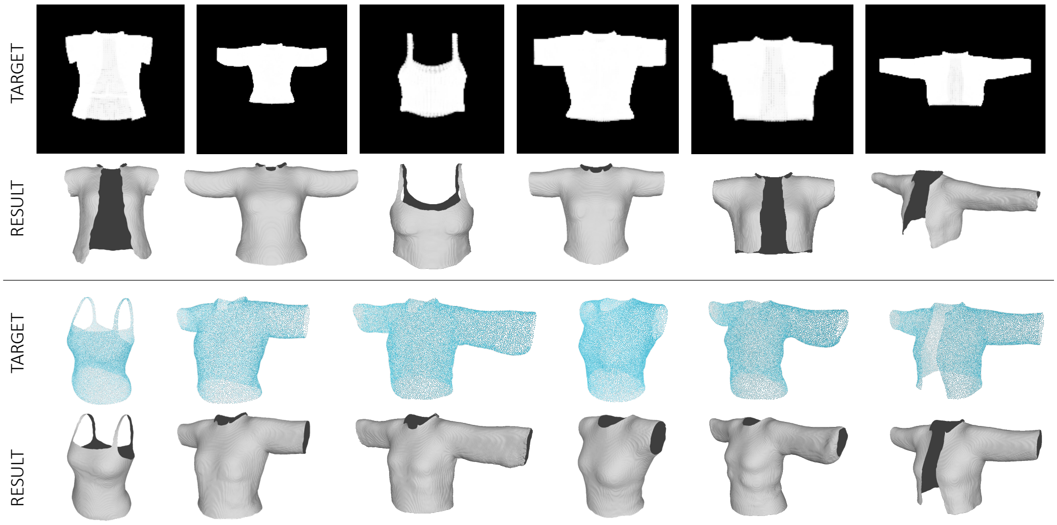


Figure A.32 – **Generative network: latent space optimization (top garments).** After training, we can explore the latent space learned by the garment generative network with gradient descent, to recover target garments from 2D silhouettes (top) or 3D point clouds (bottom).

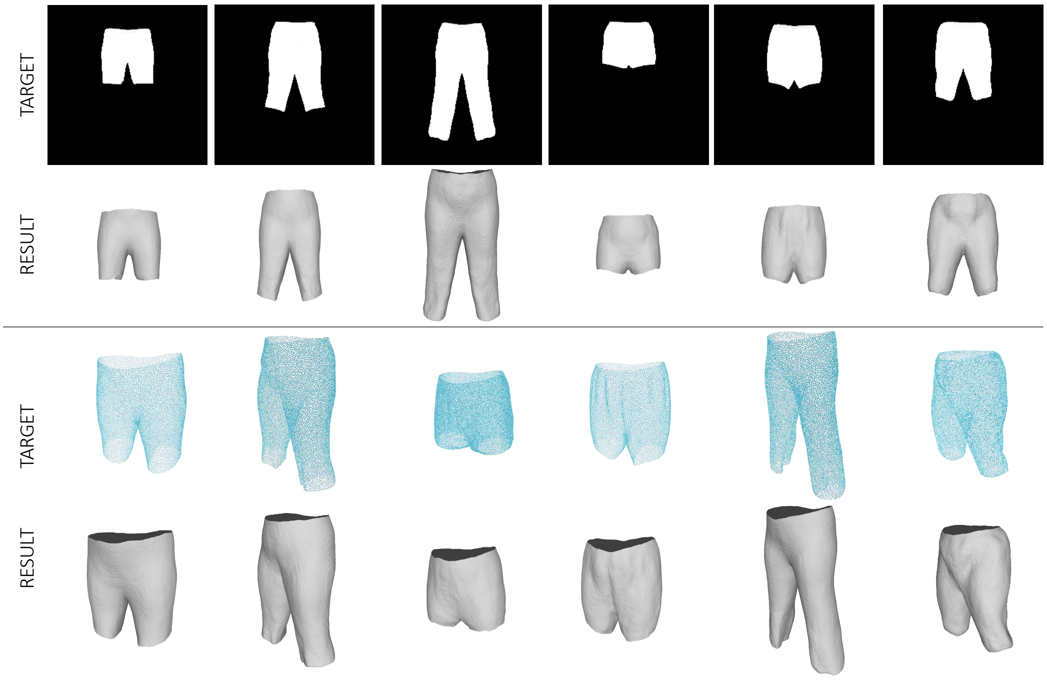


Figure A.33 – **Generative network: latent space optimization (bottom garments).** After training, we can explore the latent space learned by the garment generative network with gradient descent, to recover garments from 2D silhouettes (top) or 3D point clouds (bottom).

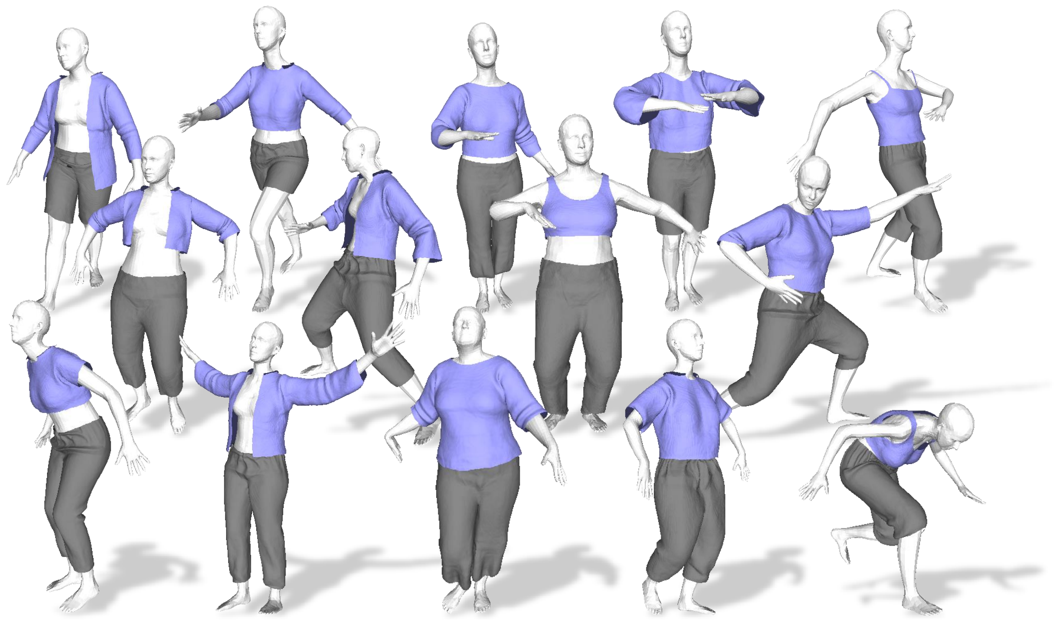


Figure A.34 – **Additional results:** draping garments of different topologies over bodies in various shapes and poses with our method.

Given the silhouette $\mathscr{S}$ of a target garment, we can retrieve its latent code $\mathbf{z}$ by minimizing

$$\begin{aligned} L(\mathbf{z}) &= L_{\mathrm{IoU}}(R(\mathbf{G}), \mathscr{S}) , \\ \mathbf{G} &= \mathrm{MeshUDF}(D_G(\cdot, \mathbf{z})) , \end{aligned} \tag{A.28}$$

where L_{IoU} is the IoU loss [LZK+21] in pixel space measuring the difference between 2D silhouettes , $R(\cdot)$ is a differentiable silhouette renderer for meshes [RRN+20], and $\mathbf{G}$ is the garment mesh reconstructed with our garment decoder using $\mathbf{z}$.

In the case of a target garment provided as a point cloud $\mathscr{P}$, the garment latent code $\mathbf{z}$ can be obtained by minimizing

$$\begin{aligned} L(\mathbf{z}) &= d(ps(\mathbf{G}), \mathscr{P}) , \\ \mathbf{G} &= \mathrm{MeshUDF}(D_G(\cdot, \mathbf{z})) , \end{aligned} \tag{A.29}$$

where $d(a, b)$ is the Chamfer distance [FSG17] between point clouds a and b, and $ps(\cdot)$ represents a differentiable procedure to sample points from a given mesh [RRN+20].

In both cases, we run the optimization for 1000 steps, with Adam optimizer [KB15b] and learning rate set to 0.01.

In Fig. A.32 and Fig. A.33 we present some results of the LSO procedures here described, showing that the latent space learned by the garment generative network can be explored effectively with gradient descent to recover the codes associated with the target garments.

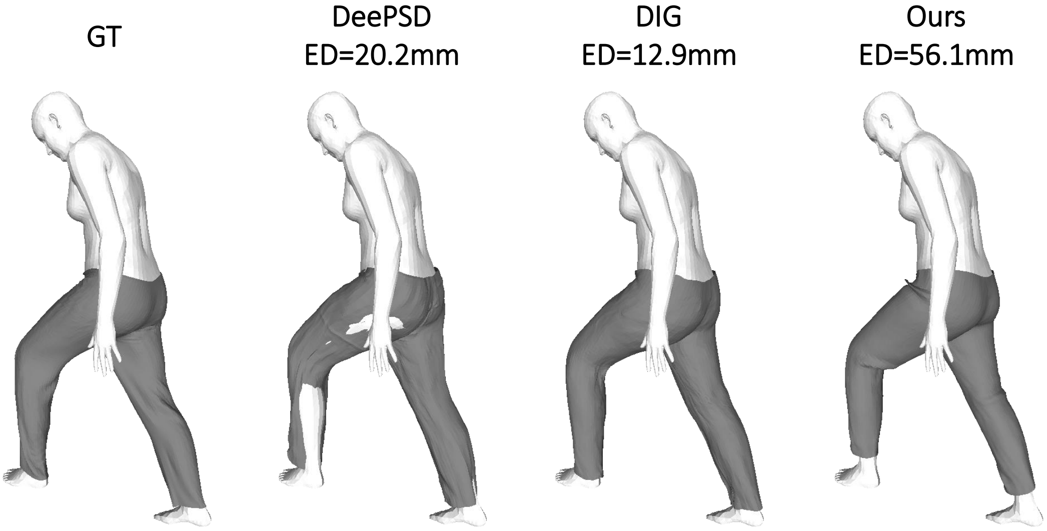


Figure A.35 – **Comparison between DeePSD, DIG and our method.** Our result is more realistic than the others despite having the highest Euclidean distance (ED) error.

Draping Network

Additional Qualitative Results In Fig. A.34 we show additional qualitative results of garment draping produced by our method, where the garment meshes are generated by our UDF model. It can be seen that our method can realistically drape garments with different topologies over bodies of various shapes and poses.

Euclidean Distance is not a Good Metric In Fig. A.35, we show an example of bottom garment where our result is more realistic than the competitors DeePSD [BMTE21] and DIG [LGRF22] despite having the highest Euclidean distance. This demonstrates again that Euclidean distance is not able to measure the draping quality, as discussed in the main chapter.

Quantitative Evaluation in Physics-based Energy In Table A.9, we report the physics-based energy of *Strain*, *Bending* and *Gravity* as proposed by [SOC22] on test garment meshes when draped by DeePSD, DIG and our method. These energy terms are used as training losses for our garment network (Eqs. (7.7) and (7.8) of the main chapter). For the gravitational potential energy, we choose the lowest body vertex as the 0 level. Generally, our results have the lowest energies, especially for the *Strain* component. Since DeePSD and DIG do not apply constraints on mesh faces, their results exhibit much higher *Strain* energy. This indicates that our method can produce results that have more realistic physical properties.

Top	Strain ↓	Bending ↓	Gravity ↓	Total ↓
DeePSD	7.22	**0.01**	**0.98**	8.21
DIG	6.32	**0.01**	1.05	7.38
Ours	**0.43**	**0.01**	1.05	**1.81**

Bottom	Strain ↓	Bending ↓	Gravity ↓	Total ↓
DeePSD	8.46	0.02	0.90	9.38
DIG	7.48	**0.01**	0.90	8.39
Ours	**0.41**	**0.01**	**0.86**	**1.28**

Table A.9 – **Draping unseen garment meshes.** Quantitative comparison in physics-based energy between DeePSD, DIG and our method. “Strain”, “Bending” and “Gravity“ denote the membrane strain energy, the bending energy and the gravitational potential energy, respectively.

Inference Times

We report inference times for the components of our framework, computed on an NVIDIA Tesla V100 GPU. The garment encoder, which needs to be run only once for each garment, takes ∼25 milliseconds. The decoder takes ∼2 seconds to reconstruct an explicit garment mesh from a given latent code, including the modified Marching Cubes from [GSF22] at resolution 256^3.

The draping network takes ∼5 ms to deform a garment mesh composed of 5K vertices. Since it is formulated in an implicit manner and is queried at each vertex, its inference time increases to ∼8 ms for a mesh with 8K vertices, or ∼53 ms with 100K vertices.

Fitting SMPLicit [CPA$^+$21] to 3D Scans

In Fig. A.36 we show results of fitting the concurrent approach SMPLicit [CPA$^+$21] to 3D scans of the SIZER dataset [TBTPM20]. We can observe that they are not as realistic as ours shown in Fig. 7.10 of the main chapter. Since we have no access to their code and not enough information for a re-implementation, we directly extracted this figure from [CPA$^+$21].

A.5.4 Human Evaluation

In Fig. A.37 we show the interface and instructions that were presented to the 187 respondents of our survey. These evaluators were volunteers with various backgrounds from the authors respective social circles, which were purposely not given any further detail or instruction. We collected collected 3738 user opinions in total, each user expressing 20 opinions on average.

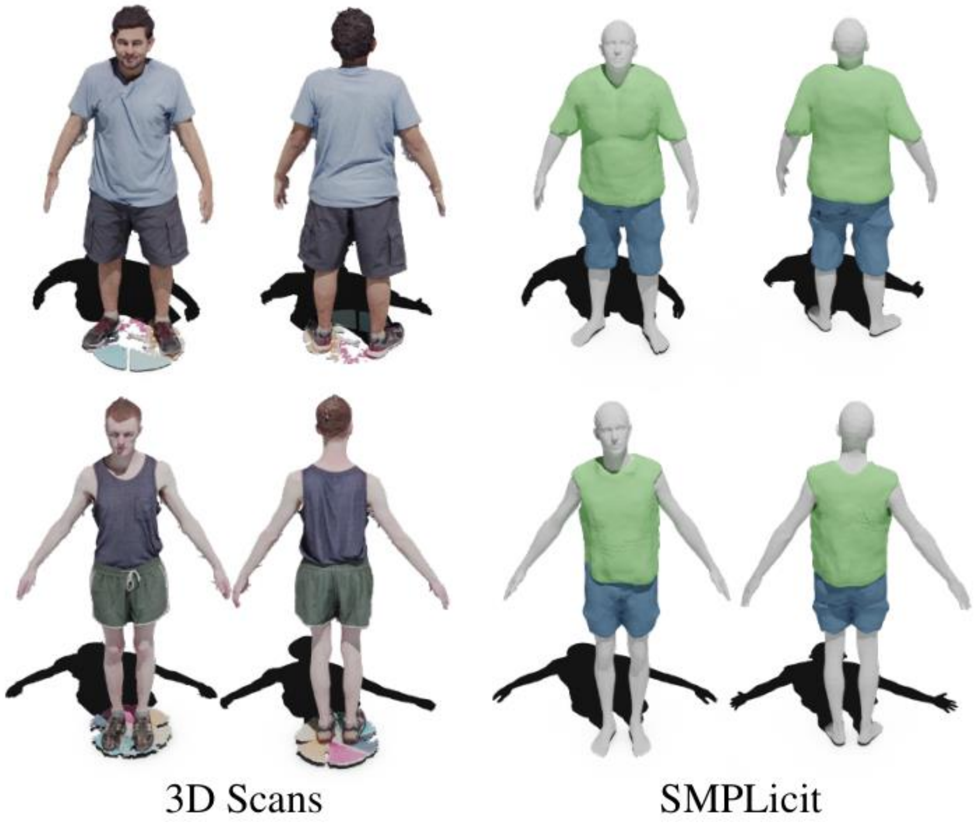
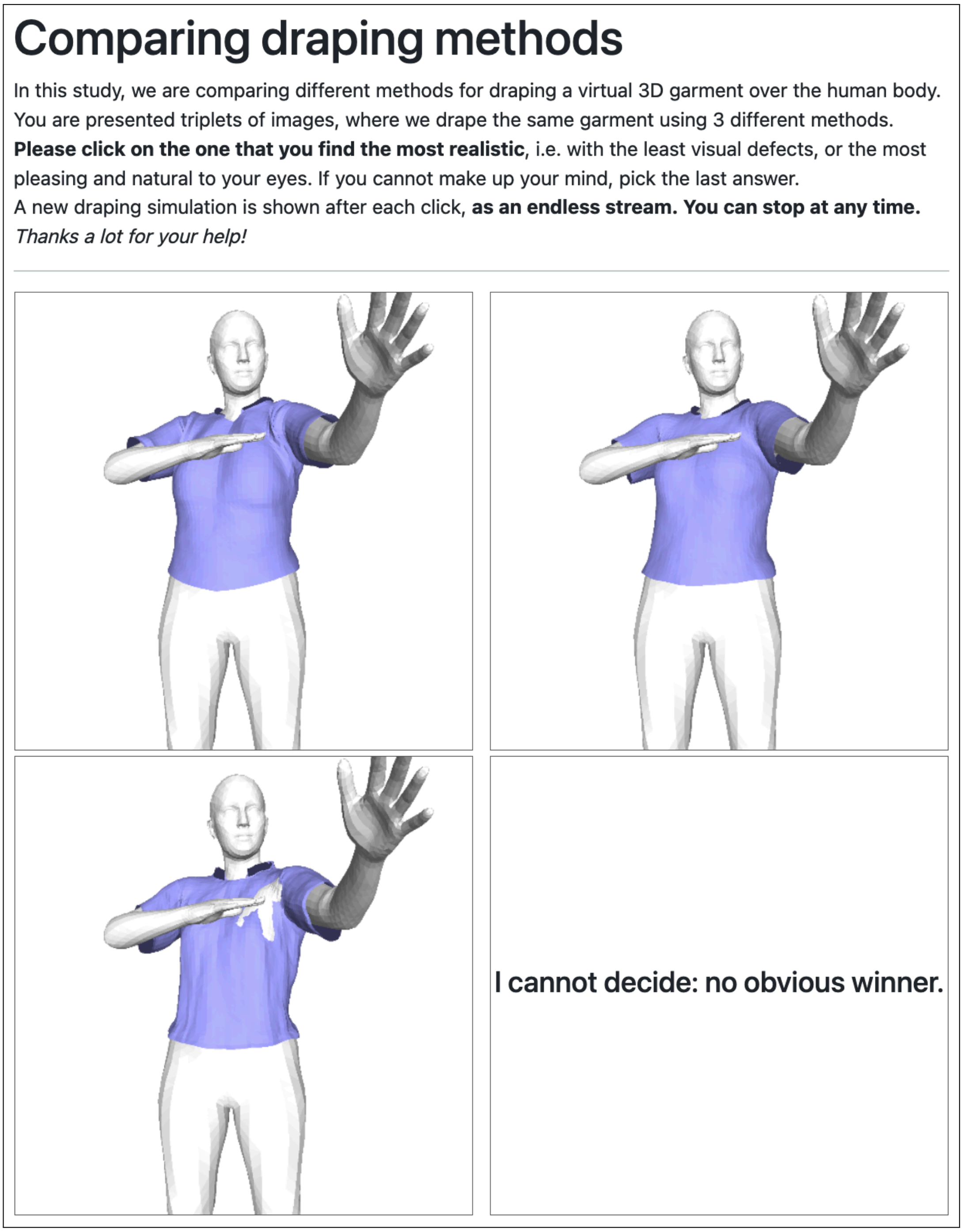


Figure A.36 – **Recovered garments of SMPLicit from 3D scans**. Figures are extracted from [CPA+21].



Comparing draping methods

In this study, we are comparing different methods for draping a virtual 3D garment over the human body. You are presented triplets of images, where we drape the same garment using 3 different methods. **Please click on the one that you find the most realistic**, i.e. with the least visual defects, or the most pleasing and natural to your eyes. If you cannot make up your mind, pick the last answer.
A new draping simulation is shown after each click, **as an endless stream. You can stop at any time.**
Thanks a lot for your help!

I cannot decide: no obvious winner.

Figure A.37 – **Interface of our qualitative survey.** The garment is draped with our method, DIG, and DeePSD, in a random order.

Bibliography

[AHY$^+$19] M. Atzmon, N. Haim, L. Yariv, O. Israelov, H. Maron, and Y. Lipman. Controlling Neural Level Sets. In *NeurIPS*, 2019.

[AJT02] Grégoire Allaire, François Jouve, and Anca-Maria Toader. A Level-Set Method for Shape Optimization. *Comptes Rendus Mathématiques*, 334(12):1125–1130, 2002.

[AL20a] M. Atzmon and Y. Lipman. SAL: Sign Agnostic Learning of Shapes from Raw Data. In *CVPR*, 2020.

[AL20b] M. Atzmon and Y. Lipman. SALD: Sign Agnostic Learning with Derivatives. In *ICLR*, 2020.

[AT04] A. Agarwal and B. Triggs. 3D Human Pose from Silhouettes by Relevance Vector Regression. In *CVPR*, 2004.

[BME20] H. Bertiche, M. Madadi, and S. Escalera. CLOTH3D: Clothed 3D Humans. In *ECCV*, pages 344–359, 2020.

[BME21] H. Bertiche, M. Madadi, and S. Escalera. PBNS: Physically Based Neural Simulation for Unsupervised Garment Pose Space Deformation. *ACM Transactions on Graphics*, 2021.

[BMR$^+$99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The Ball-Pivoting Algorithm for Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[BMTE21] H. Bertiche, M. Madadi, E. Tylson, and S. Escalera. DeePSD: Automatic Deep Skinning and Pose Space Deformation for 3D Garment Animation. In *ICCV*, 2021.

[BPG$^+$19] J. Bednarík, S. Parashar, E. Gundogdu, M. Salzmann, and P. Fua. Shape reconstruction by learning differentiable surface representations. *arXiv Preprint*, abs/1911.11227, 2019.

[BRB$^+$19] T. Buffet, D. Rohmer, L. Barthe, L. Boissieux, and M-P. Cani. Implicit untangling: A robust solution for modeling layered clothing. *ACM Transactions on Graphics*, 38(4):1–12, 2019.

[BRFF18] P. Baqué, E. Remelli, F. Fleuret, and P. Fua. Geodesic Convolutional Shape Optimization. In *International Conference on Machine Learning*, 2018.

[BTTPM19] B. L. Bhatnagar, G. Tiwari, C. Theobalt, and G. Pons-Moll. Multi-Garment Net: Learning to Dress 3D People from Images. In *ICCV*, 2019.

[BV99] V. Blanz and T. Vetter. A Morphable Model for the Synthesis of 3D Faces. In *ACM SIGGRAPH*, pages 187–194, August 1999.

[BW98] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *ACM SIGGRAPH*, pages 43–54, 1998.

[BWS$^+$18] T. Bagautdinov, C. Wu, J. Saragih, P. Fua, and Y. Sheikh. Modeling Facial Geometry Using Compositional VAEs. In *CVPR*, 2018.

[CAP20] J. Chibane, T. Alldieck, and G. Pons-Moll. Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion. In *CVPR*, 2020.

[CBZ$^+$19] 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.

[CCC$^+$08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: An Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, 2008.

[CFG$^+$15] 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.

[Che95] E. V. Chernyaev. Marching Cubes 33: Construction of Topologically Correct Isosurfaces. In *Institute for High Energy Physics, Moscow, Russia, Report CN/95-17*, 1995.

[Che18] Haofeng Chen. Single image depth estimation with feature pyramid network. https://github.com/xanderchf/MonoDepth-FPN-PyTorch, 2018.

[CMPM20] J. Chibane, A. Mir, and G. Pons-Moll. Neural Unsigned Distance Fields for Implicit Function Learning. In *NeurIPS*, 2020.

[CPA$^+$21] E. Corona, A. Pumarola, G. Alenya, G. Pons-Moll, and F. Moreno-Noguer. Smplicit: Topology-Aware Generative Model for Clothed People. In *CVPR*, 2021.

[CSMS13] Frederic Cordier, Hyewon Seo, Mahmoud Melkemi, and Nickolas S Sapidis. Inferring mirror symmetric 3d shapes from sketches. *Computer-Aided Design*, 45(2):301–311, 2013.

[CTFZ22] Z. Chen, A. Tagliasacchi, T. Funkhouser, and H. Zhang. Neural Dual Contouring. In *arXiv Preprint*, 2022.

[CXG$^+$16a] 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 *ECCV*, pages 628–644, 2016.

[CXG$^+$16b] C. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3DR2N2: A Unified Approach for Single and Multi-View 3D Object Reconstruction. In *ECCV*, 2016.

[CZ19] Z. Chen and H. Zhang. Learning Implicit Fields for Generative Shape Modeling. In *CVPR*, 2019.

[CZ21] Z. Chen and H. Zhang. Neural Marching Cubes. In *ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia)*, 2021.

[DAI$^+$18] J. Delanoy, M. Aubry, P. Isola, A. Efros, and A. Bousseau. 3D Sketching Using Multi-View Deep Volumetric Prediction. *ACM on Computer Graphics and Interactive Techniques*, 1(1):1–22, 2018.

[Daw] Dawson-Haggerty et al. trimesh.

[Des18] M. Designer, 2018. https://www.marvelousdesigner.com.

[DFRS03] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive Contours for Conveying Shape. *ACM SIGGRAPH*, 22(3):848–855, July 2003.

[DK91] Akio Doi and Akio Koide. An Efficient Method of Triangulating Equivalued Surfaces by Using Tetrahedral Cells. *Transactions on Information and Systems*, 74(1):214–224, 1991.

[DRC$^+$22] A. Davydov, A. Remizova, V. Constantin, S. Honari, M. Salzmann, and P. Fua. Adversarial Parametric Pose Prior. In *CVPR*, 2022.

[DSC$^+$20] M. Dvorožňák, D. Sỳkora, C. Curtis, B. Curless, O. Sorkine-Hornung, and D. Salesin. Monster Mash: A Single-View Approach to Casual 3D Modeling and Animation. *ACM Transactions on Graphics*, 2020.

[DZW$^+$20] Y. Duan, H. Zhu, H. Wang, L. Yi, R. Nevatia, and L. J. Guibas. Curriculum DeepSDF. In *ECCV*, 2020.

[FGF$^+$05] Yohan D Fougerolle, Andrei Gribok, Sebti Foufou, Frédéric Truchetet, and Mongi A Abidi. Boolean Operations with Implicit and Parametric Representation of Primitives using R-Functions. *TVCG*, 2005.

[FLWM18] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. Splinecnn: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In *CVPR*, 2018.

[FSG17] H. Fan, H. Su, and L. Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *CVPR*, 2017.

[Fua96] P. Fua. Model-Based Optimization: Accurate and Consistent Site Modeling. In *ISPRS*, July 1996.

[GCP+22] E. Gundogdu, V. Constantin, S. Parashar, A. Seifoddini, M. Dang, M. Salzmann, and P. Fua. Garnet++: Improving Fast and Accurate Static 3D Cloth Draping by Curvature Loss. *PAMI*, 22(1):181–195, 2022.

[GCS+19] E. Gundogdu, V. Constantin, A. Seifoddini, M. Dang, M. Salzmann, and P. Fua. Garnet: A Two-Stream Network for Fast and Accurate 3D Cloth Draping. In *ICCV*, 2019.

[GD95] D.M. Gavrila and L.S. Davis. 3D Model-Based Tracking of Human Upper Body Movement: A Multi-View Approach. In *IEEE International Symposium on Computer Vision*, pages 253–258, November 1995.

[GFK+18] T. Groueix, M. Fisher, V. Kim, B. Russell, and M. Aubry. Atlasnet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *CVPR*, 2018.

[GMJ19] G. Gkioxari, J. Malik, and J. Johnson. Mesh R-CNN. In *ICCV*, 2019.

[GRF20] B. Guillard, E. Remelli, and P. Fua. UCLID-Net: Single View Reconstruction in Object Space. In *NeurIPS*, 2020.

[GRL+22] B. Guillard, E. Remelli, A. Lukoianov, S. Richter, T. Bagautdinov, P. Baque, and P. Fua. Deepmesh: Differentiable Iso-Surface Extraction. In *arXiv Preprint*, 2022.

[GRYF21] B. Guillard, E. Remelli, P. Yvernay, and P. Fua. Sketch2mesh: Reconstructing and Editing 3D Shapes from Sketches. In *ICCV*, 2021.

[GSF22] B. Guillard, F. Stella, and P. Fua. MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks. In *ECCV*, 2022.

[GSH+19] Y. Gryaditskaya, M. Sypesteyn, J.W. Hoftijzer, S.C. Pont, F. Durand, and A. Bousseau. Opensketch: A Richly-Annotated Dataset of Product Design Sketches. In *ACM Transactions on Graphics*, 2019.

[GYH+20] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit Geometric Regularization for Learning Shapes. In *ICML*, 2020.

[HAESB20] Z. Hao, H. Averbuch-Elor, N. Snavely, and S. Belongie. DualSDF: Semantic Shape Manipulation Using a Two-Level Representation. In *CVPR*, pages 7631–7641, 2020.

[HCJS20] Tong He, John Collomosse, Hailin Jin, and Stefano Soatto. Geo-pifu: Geometry and pixel aligned implicit functions for single-view human reconstruction. *NeurIPS*, 2020.

[HF20] P. Henderson and V. Ferrari. Learning Single-Image 3D Reconstruction by Generative Modelling of Shape, Pose and Shading. *IJCV*, 128(4):835–854, 2020.

[HTM17] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *International Conference on 3D Vision*, pages 412–420. IEEE, 2017.

[HZRS16] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

[IBLM19] Karim Iskakov, Egor Burkov, Victor Lempitsky, and Yury Malkov. Learnable triangulation of human pose. In *ICCV*, pages 7718–7727, 2019.

[IMT06] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *ACM SIGGRAPH 2006 Courses*, 2006.

[IS15] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015.

[ISF07] S. Ilić, M. Salzmann, and P. Fua. Implicit Meshes for Effective Silhouette Handling. *IJCV*, 72(7), 2007.

[IZZE17] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-To-Image Translation with Conditional Adversarial Networks. In *CVPR*, pages 1125–1134, 2017.

[JFD20] A. Jin, Q. Fu, and Z. Deng. Contour-Based 3D Modeling through Joint Embedding of Shapes and Contours. In *Symposium on Interactive 3D Graphics and Games*, 2020.

[JGZ$^+$17] Mengqi Ji, Juergen Gall, Haitian Zheng, Yebin Liu, and Lu Fang. Surfacenet: An end-to-end 3d neural network for multiview stereopsis. In *ICCV*, pages 2307–2315, 2017.

[JHR$^+$15] Amaury Jung, Stefanie Hahmann, Damien Rohmer, Antoine Begault, Laurence Boissieux, and Marie-Paule Cani. Sketching folds: Developable surfaces from non-planar silhouettes. *Acm Transactions on Graphics (TOG)*, 34(5):1–12, 2015.

[JJHZ20] 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.

[JJT$^+$07] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. OpenFOAM: A C++ Library for Complex Physics Simulations. In *International workshop on coupled methods in numerical dynamics*, 2007.

[JLSW02] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. In *SIGGRAPH*, 2002.

[JZH$^+$20] B. Jiang, J. Zhang, Y. Hong, J. Luo, L. Liu, and H. Bao. Bcnet: Learning body and cloth shape from a single image. In *ECCV*, 2020.

[KB15a] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimisation. In *International Conference on Learning Representations*, 2015.

[KB15b] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson Surface Reconstruction. In *Eurographics Symposium on Geometry processing*, pages 61–70, 2006.

[KG07] Y. Kho and M. Garland. Sketching Mesh Deformations. In *ACM SIGGRAPH courses*, 2007.

[KHR02] O. Karpenko, J. F. Hughes, and R. Raskar. Free-Form Ssketching with Variational Implicit Surfaces. *Computer Graphics Forum*, 2002.

[KKM10] Gaetan Kenway, Graeme Kennedy, and Joaquim R. R. A. Martins. A cad-free approach to high-fidelity aerostructural optimization. In *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, 2010.

[KL21] M. Korosteleva and S. Lee. Generating Datasets of 3D Garments with Sewing Patterns. In *Advances in Neural Information Processing Systems*, 2021.

[KTEM18] A. Kanazawa, S. Tulsiani, A. Efros, and J. Malik. Learning Category-Specific Mesh Reconstruction from Image Collections. In *CVPR*, 2018.

[KUH18] H. Kato, Y. Ushiku, and T. Harada. Neural 3D Mesh Renderer. In *CVPR*, 2018.

[LB03] A. Lopes and K. Brodlie. Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. In *TVCG*, 2003.

[LBK17] T. Liu, S. Bouaziz, and L. Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics*, 2017.

[LC87] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *ACM SIGGRAPH*, pages 163–169, 1987.

[LCT18] Z. Lahner, D. Cremers, and T. Tung. Deepwrinkles: Accurate and Realistic Clothing Modeling. In *European Conference on Computer Vision*, September 2018.

[LDG18] Y. Liao, S. Donné, and A. Geiger. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.

[LF92] Y. G. Leclerc and M. Fischler. An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3D Wire Frames. *IJCV*, 9(2):113–136, 1992.

[LGF23] Ren Li, Benoît Guillard, and Pascal Fua. ISP: Multi-Layered Garment Draping with Implicit Sewing Patterns. *arXiv Preprint*, 2023.

[LGK$^+$17] Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji, and R. Wang. 3D Shape Reconstruction from Sketches via Multi-View Convolutional Networks. In *International Conference on 3D Vision*, pages 67–77, 2017.

[LGRF22] R. Li, B. Guillard, E. Remelli, and P. Fua. DIG: Draping Implicit Garment over the Human Body. In *ACCV*, 2022.

[LHT$^+$21] Y. Li, M. Habermann, B. Thomaszewski, S. Coros, T. Beeler, and C. Theobalt. Deep physics-aware inference of cloth deformation for monocular human performance capture. In *International Conference on 3D Vision*, 2021.

[LLG$^+$23] Luca De Luigi, Ren Li, Benoît Guillard, Mathieu Salzmann, and Pascal Fua. DrapeNet: Generating Garments and Draping them with Self-Supervision. In *CVPR*, 2023.

[LLK19] J. Liang, M. Lin, and V. Koltun. Differentiable Cloth Simulation for Inverse Problems. In *Advances in Neural Information Processing Systems*, 2019.

[LLL$^+$22] Xiaoxiao Long, Cheng Lin, Lingjie Liu, Yuan Liu, Peng Wang, Christian Theobalt, Taku Komura, and Wenping Wang. NeuralUDF: Learning Unsigned Distance Fields for Multi-view Reconstruction of Surfaces with Arbitrary Topologies. In *CVPR*, 2022.

[LLVT03] 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.

[LMB$^+$14] T.-.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, pages 740–755, 2014.

[LMR$^+$15] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M.J. Black. SMPL: A Skinned Multi-Person Linear Model. *ACM SIGGRAPH Asia*, 34(6), 2015.

[LS96] H. Lipson and M. Shpitalni. Optimization-Based Reconstruction of a 3D Object from a Single Freehand Line Drawing. *Computer-Aided Design*, 28(8):651–663, 1996.

[LW94] H.J. Lamousin and W.N. Waggenspack. Nurbs-Based Free-Form Deformations. *Computer Graphics and Applications*, 16(14):59–65, 1994.

[LWL19] S. Liu, S. A. W.Chen, and H. Li. Learning to Infer Implicit Surfaces Without 3D Supervision. In *NeurIPS*, 2019.

[LZK+21] R. Li, M. Zheng, S. Karanam, T. Chen, and Z. Wu. Everybody Is Unique: Towards Unbiased Human Mesh Recovery. In *BMVC*, 2021.

[LZP+20] 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 *CVPR*, 2020.

[MBM+17] 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*, pages 5425–5434, 2017.

[MCR22] I. Mehta, M. Chandraker, and R. Ramamoorthi. A Level Set Theory for Neural Implicit Evolution under Explicit Flows. In *ECCV*, 2022.

[MESK22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 2022.

[MGT+19] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black. AMASS: Archive of Motion Capture as Surface Shapes. In *ICCV*, pages 5442–5451, 2019.

[MLX+16] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, and Zhen Wang. Multi-class generative adversarial networks with the l2 loss function. *arXiv preprint arXiv:1611.04076*, 2016.

[MM89] J. Malik and D. Maydan. Recovering Three-Dimensional Shape from a Single Image of Curved Objects. *PAMI*, 11(6):555–566, 1989.

[MMNKF18] A. Mosińska, P. Marquez-Neila, M. Kozinski, and P. Fua. Beyond the Pixel-Wise Loss for Topology-Aware Delineation. In *CVPR*, pages 3136–3145, 2018.

[MNSL22] G. Moon, H. Nam, T. Shiratori, and K.M. Lee. 3d clothed human reconstruction in the wild. In *ECCV*, 2022.

[MOHR13] B.R. Munson, T.H. Okiishi, W.W. Huebsch, and A.P. Rothmayer. *Fluid Mechanics*. Wiley Singapore, 2013.

[MON+19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *CVPR*, pages 4460–4470, 2019.

[MPJ+19] M. Michalkiewicz, J.K. Pontes, D. Jack, M. Baktashmotlagh, and A.P. Eriksson. Implicit Surface Representations as Layers in Neural Networks. In *ICCV*, 2019.

[MPT+20] Ben 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 *ECCV*, 2020.

[MT99] T. Mcinerney and D. Terzopoulos. Topology Adaptive Deformable Surfaces for Medical Image Volume Segmentation. *TMI*, 18(10):840–850, 1999.

[MYR+20] Q. Ma, J. Yang, A. Ranjan, S. Pujades, G. Pons-Moll, S. Tang, and M. J. Black. Learning to Dress 3D People in Generative Clothing. In *CVPR*, 2020.

[NDVZJ19] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Transactions on Graphics*, 38(6):1–17, 2019.

[NMOG20] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. In *CVPR*, 2020.

[NPO13] R. Narain, T. Pfaff, and J.F. O'Brien. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics*, 2013.

[NSACO05] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A Sketch-Based Interface for Detail-Preserving Mesh Editing. In *ACM SIGGRAPH*, 2005.

[NSO12] R. Narain, A. Samii, and J.F. O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics*, 2012.

[Nvi18a] Nvidia. Nvcloth, 2018.

[Nvi18b] Nvidia. NVIDIA Flex, 2018. https://developer.nvidia.com/flex.

[NY06] T.S. Newman and H. Yi. A Survey of the Marching Cubes Algorithm. *Computers & Graphics*, 30(5):854–879, 2006.

[OKC+21] 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. *PAMI*, 44(9):5401–5413, 2021.

[P+95] Xavier Provot et al. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*, 1995.

[PFAK20] O. Poursaeed, M. Fisher, N. Aigerman, and V.G. Kim. Coupling Explicit and Implicit Surface Representations for Generative 3D Modeling. In *ECCV*, pages 667–683, 2020.

[PFS+19] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove. Deepsdf: Learning Continuous Signed Distance Functions for Shape Representation. In *CVPR*, 2019.

[PGC+17] 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 *NeurIPS*, 2017.

[Pie91] L. Piegl. On NURBS: A Survey. *Computer Graphics and Applications*, 11:55–71, 1991.

[PJH16] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: from Theory to Implementation.* Morgan Kaufmann, 2016.

[PJL+21] S. Peng, C. Jiang, Y. Liao, M. Niemeyer, M. Pollefeys, and A. Geiger. Shape as Points: A Differentiable Poisson Solver. In *NeurIPS*, 2021.

[PKS+18] J. K. Pontes, C. Kong, S. Sridharan, S. Lucey, A. Eriksson, and C. Fookes. Image2mesh: A Learning Framework for Single Image 3D Reconstruction. In *ACCV*, 2018.

[PLPM20] C. Patel, Z. Liao, and G. Pons-Moll. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *CVPR*, 2020.

[PMJ+22] X. Pan, J. Mai, X. Jiang, D. Tang, J. Li, T. Shao, K. Zhou, X. Jin, and D. Manocha. Predicting loose-fitting garment deformations using bone-driven motion networks. In *ACM SIGGRAPH*, 2022.

[PMPHB17] G. Pons-Moll, S. Pujades, S. Hu, and M.J. Black. Clothcap: Seamless 4D Clothing Capture and Retargeting. *ACM SIGGRAPH*, 36(4):731–7315, July 2017.

[PNJO14] T. Pfaff, R. Narain, J.M. De Joya, and J.F. O'Brien. Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics*, 33(4):1–9, 2014.

[PNM+20] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional Occupancy Networks. In *ECCV*, pages 523–540, 2020.

[PPV19] K. Park, T. Patten, and M. Vincze. Pix2pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation. In *ICCV*, 2019.

[Pro97] X. Provot. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Computer Animation and Simulation*, 1997.

[Qi18] Charles R. Qi. Autoencoder for point clouds. https://github.com/charlesq34/pointnet-autoencoder, 2018.

[RBS+22] Edoardo Remelli, Timur Bagautdinov, Shunsuke Saito, Chenglei Wu, Tomas Simon, Shih-En Wei, Kaiwen Guo, Zhe Cao, Fabian Prada, Jason Saragih, et al. Drivable volumetric avatars using texel-aligned features. In *SIGGRAPH*, 2022.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Conference on Medical Image Computing and Computer Assisted Intervention*, pages 234–241, 2015.

[RLR+20] E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua. Meshsdf: Differentiable Iso-Surface Extraction. In *NeurIPS*, 2020.

[RLT$^+$20] M. Runz, K. Li, M. Tang, L. Ma, C. Kong, T. Schmidt, I. Reid, L. Agapito, J. Straub, S. Lovegrove, and R. Newcombe. Frodo: from Detections to 3D Objects. In *CVPR*, June 2020.

[RR18] S. Richter and S. Roth. Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers. In *CVPR*, 2018.

[RRN$^+$20] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *ACM SIGGRAPH Asia*, 2020.

[RTTP17] E. Remelli, A. Tkach, A. Tagliasacchi, and M. Pauly. Low-Dimensionality Calibration through Local Anisotropic Scaling for Robust Hand Model Personalization. In *ICCV*, 2017.

[Sea16] Seamplex. Fino, a free finite element solver, 2016.

[Set99] 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.

[SF09] M. Salzmann and P. Fua. Reconstructing Sharply Folding Surfaces: A Convex Formulation. In *CVPR*, June 2009.

[SLL20] Y. Shen, J. Liang, and M.C. Lin. Gan-based garment generation using sewing pattern images. In *ECCV*, 2020.

[SMB$^+$20] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *NeurIPS*, 2020.

[SOC19] I. Santesteban, M. A. Otaduy, and D. Casas. Learning-Based Animation of Clothing for Virtual Try-On. *Computer Graphics Forum (Proc. of Eurographics)*, 33(2), 2019.

[SOC22] I. Santesteban, M.A. Otaduy, and D. Casas. SNUG: Self-Supervised Neural Dynamic Garments. In *Conference on Computer Vision and Pattern Recognition*, 2022.

[Sof18] Optitext Fashion Design Software, 2018. https://optitex.com/.

[SOTC22] I. Santesteban, M.A. Otaduy, N. Thuerey, and D. Casas. Ulnef: Untangled layered neural fields for mix-and-match virtual try-on. In *Advances in Neural Information Processing Systems*, 2022.

[SP86] T.W. Sederberg and S.R. Parry. Free-Form Deformation of Solid Geometric Models. *ACM SIGGRAPH*, 20(4), 1986.

[SS11] J. Stam and R. Schmidt. On the Velocity of an Implicit Surface. *ACM Transactions on Graphics*, 30(3):1–7, 2011.

[SSB13] F. S. Sin, D. Schroeder, and J. Barbič. Vega: Non-Linear FEM Deformable Object Simulator. In *CGF*, 2013.

[ST90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990.

[ST03] C. Sminchisescu and B. Triggs. Kinematic Jump Processes for Monocular 3D Human Tracking. In *CVPR*, 2003.

[STH⁺19] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhofer. Deepvoxels: Learning Persistent 3D Feature Embeddings. In *CVPR*, pages 2437–2446, 2019.

[STOC21] I. Santesteban, N. Thuerey, M. A. Otaduy, and D. Casas. Self-Supervised Collision Handling via Generative 3D Garment Models for Virtual Try-On. In *Conference on Computer Vision and Pattern Recognition*, 2021.

[Sun18] Xingyuan Sun. Pix3d: Dataset and methods for single-image 3d shape modeling. https://github.com/xingyuansun/pix3d, 2018.

[SWZ⁺18] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *CVPR*, pages 2974–2983, 2018.

[SZZ⁺18] Tongkui Su, Yan Zhang, Yu Zhou, Yao Yu, and Sidan Du. GPU-based Real-time Cloth Simulation for Virtual Try-on. In *Pacific Conference on Computer Graphics and Applications*, 2018.

[TBTPM20] G. Tiwari, B. L. Bhatnagar, T. Tung, and G. Pons-Moll. Sizer: A Dataset and Model for Parsing 3D Clothing and Learning Size Sensitive 3D Clothing. In *European Conference on Computer Vision*, 2020.

[TDB15] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Single-View to Multi-View: Reconstructing Unseen Views with a Convolutional Network. *CoRR abs/1511.06702*, 1:2, 2015.

[TDB16] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-View 3D Models from Single Images with a Convolutional Network. In *ECCV*, pages 322–337, 2016.

[TDB17] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs. In *ICCV*, 2017.

[TK11] D. Toal and A.J. Keane. Efficient Multipoint Aerodynamic Design Optimization via Cokriging. *Journal of Aircraft*, 48(5):1685–1695, 2011.

[TRR+19] M. Tatarchenko, S. Richter, R. Ranftl, Z. Li, V. Koltun, and T. Brox. What Do Single-View 3D Reconstruction Networks Learn? In *CVPR*, pages 3405–3414, 2019.

[TTN+13] M. Tang, R. Tong, R. Narain, C. Meng, and D. Manocha. A GPU-based streaming algorithm for high-resolution cloth simulation. In *Computer Graphics Forum*, 2013.

[TWL+18] M. Tang, T. Wang, Z. Liu, R. Tong, and D. Manocha. I-Cloth: Incremental Collision Handling for Gpu-Based Interactive Cloth Simulation. In *TOG*, 2018.

[UB18] N. Umetani and B. Bickel. Learning Three-Dimensional Flow for Interactive Aerodynamic Design. *ACM Transactions on Graphics*, 37(4):89, 2018.

[UVL16] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. In *arXiv Preprint*, 2016.

[VKS+21] R. Venkatesh, T. Karmali, S. Sharma, A. Ghosh, R. V. Babu, L. A. Jeni, and M. Singh. Deep Implicit Surface Point Prediction Networks. In *ICCV*, 2021.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. In *Computer Graphics Forum*, 2001.

[VSGC20] R. Vidaurre, I. Santesteban, E. Garces, and D. Casas. Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On. In *Computer Graphics Forum*, 2020.

[VSJ22] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable Signed Distance Function Rendering. *TOG*, 2022.

[VSM+17] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A.C. Courville. Modulating Early Visual Processing by Language. In *Advances in Neural Information Processing Systems*, 2017.

[WCPM18] T. Y. Wang, D. Ceylan, J. Popovic, and N. J. Mitra. Learning a Shared Shape Space for Multimodal Garment Design. In *ACM SIGGRAPH Asia*, 2018.

[WGFB22] Zhen Wei, Benoît Guillard, Pascal Fua, and Michaël Bauerheim. Latent representation of cfd meshes and application to 2d airfoil aerodynamics. In *AIAA Journal*, 2022.

[WKF21] U. Wickramasinghe, G. Knott, and P. Fua. Deep Active Surface Models. In *Conference on Computer Vision and Pattern Recognition*, 2021.

[WLL+21] 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 *NeurIPS*, 2021.

[WRKF20] 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.

[WSL+19] Y. Wang, Y. Sun, Z. Liu, S. Sarma, M. Bronstein, and J.M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. In *TOG*, 2019.

[WWX+17] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3d shape reconstruction via 2.5 d sketches. In *NeurIPS*, pages 540–550, 2017.

[WZL+18] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y. Jiang. Pixel2mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV*, 2018.

[XJX+18] S. Xingyuan, W. Jiajun, Z. Xiuming, Z. Zhoutong, Z. Chengkai, X. Tianfan, J.B. Tenenbaum, and W.T. Freeman. Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling. In *CVPR*, 2018.

[XLY+17] 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*, 12(1):1–14, 01 2017.

[XWC+19] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann. DISN: Deep Implicit Surface Network for High-Quality Single-View 3D Reconstruction. In *NeurIPS*, 2019.

[Yat20] Tatsuya Yatagawa. mcubes_pytorch: PyTorch Implementation for Marching Cubes. 2020.

[YFST18] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Point Cloud Auto-Encoder via Deep Grid Deformation. In *CVPR*, 2018.

[YKM+20] 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 *NeurIPS*, 2020.

[YRSJ21] Yu Y. Rong, T. Shiratori, and H. Joo. Frankmocap: Fast monocular 3d hand and body motion capture by regression and integration. In *ICCVW*, 2021.

[YSW+20] L. Yang, Q. Song, Z. Wang, M. Hu, C. Liu, X. Xin, W. Jia, and S. Xu. Renovating parsing R-CNN for accurate multiple human parsing. In *ECCV*, 2020.

[Zel05] C. Zeller. Cloth simulation on the gpu. In *ACM SIGGRAPH*, 2005.

[ZGZS20] Y. Zhong, Y. Gryaditskaya, H. Zhang, and Y.-Z. Song. Deep Sketch-Based Modelling: Tips and Tricks. In *International Conference on 3D Vision*, 2020.

[ZMGL21] I. Zakharkin, K. Mazur, A. Grigorev, and V. Lempitsky. Point-based modeling of human clothing. In *ICCV*, 2021.

[ZQG$^+$20] Y. Zhong, Y. Qi, Y. Gryaditskaya, H. Zhang, and Y.-Z. Song. Towards Practical Sketch-Based 3D Shape Generation: The Role of Professional Sketches. In *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[ZWLS21] F. Zhao, W. Wang, S. Liao, and L. Shao. Learning Anchored Unsigned Distance Functions with Gradient Direction Alignment for Single-View Garment Reconstruction. In *ICCV*, 2021.

BENOÎT GUILLARD

Final year PhD Student in Computer Vision and Deep Learning

@ benoit.guillard@epfl.ch ☎ 0041.78.848.33.42 ⌂ Lausanne, SWITZERLAND

EDUCATION

2019-present **EPFL, CVLab,** Lausanne, Switzerland
PhD student working with Pr. Pascal Fua on 3D deep learning for surface reconstruction.

2018-2019 **Imperial College,** London, United Kingdom
MSc in Advanced Computing, graduated with distinction, ranked 1st of my cohort.

2015-2018 **École Polytechnique,** Palaiseau, France
MSc, Applied Maths & Computer Science in Image, Computer Vision, Learning. 3.89/4 GPA.

PROFESSIONAL EXPERIENCES

Summer 2022 **Research Intern, Meta Reality Labs**, Zürich, Switzerland
- Deep learning based reconstruction of 3D garments from images.

Summer 2021 **Research Intern, Microsoft Research**, Autonomous Systems (Redmond WA / remote)
- Deep learning based sensor simulation for LiDAR data, presented at the *IROS2022* conference.

Summer 2019 **Research Project, Imperial College,** London, under the supervision of Pr. Paul Kelly
- Ultra-low power and high-throughput analog vision systems, for CNN inference: achieved 96.9% accuracy on MNIST classification, at 2260 fps, using only 0.7 mJ per frame.

Summer 2018 **Research Intern** in Computational Photography**, DxO** (DxOMark sister company), Paris
- Raw camera sensor data processing on GPU, using deep learning for noisy image restoration.

Summer 2017 **Data Analyst Intern,** start-up company **EZsolution**, Ho Chi Minh city, Vietnam
- Designed and delivered a marketing leads' scoring based on machine learning.
- Trained sales representatives to use it.

2016-2017 **Scientific group project**, in collaboration with **EDF**, France's first energy provider
- Led a 6 students team throughout the 8 months project.
- High dimension systems to predict thermal exchanges in a building, based on weather conditions.

2015-2016 **Junior Officer in the French Air Force**, France
- Designed a fault detection alert for very large communication systems.
- Collaborated with Superior Officers to promote it.

SCIENTIFIC PUBLICATIONS

DrapeNet: Generating Garments and Draping them with Self-Supervision, Li, de Luigi et al., CVPR 2023
MeshUDF: Fast and Differentiable Meshing of UDF Networks, Guillard et al., ECCV 2022
Learning to Simulate Realistic LiDARs, Guillard et al., IROS 2022
Sketch2Mesh: Reconstructing and Editing 3D Shapes from Sketches, Guillard et al., ICCV 2021
MeshSDF: Differentiable Iso-Surface Extraction, Remelli et al., NeurIPS 2020
UCLID-Net: Single View Reconstruction in Object Space, Guillard et al., NeurIPS 2020

LANGUAGES AND SKILLS

Speaking: French (*native language*), English (*fluent, TOEFL: 113, GRE: 162/169/4*), German (*basic*)
Programming: mainly: Python with Pytorch (and OpenCV + Tensorflow basics)
+ notions of: C++, Prolog, Caml, Java, PHP, PIC assembly

ACTIVITIES AND INTERESTS

Photography: Analog and digital hobbyist photographer, former beta-tester for a medium-sized company designing leading-edge cameras and image processing software.

AWARDS

2022 – Best reviewer award: CVPR & NeurIPS, leading Computer Vision & Machine Learning conferences.
2019 – Two Winton Capital prizes at Imperial: Research Project Prize & best MSc Student of my program.
2018 – Ranked 4th at Huawei's UK Artificial Intelligence Students Challenge on image denoising.