Learning-based representations and methods for 3D shape analysis, manipulation and reconstruction

Marie-Julie Rakotosaona

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX114

Thèse de doctorat

ÉCOLE POLYTECHNIQUE

IP PARIS

Learning-based representations and methods for 3D shape analysis, manipulation and reconstruction

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 10 Décembre 2021, par

MARIE-JULIE RAKOTOSAONA

Composition du Jury :

Stefanie Wuhrer Chargée de recherche, INRIA Grenoble Rhône-Alpes (Morpheo Team)	Rapporteur
Mathieu Aubry Professeur, École des Ponts ParisTech (Imagine team, LIGM lab)	Rapporteur
Jean Ponce Directeur de recherche, Inria	Président
Niloy Mitra Professeur, University College London (UCL)	Examinateur
Nicolas Bonneel Chargé de recherche, CNRS (ORIGAMI team)	Examinateur
Maks Ovsjanikov Professeur, École Polytechnique (LIX)	Directeur de thèse

Contents

Abstract

Efficiently processing and analysing 3D data is a crucial challenge in modern applications as 3D shapes are becoming more and more widespread with the proliferation of acquisition devices and modeling tools. While successes of 2D deep learning have become commonplace and surround our daily life, applications that involve 3D data are lagging behind. Due to the more complex non-uniform structure of 3D shapes, successful methods from 2D deep learning cannot be easily extended and there is a strong demand for novel approaches that can both exploit and enable learning using geometric structure. Moreover, being able to handle the various existing representations of 3D shapes such as point clouds and meshes, as well as the artefacts produced from 3D acquisition devices increases the difficulty of the task. In this thesis, we propose systematic approaches that fully exploit geometric information of 3D data in deep learning architectures. We contribute to point cloud denoising, shape interpolation and shape reconstruction methods. We observe that deep learning architectures facilitate learning the underlying surface structure on point clouds that can then be used for denoising as well as shape interpolation. Encoding local patch-based learned priors, as well as complementary geometric information such as edge lengths, leads to powerful pipelines that generate realistic shapes. The key common thread throughout our contributions is facilitating seamless conversion between different representations of shapes. In particular, while using deep learning on triangle meshes is highly challenging due to their combinatorial nature we introduce methods inspired from geometry processing that enable the creation and manipulation of triangle faces. Our methods are robust and generalize well to unseen data despite limited training sets. Our work, therefore, paves the way towards more general, robust and universally useful manipulation of 3D data.

Chapter 1

Introduction

The field of AI and deep learning, for analysing 2D images has recently led to tremendous achievements for image manipulation [128], understanding [96] and generation [101, 140]. Even more impressively, these advances are currently being used in real life applications ranging from retail to healthcare as well as banking and agriculture. Surprisingly methods for processing 3D data are not encountering such success. Analogous applications on shape manipulation [125], understanding or generation [16, 15] are lagging behind, failing to produce similar quality results. This naturally raises the question: *How can we achieve similar success in 3D data applications?* Despite representing the same environment and objects, 3D data are fundamentally different from 2D images. Intuitively, since they combine both appearance and geometry, 3D shapes carry richer information than 2D images. However they do not enjoy the same canonical representation that can be found in a grid of pixels for instance. Popular representations of shapes such as point clouds, a set of unordered data points in space, or triangle meshes, a set of triangles that are connected by common edges, do not present a clear ordering or structure that suits existing deep learning architectures as shown in Figure 1.1. Therefore, a potential cause to the discrepancy in performance lies in the fact that current 3D methods tend to focus on extending successful approaches from 2D vision to 3D without exploiting the inherently different nature of 3D shapes. We are interested in exploring methods and representations that process data in accordance with their specific nature and properties. Furthermore, in addition to their particular geometric nature, 3D data present further challenges in their representation that are twofold: (i) different historical source of 3D data and tasks has led to numerous representations of shapes (point clouds, meshes, voxels) (ii) due to hardware limitations, input data is often noisy and fails to capture high frequency details.

1.1 Background

With the ubiquitous use of 3D data in many fields and industries such as entertainment, cultural heritage, geo-exploration, architecture, and urban modeling, it is essential to analyse and process such data before they can be used in downstream tasks. The discipline of *geometry processing [34]* deals with mathematical models and algorithms for analyzing and manipulating geometric data. Typical operations include surface reconstruction from point samples (Figure 1.3), filtering operations for noise removal (Figure 1.2), geometric data analysis and manipulation [206], shape simplification [6], and geometric modeling and interactive design [35].

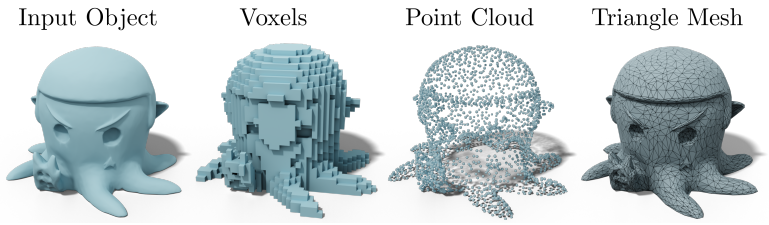

Figure 1.1 – Main representations of 3D shapes. While point clouds and triangle meshes allow highly detailed representations of shapes, they do not enjoy an ordered grid structure such as 2D images or voxels. While voxels are structured as grids, they have limited accuracy for the same memory budget.

Traditionally, 3D data are represented via numerous representations. The most commonly used representations have naturally evolved from the various data sources and processing operations on 3D data. While many of the essential operations from geometry processing remain indispensable today, and are even becoming more abundant with the emergence of new fields such as automatic shape generation, existing representations can hinder progress on these same applications as they are not well suited for deep learning applications as we highlight in Section 1.1.1.

In this thesis we are interested in learning-based algorithms that consume 3D information obtained from acquisition devices and produce well-structured representations of the underlying shapes.

1.1.1 Operations and Representations on 3D shapes

Shape Registration and Reconstruction Low cost sensors and acquisition methods have recently become widely available, making it easy to acquire raw 3D data. Popular tools such as Microsoft Kinect [109], photogrammetry or lidars produce RGBD images and point clouds that suffer from noise and outliers due to limitations and inherent noise of sensors and their environment [110, 187]. In order to correctly process 3D shapes in later stages, it is necessary to denoise point clouds and possibly generate more structured representations.

For reconstructing 3D shapes, we distinguish two preferred representations: point clouds and meshes. Due to their simplicity, compactness, flexibility and powerful representation capability point clouds are an immensely popular representation often used for shape understanding [184, 235], shape generation and reconstruction [222]. A major challenge in generating point clouds is that contrary to 2D images, the points do not follow any canonical ordering, which makes extending existing methods intended for structured data not straightforward. Furthermore, point clouds lack connectivity information that characterize surfaces.

On the other hand, meshes remain a preferred representation especially for efficiently storing and rendering 3D models. Meshes enable efficient and accurate representations of 3D shapes as they can adapt to different levels of detail in separate regions and are

well suited for tasks such as rendering due to their explicit representation of the surface. However, due to their complexity it remains a challenge to apply deep learning methods on meshes. They are often irregular and the combinatorial nature of the connectivity information between vertices makes it difficult to represent them in a differentiable way.

Shape Manipulation and Generation A core problem in 3D computer vision is to build models that can generate new, diverse and realistic shapes. In particular, morphing and manipulating shapes to produce new models or animations from interpolations or operations on existing shapes is a core problem that has been investigated in many recent works [125, 188, 158, 19] as well as more classical approaches [138, 117]. Classical works from geometry processing often formulate the task of interpolating shapes as a constrained energy based minimization problem where physical characteristics such as volume or shell deformations are preserved. While such methods can produce very high quality results, they are often complex, expensive and rely on using meshes with one-to-one correspondence. Recently, learning based methods have emerged that process point clouds that are not necessarily in one-to-one correspondence efficiently. However, these methods can lead to distortions during interpolations [104, 188] as they lack surface information about the shape. We are interested in the problem of realistic shape interpolation from point cloud data that are not in one-to-one correspondence and can be noisy or incomplete.

1.1.2 Deep Learning on 3D data

Early learning based methods on 3D data handle multiple representations: voxels [212], point clouds [235, 184], meshes [111] for tasks such as shape classification or part segmentation. While they do produce good results in such tasks, they are often not well adapted to handle tasks from geometry processing such as denoising, reconstruction or shape generation that require dealing with dense representations, going between different representations, and understanding and respecting geometric properties.

We make the distinction between recent high-level operations extended from 2D deep learning such as classification or segmentation problems that require a high-level understanding of shapes, and shape analysis and processing operations that rely on processing the geometric information of surfaces in different representations. We show that deep learning is a powerful tool for processing geometric information on 3D shapes. In turn, integrating geometry leads to more efficient representations for high-level operations such as shape generation.

Finally, while successful applications of 2D deep learning rely on very large datasets that contain up to 14 million images [62], annotated 3D datasets are often much smaller such as ShapeNet [44] that only contain 51 thousand models. The scarcity of 3D data compared to 2D images, is a major limitation that we address by infusing geometric priors at different scales to produce more robust models.

1.2 Contributions

We organize the contributions of this thesis in three main parts.

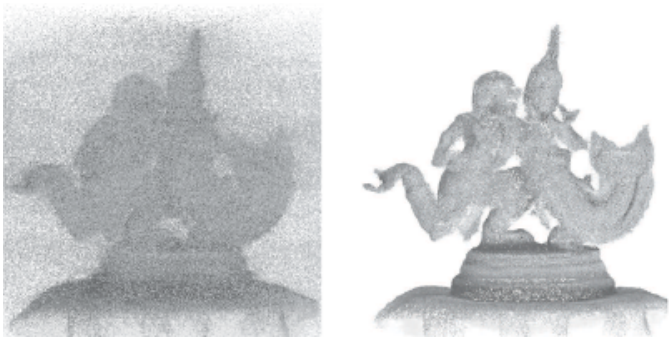

Figure 1.2 – Noisy point clouds with outliers (left). We remove outliers and project the remaining points to recover the underlying clean surface (right).

Part 1: Point Cloud Denoising As mentioned above, learning to denoise point clouds is a key challenge for producing accurate reconstructions. Traditional methods for point cloud denoising largely rely on local surface fitting (e.g., jets or MLS surfaces), local or non-local averaging, or on statistical assumptions about the underlying noise model. In Chapter 3, we develop a simple learning-based method for removing outliers and reducing noise in unordered point clouds. Our approach is efficient and robust to varying amounts of noise and outliers, while being able to handle large densely-sampled point clouds.

While the method described in Chapter 3 helps producing clean point clouds from noisy data, point cloud representations still lack information about the underlying surface, and specifically the *connectivity* between points. This fact motivates the next parts, namely the need to encode intrinsic information as well as explicitly retrieving the said connectivity in the form of meshes.

Part 2: Intrinsic shape representations While being widely used and available, point cloud data does not encode any connectivity between points. Thus, methods trained on point clouds can by their nature be insensitive to distortions that might appear on generated shapes. This raises the following question: *Can we learn to encode edge or intrinsic information into shape representations computed from point clouds?* In Chapter 4, we develop a representation and an architecture that allow to encode intrinsic shape information and drastically improve the quality of reconstructions and interpolations of shapes. Our approach is based on constructing a dual encoding space that enables shape synthesis and, at the same time, provides links to the intrinsic shape information, which is typically not available on point cloud data. Furthermore, the strong regularization provided by our dual latent space approach also helps to improve shape recovery in challenging settings from noisy point clouds across different datasets.

Chapter 4 demonstrates that we can learn to enrich existing representations with learned intrinsic information. However, our method relies on a fixed template to generate surfaces and does not adapt to new topologies which is the main goal of our final part.

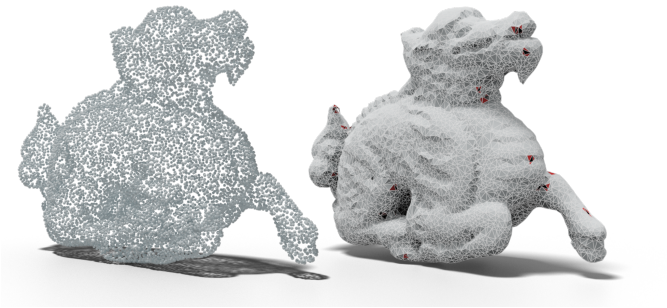

Figure 1.3 – Given an input clean point cloud (left), we learn triangles between input points that best interpolate the surface (right).

Part 3: Differentiable meshing Generating high quality meshes efficiently from noisy input is an essential task in 3D computer vision. However, their combinatorial nature is a substantial obstacle to using them in learning-based methods. We aim at generating meshes in an end-to-end manner since, as highlighted above, meshes are highly efficient shape representations. In Chapter 5 and Chapter 6 we develop key tools for enabling end-to-end meshing of point clouds and surfaces.

In Chapter 5 , we introduce a method for reconstructing triangle meshes from point clouds. Existing learning-based methods for mesh reconstruction mostly generate triangles individually, making it hard to create manifold meshes. We leverage the properties of 2D Delaunay triangulations to construct a mesh from manifold surface elements. While our method is currently the state of the art for similar methods, developing methods that are fully differentiable is a crucial challenge for the future.

In Chapter 6, we present a differentiable approach for manipulating the topology of triangles meshes. Our approach allows to optimize meshes using any differentiable objective function by computing a soft existence score associated with the triangles of the mesh. Our approach does not require any post-processing or combinatorial operations such as edge flips or vertex splits.

1.3 List of publications

This dissertation is based on the following publications:

1. M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, *PointCleanNet: Learning to denoise and remove outliers from dense point clouds*, in Computer Graphics Forum, vol. 39, Wiley Online Library, 2020, pp. 185-203. [187]

2. M.-J. Rakotosaona and M. Ovsjanikov, *Intrinsic point cloud interpolation via dual latent space navigation*, in European Conference on Computer Vision, Springer, 2020, pp. 655-672. [188]

3. M.-J. Rakotosaona, P. Guerrero, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, *Learning delaunay surface elements for mesh reconstruction*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 22-31. [186]

4. M.-J. Rakotosaona, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, P. Guerrero, *Differentiable surface triangulation*, ACM Transactions on Graphics (TOG), (2021). [185]

The following works have also been developed during my PhD and are closely related to the topics mentioned above. However, they are not included in this dissertation:

1. R. Huang*, M.-J. Rakotosaona*, P. Achlioptas, L. J. Guibas, and M. Ovsjanikov, *OperatorNet: Recovering 3d shapes from difference operators*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 8588-8597. [125]

2. A. Poulenard, M.-J. Rakotosaona, Y. Ponty, and M. Ovsjanikov, *Effective rotation-invariant point cnn with spherical harmonics kernels*, in 2019 International Conference on 3D Vision (3DV), IEEE, 2019, pp. 47-56. [182]

3. R. Marin*, M.-J. Rakotosaona*, S. Melzi, and M. Ovsjanikov, *Correspondence learning via linearly-invariant embedding*, in NeurIPS, 2020. [161]

* denotes equal contribution

Chapter 2

Introduction en Français

Le domaine de l'IA et de l'apprentissage profond, pour l'analyse des images 2D, a récemment donné lieu à d'énormes avancements en matière de manipulation d'images [128], de compréhension [96] et de génération [101, 140]. Plus impressionnant encore, ces progrès sont actuellement utilisés dans des applications réelles allant du commerce, à la santé, en passant par le secteur bancaire et l'agriculture. Étonnamment, les méthodes de traitement des données 3D ne rencontrent pas un tel succès. Des applications analogues sur la manipulation des formes [125], la compréhension ou la génération [16, 15] sont à la traîne, ne parvenant pas à produire des résultats de qualité similaire. Cela amène naturellement la question suivante : *Comment pouvons-nous obtenir un succès similaire dans les applications de données 3D ?* Bien qu'elles représentent le même environnement et les mêmes objets, les données 3D sont fondamentalement différentes des images 2D. Intuitivement, puisqu'elles combinent à la fois l'apparence et la géométrie, les formes 3D sont porteuses d'informations plus riches que les images 2D. Cependant, elles ne bénéficient pas de la même représentation canonique que celle que l'on peut trouver dans une grille de pixels par exemple. Les représentations populaires des formes telles que les nuages de points, un ensemble de points non ordonnés dans l'espace, ou les mailles triangulaires, un ensemble de triangles reliés par des arêtes communes, ne présentent pas un ordre ou une structure claire qui convienne aux architectures d'apprentissage profond existantes, comme montré sur la figure 2.1. Ainsi, une cause potentielle de l'écart de performance réside dans le fait que les méthodes 3D actuelles ont tendance à se concentrer sur l'extension des approches fructueuses de la vision 2D à la 3D sans exploiter la nature intrinsèquement différente des formes 3D. Nous nous intéressons à l'exploration de méthodes et de représentations qui traitent les données en respectant leur nature et de leurs propriétés spécifiques. En outre, en plus de leur nature géométrique particulière, les données 3D présentent des défis supplémentaires dans leur représentation: (i) les différentes sources historiques des données et des tâches 3D ont conduit à de nombreuses représentations de formes (nuages de points, maillages, voxels) (ii) en raison des limitations matérielles, les données d'entrée sont souvent bruitées et ne parviennent pas à capturer les détails à haute fréquence.

2.1 Contexte

Avec l'utilisation systématique de données 3D dans de nombreux domaines et industries telles que le divertissement, le patrimoine culturel, la géo-exploration, l'architecture et la modélisation urbaine, il est essentiel d'analyser et de traiter ces données avant de pouvoir les utiliser dans des applications connexes. La discipline du traitement de la géométrie

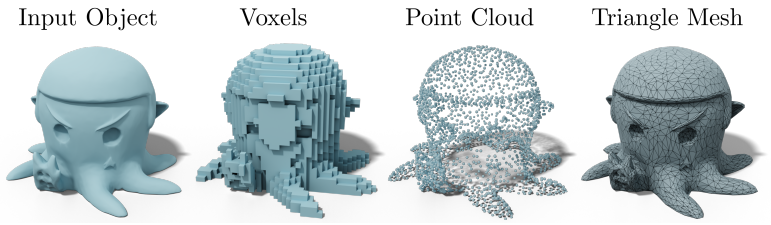

Figure 2.1 – Principales représentations des formes 3D. Si les nuages de points et les mailles triangulaires permettent des représentations très détaillées des formes, ils ne bénéficient pas d'une structure de grille ordonnée comme les images 2D ou les voxels. Bien que les voxels soient structurés comme des grilles, ils ont une précision limitée pour le même budget en mémoire.

(*geometry processing [34]*) traite des modèles mathématiques et des algorithmes d'analyse et de manipulation des données géométriques. Les opérations typiques comprennent la reconstruction de surfaces à partir de nuages de points (figure 2.3), les opérations de filtrage pour la suppression du bruit (figure 2.2), l'analyse et la manipulation de données géométriques [206], la simplification de formes [6], et la modélisation géométrique et la conception interactive [35].

Traditionnellement, les données 3D sont représentées par de nombreuses représentations. Les représentations les plus couramment utilisées ont naturellement évolué à partir de diverses sources de données et opérations de traitement des données 3D. Alors que de nombreuses opérations essentielles du traitement de la géométrie restent indispensables aujourd'hui, et deviennent même plus abondantes avec l'émergence de nouveaux domaines tels que la génération automatique de formes, les représentations existantes peuvent entraver les progrès de ces mêmes applications car elles ne sont pas bien adaptées aux applications d'apprentissage profond, comme nous le soulignons dans la section 2.1.1.

Dans cette thèse, nous nous intéressons aux algorithmes basés sur l'apprentissage qui consomment des informations 3D obtenues à partir de dispositifs d'acquisition et produisent des représentations bien structurées des surfaces concernées.

2.1.1 Opérations et Représentations sur les Formes 3D

Acquisition et reconstruction de formes Des capteurs et des méthodes d'acquisition à faible coût sont récemment devenus largement accessible, ce qui facilite l'acquisition de données 3D brutes. Des outils populaires tels que Microsoft Kinect [109], la photogrammétrie ou les lidars produisent des images RGBD et des nuages de points qui souffrent de bruit et de valeurs aberrantes en raison des limitations et du bruit inhérent aux capteurs et à leur environnement [110, 187]. Afin de traiter correctement les formes 3D dans les étapes ultérieures, il est nécessaire de débruiter les nuages de points et éventuellement de générer des représentations plus structurées.

Pour reconstruire des formes 3D, on distingue deux représentations privilégiées : les

nuages de points et les maillages. En raison de leur simplicité, de leur compacité, de leur flexibilité et de leur puissante capacité de représentation, les nuages de points sont une représentation extrêmement populaire, souvent utilisée pour la compréhension des formes [184, 235], la génération et la reconstruction de formes [222]. Un défi majeur dans la génération de nuages de points est que, contrairement aux images 2D, les points ne suivent aucun ordre canonique, ce qui rend l'extension des méthodes existantes destinées aux données structurées difficile. En outre, les nuages de points manquent d'informations sur la connectivité qui caractérise les surfaces.

D'autre part, les maillages restent une représentation privilégiée, notamment pour le stockage et le rendu efficaces des modèles 3D. Les maillages permettent une représentation efficace et précise des formes 3D car ils peuvent s'adapter à différents niveaux de détail dans des régions distinctes et sont bien adaptés à des tâches telles que le rendu grâce à leur représentation explicite de la surface. Cependant, en raison de leur complexité, l'application de méthodes d'apprentissage profond aux maillages reste un défi. Ils sont souvent irréguliers et la nature combinatoire des informations de connectivité entre les sommets rend difficile leur représentation de manière différentiable.

Manipulation et Génération de Formes L'un des principaux problèmes de la vision par ordinateur en 3D est de construire des modèles capables de générer de nouvelles formes qui sont diverses et réalistes. En particulier, la génération et la manipulation de formes pour produire de nouveaux modèles ou animations à partir d'interpolations ou d'opérations sur des formes existantes est un problème central qui a été étudié dans de nombreux travaux récents [125, 188, 158, 19] ainsi que dans des approches plus classiques [138, 117]. Les travaux classiques de traitement de la géométrie formulent souvent la tâche d'interpolation des formes comme un problème de minimisation sous contrainte basé sur des énergies traduisant des caractéristiques physiques telles que des déformations de volume. Bien que ces méthodes puissent produire des résultats de très haute qualité, elles sont souvent complexes, coûteuses et reposent sur l'utilisation de maillages en correspondance. Récemment, des méthodes basées sur l'apprentissage ont vu le jour et traitent efficacement des nuages de points qui ne sont pas nécessairement en correspondances. Cependant, ces méthodes peuvent conduire à des distorsions lors des interpolations [104, 188] car elles manquent d'informations de surface sur sous-jacente de la forme 3D. Nous nous intéressons au problème d'interpolation de formes à partir de données de nuages de points qui ne sont pas en correspondance et qui peuvent être bruités ou incomplets.

2.1.2 Apprentissage Profond sur des Données 3D

Les méthodes d'apprentissage sur des données 3D permettent de traiter plusieurs représentations : voxels [212], nuages de points [235, 184], maillages [111] pour des tâches telles que la classification de formes ou la segmentation de parties. Bien qu'ils produisent de bons résultats dans ces tâches, ils ne sont souvent pas bien adaptés pour traiter des tâches de traitement de la géométrie telles que le débruitage, la reconstruction ou la génération de formes qui nécessitent de traiter des représentations denses, de passer d'une

représentation à l'autre, et de comprendre et respecter les propriétés géométriques.

Nous faisons la distinction entre les opérations récentes de haut niveau issues de techniques d'apprentissage profond en 2D, telles que les problèmes de classification ou de segmentation qui nécessitent une compréhension de haut niveau des formes, et les opérations d'analyse et de traitement des formes qui reposent sur le traitement des informations géométriques des surfaces dans différentes représentations. Nous montrons que l'apprentissage profond est un outil puissant pour traiter l'information géométrique des formes 3D. En conséquence, l'intégration de la géométrie conduit à des représentations plus efficaces pour les opérations de haut niveau telles que la génération de formes.

Enfin, alors que les applications fructueuses de l'apprentissage profond en 2D reposent sur de très grands jeux de données qui contiennent jusqu'à 14 millions d'images [62], les jeux de données 3D annotés sont souvent beaucoup plus restreints, comme ShapeNet [44] qui ne contient que 51 000 modèles. La rareté des données 3D par rapport aux images 2D est une limitation majeure que nous abordons en apprenant des données géométriques à différentes échelles pour produire des modèles plus robustes.

2.2 Contributions

Nous organisons les contributions de cette thèse en trois parties principales.

Partie 1 : Débruitage des nuages de points Comme nous l'avons mentionné plus haut, apprendre à débruiter les nuages de points est un défi majeur pour produire des reconstructions précises. Les méthodes traditionnelles de débruitage des nuages de points reposent en grande partie sur l'ajustement des surfaces locales (par exemple, les jets ou les surfaces MLS), le moyennage local ou non local, ou sur des hypothèses statistiques concernant le modèle de bruit sous-jacent. Dans le chapitre 3, nous développons une méthode simple basée sur l'apprentissage pour supprimer les valeurs aberrantes et réduire le bruit dans les nuages de points non ordonnés. Notre approche est efficace et robuste face à des quantités variables de bruit et de valeurs aberrantes, tout en étant capable de traiter des nuages de points denses.

Bien que la méthode décrite au chapitre 3 permette de produire des nuages de points non bruités à partir de données bruitées, les représentations de nuages de points manquent toujours d'informations sur la surface sous-jacente, et plus particulièrement sur la *connectivité* entre les points. Ce constat motive les parties suivantes, à savoir la nécessité d'encoder des informations intrinsèques ainsi que de récupérer explicitement ladite connectivité sous forme de mailles.

Partie 2 : Représentations intrinsèques de formes Bien qu'elles soient largement utilisées et accessibles, les données des nuages de points n'encodent aucune connectivité entre les points. Ainsi, les méthodes entraînées sur des nuages de points peuvent par nature être insensibles aux distorsions qui peuvent apparaître sur les formes générées. Cela amène à se poser la question suivante : *Peut-on apprendre à encoder des informations sur la connectivité ou des informations intrinsèques dans les représentations de formes*

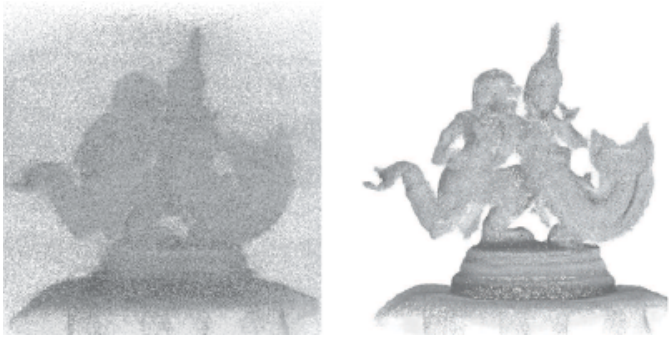

Figure 2.2 – Nuages de points bruités avec des points aberrants (à gauche). Nous supprimons les points aberrants et projetons les points restants pour retrouver la surface non bruitée (à droite).

calculées à partir de nuages de points ? Dans le chapitre 4, nous développons une représentation et une architecture qui permettent d'encoder les informations intrinsèques des formes et d'améliorer drastiquement la qualité des reconstructions et interpolations de formes. Notre approche est basée sur la construction d'un espace d'encodage double qui permet la synthèse de formes et, en même temps, fournit des liens vers les données intrinsèques de la forme , qui ne sont généralement pas disponible sur les données de nuages de points. En outre, la forte régularisation fournie par notre approche de l'espace latent double permet également d'améliorer la reconstruction des formes dans des contextes complexes à partir de nuages de points bruités dans différents ensembles de données.

Le chapitre 4 démontre que nous pouvons apprendre à enrichir des représentations existantes avec des informations intrinsèques apprises. Cependant, notre méthode s'appuie sur un maillage fixe pour générer des surfaces et ne s'adapte pas à de nouvelles topologies, ce qui est l'objectif principal de notre dernière partie.

Partie 3 : Maillage différentiable La génération efficace de maillages de haute qualité à partir d'une entrée bruitée est une tâche essentielle en vision par ordinateur 3D. Cependant, leur nature combinatoire est un obstacle important à leur utilisation dans des méthodes basées sur l'apprentissage. Nous cherchons à générer des maillages de bout en bout car, comme nous l'avons souligné plus haut, les maillages sont des représentations de forme très efficaces. Dans le chapitre 5 et le chapitre 6, nous développons des outils clés pour permettre le maillage de bout en bout de nuages de points et de surfaces.

Dans le chapitre 5 , nous présentons une méthode de reconstruction de maillages triangulaires à partir de nuages de points. Les méthodes existantes basées sur l'apprentissage pour la reconstruction de maillages génèrent généralement les triangles individuellement, ce qui rend difficile la création de maillages bien structurés. Nous tirons parti des propriétés des triangulations de Delaunay en 2D pour construire un maillage à partir d'éléments de surface bien structurés. Bien que notre méthode soit actuellement l'état de l'art pour des méthodes similaires, le développement de méthodes différentiables de bout

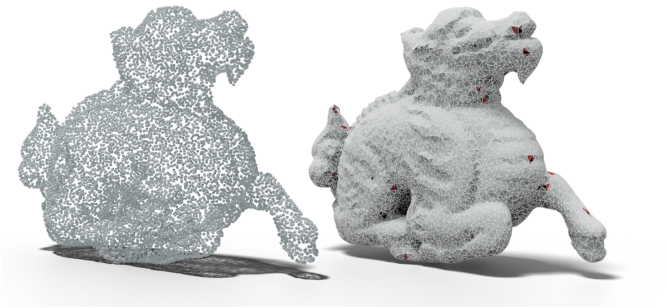

Figure 2.3 – Étant donné un nuage de points non bruité en entrée (à gauche), nous prédisons les triangles entre les points qui interpolent le mieux la surface (à droite).

en bout est un défi crucial pour l'avenir.

Dans le chapitre 6, nous présentons une approche différentiable pour manipuler la topologie des maillages de triangles. Notre approche permet d'optimiser les mailles en utilisant n'importe quelle fonction objectif différentiable en calculant un score d'existence continu associé aux triangles de la maille. Notre approche ne nécessite pas de post-traitement ou d'opérations combinatoires telles que des retournements d'arêtes.

2.3 Liste des publications

Cette thèse s'appuie sur les publications suivantes :

1. M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, *PointCleanNet: Learning to denoise and remove outliers from dense point clouds*, in Computer Graphics Forum, vol. 39, Wiley Online Library, 2020, pp. 185-203. [187]

2. M.-J. Rakotosaona and M. Ovsjanikov, *Intrinsic point cloud interpolation via dual latent space navigation*, in European Conference on Computer Vision, Springer, 2020, pp. 655-672. [188]

3. M.-J. Rakotosaona, P. Guerrero, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, *Learning delaunay surface elements for mesh reconstruction*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 22-31. [186]

4. M.-J. Rakotosaona, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, P. Guerrero, *Differentiable surface triangulation*, ACM Transactions on Graphics (TOG), (2021). [185]

Les travaux suivants ont également été développés pendant mon doctorat et sont étroitement liés aux sujets mentionnés ci-dessus. Cependant, ils ne sont pas inclus dans cette thèse :

1. R. Huang*, M.-J. Rakotosaona*, P. Achlioptas, L. J. Guibas, and M. Ovsjanikov, *OperatorNet: Recovering 3d shapes from difference operators*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 8588-8597. [125]

2. A. Poulenard, M.-J. Rakotosaona, Y. Ponty, and M. Ovsjanikov, *Effective rotation-invariant point cnn with spherical harmonics kernels*, in 2019 International Conference on 3D Vision (3DV), IEEE, 2019, pp. 47-56. [182]

3. R. Marin*, M.-J. Rakotosaona*, S. Melzi, and M. Ovsjanikov, *Correspondence learning via linearly-invariant embedding*, in NeurIPS, 2020. [161]

* dénote une contribution égale

Chapter 3

PointCleanNet: Learning to Denoise and Remove Outliers from Dense Point Clouds

Point clouds obtained with 3D scanners or by image-based reconstruction techniques are often corrupted with significant amount of noise and outliers. Traditional methods for point cloud denoising largely rely on local surface fitting (e.g., jets or MLS surfaces), local or non-local averaging, or on statistical assumptions about the underlying noise model. In contrast, we develop a simple data-driven method for removing outliers and reducing noise in unordered point clouds. We base our approach on a deep learning architecture adapted from PCPNet, which was recently proposed for estimating local 3D shape properties in point clouds. Our method first classifies and discards outlier samples, and then estimates correction vectors that project noisy points onto the original clean surfaces. The approach is efficient and robust to varying amounts of noise and outliers, while being able to handle large densely-sampled point clouds. In our extensive evaluation, both on synthesic and real data, we show an increased robustness to strong noise levels compared to various state-of-the-art methods, enabling accurate surface reconstruction from extremely noisy real data obtained by range scans. Finally, the simplicity and universality of our approach makes it very easy to integrate in any existing geometry processing pipeline. Both the code and pre-trained networks can be found on the project page[1].

3.1 Introduction

Raw 3D point clouds obtained directly from acquisition devices such as laser scanners or as output of a reconstruction algorithm (e.g., image-based reconstruction) are regularly contaminated with noise and outliers. The first stage of most geometry processing workflows typically involves *cleaning* such raw point clouds by discarding the outlier samples and denoising the remaining points to reveal the (unknown) scanned surface. The clean output is then used for a range of applications like surface reconstruction, shape matching, model retrieval, etc.

Any good point cloud cleanup algorithm should (i) *balance between denoising and feature-preservation*, i.e., remove outliers and noise while retaining data fidelity by preserving sharp edges and local details of the underlying scanned surface; (ii) *be self-tuning*, i.e., not require as input precise estimates of the noise model or statistics of the

[1] https://github.com/mrakotosaon/pointcleannet

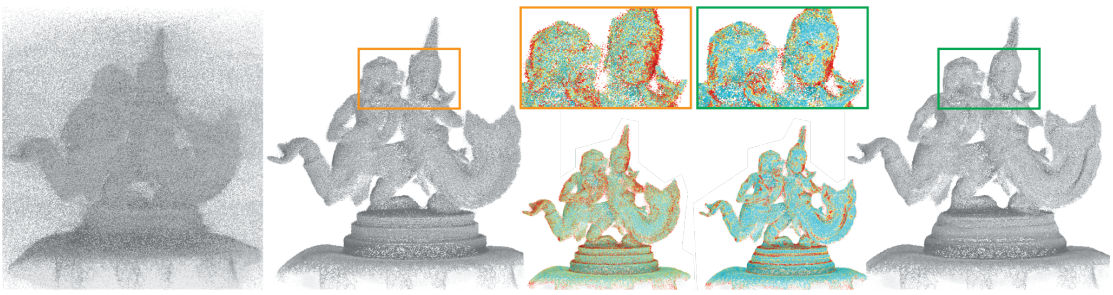

Figure 3.1 – We present PointCleanNet, a two-stage network that takes a raw point cloud (left) and first removes outliers (middle) and then denoises the remaining pointset (right). Our method, unlike many traditional approaches, is parameter-free and automatically discovers and preserves high-curvature features without requiring additional information about the underlying surface type or device characteristics. Here, point clouds are colored based on error compared to the ground truth point cloud (blue denoting low error, red denoting high error).

unknown scanned surface (e.g., local surface type or curvature characteristics); (iii) *be invariant to permutation and rigid transform applied to the pointset*, i.e., the denoised output should not depend on angle of scanning or choice of coordinate system; and (iv) *avoid unnecessarily degrading the input*, i.e., leave the points on the scanned surface if the input happens to be noise-free. Note that the last criterion implies that the algorithm should not oversmooth the output if the algorithm is iterated multiple times.

Decades of research have produced many variants of denoising approaches targeted for different surface types and noise models (see survey [110]). Such approaches can be broadly categorized as: classifying points as outliers using statistical methods, e.g., [3]; projecting points to estimated local surfaces (e.g., MLS surface, jet-fitting, etc.) [86, 41, 42]; consolidating similar patches to cancel out iid noise perturbations (e.g., non-local means, dictionary-based sparse coding), e.g., [70]; or, local smoothing using auxiliary input information (e.g., bilateral smoothing) [123], among many others. Unfortunately, there is no single winner among these methods. The choice of algorithm and its parameters often depends on the scanned surface and the noise characteristics of the acquisition setup. Given that the complexity of the underlying geometry and the noise characteristics are, at best, partially known at acquisition time, choosing an optimal algorithm with associated parameters is typically an iterative trial-and-error process.

Inspired by the recent successes of applying deep learning techniques for the analysis and processing of geometric data, including [162, 37, 218] among many others, and especially the seminal works designed for learning directly on point clouds [183, 216], in this paper, we present PointCleanNet, a simple data-driven denoising approach. Specifically, we design a two stage point cloud cleaning network based on the recently proposed PCPNet architecture [107] to estimate robust local features and use this information to denoise the point cloud. At training time, a variety of surface patches extracted from a set of shapes is synthetically corrupted with outliers and noise of varying magnitudes (including zero noise). This artificially corrupted set is then used to

train PointCleanNet. Our two-stage method first removes outlier samples and then estimates correction vectors for the remaining points. Figure 6.1 shows an example on a raw real-world scanned point cloud from the ETHZ dataset [221].

The process is enabled by a novel loss function that effectively cleans pointsets without requiring explicit information about the underlying surface or noise characteristics. Intuitively, the network learns to identify local noise-free patches based on estimated features extracted from corresponding raw pointsets and proposes per-point correction vectors. In other words, the network implicitly builds a dictionary of local surface patches in the form of local learned features and uses it to classify input points as outliers and project the remaining ones onto an ensemble of dictionary patches. At test time, our denoising network directly consumes raw input point clouds, classifies and discards outlier measurements, and denoises the remaining points. The approach is simple to train and use, and does not expect the user to provide parameters to characterize the surface or noise model. Additionally, unlike traditional approaches, our denoising network can easily be adapted to particular shape families and non-standard noise models.

We qualitatively and quantitatively evaluate PointCleanNet on a range of synthetic datasets (with access to groundtruth surfaces) and real world datasets. In our extensive tests, our approach performed better than a variety of state-of-the-art denoising approaches (even with manually-tuned parameters) across both shape and medium to high noise variations. Additionally, the simplicity and universality of our approach makes it very easy to integrate in any existing geometry processing workflow.

3.2 Related Work

Point cloud denoising and outlier removal have a long and rich history in diverse areas of computer science and a full overview is beyond the scope of the current article. Below, we briefly review the main general trends for addressing these problems, while concentrating on solutions most closely related to ours, and refer the interested reader to a recent survey [110].

Outlier removal The earliest classical approaches for outlier detection, classification and removal have been proposed primarily in the statistics and data mining communities, in the general setting of point clouds in arbitrary dimensions, with several monographs dedicated specifically to this topic [181, 18, 189, 3]. These methods are typically based on robust local statistics and most often come with rigorous theoretical guarantees. At the same time, their generality often comes at a cost, as purely statistical methods are often not adapted to the specific features found in geometric 3D shapes, and in most cases require non-trivial parameter tuning.

More recently, several approaches have been proposed for outlier detection, with emphasis on utility for 3D point clouds, arising e.g., from acquisition data, including [46, 108, 221]. The two former methods are implemented in widely used libraries such as CGAL and have also been used in the context of surface reconstruction from noisy point clouds [95]. These approaches are very robust, but are also based on setting critical parameters or rely on using additional information such as color [221]. This makes it

difficult to apply them, for example, across general noise models, without additional user input and tuning of parameters.

Local surface fitting, bilateral filtering Denoising and outlier removal also arise prominently, and have therefore been considered in the context of surface fitting to noisy point clouds, including the widely-used Moving Least Squares (MLS) approach and its robust variants [4, 164, 86, 177, 103]. Similarly, other local fitting approaches have also been used for point cloud denoising, using robust jet-fitting with reprojection [41, 42] or various forms of bilateral filtering on point clouds [123, 73], which take into account both point coordinates and normal directions for better preservation of edge features. A closely related set of techniques is based on sparse representation of the point normals for better feature preservation [12, 199, 163]. Denoising is then achieved by projecting the points onto the estimated local surfaces. These techniques are very robust for small noise but can lead to significant over smoothing or over-sharpening for high noise levels [163, 110].

Non-local means, dictionary-based methods Another very prominent category of methods, inspired in part from image-based techniques consist in using non-local filtering based most often on detecting similar shape parts (patches) and consolidating them into a coherent noise-free point cloud [64, 239, 70, 71, 234]. Closely related are also methods, based on constructing "dictionaries" of shapes and their parts, which can then be used for denoising and point cloud filtering, e.g., [231, 74] (see also a recent survey of dictionary-based methods [146]). Such approaches are particularly well-suited for feature-preserving filtering and avoid excessive smoothing common to local methods. At the same time, they also require careful parameter setting and, as we show below, are difficult to apply across a wide variety of point cloud noise and artefacts.

Denoising in images Denoising has also been studied in depth in other domains such as for images, with a wide variety of techniques based on both local filtering, total variation smoothing and non-local including dictionary-based methods [38, 81, 43, 159, 80].

More recently, to address the limitations mentioned above, and inspired by the success of deep learning for other tasks, several learning-based denoising methods have also been proposed for both images [236, 237, 131] and more recently meshes [213, 21], among others. These methods are especially attractive, since rather than relying on setting parameters, they allow the method to learn the correct model from data and adapt for the correct noise setting at test time, without any user intervention. In signal processing literature, it is widely believed that image denoising has reached close to optimal performance [45, 147]. One of our main motivations is therefore to show the applicability of this general idea, and especaially the supervised approaches such as [213], that learn them from a set of noisy meshes and their ground-truth counterparts, to the setting of 3D point clouds.

Learning in Point Clouds Learning-based approaches, and especially those based on deep learning, have recently attracted a lot of attention in the context of Geometric Data Analysis, with several methods proposed specifically to handle point cloud data, including PointNet [183] and several extensions such as PointNet++ [184] and Dynamic Graph CNNs [216] for shape segmentation and classification, PCPNet [107] for normal and curvature estimation, P2P-Net [230] and PU-Net [232] for cross-domain point cloud transformation and upsampling respectively. Other, convolution-based architectures have also been used for point-based filtering, including most prominently the recent

PointProNet architecture [190], designed for consolidating input patches, represented via height maps with respect to a local frame, into a single clean point set, which can be used for surface reconstruction. Although such an approach has the advantage of leveraging image-based denoising solutions, error creeps in in the local normal estimation stage, especially in the presence of noise and outliers.

Unlike these techniques, our goal is to train a general-purpose method for removing outliers and denoising point clouds, corrupted with potentially very high levels of structured noise. For this, inspired by the success of PCPNet [107] for normal and curvature estimation, we propose a simple framework aimed at learning to both classify outliers and to displace noisy point clouds by applying an adapted architecture to point cloud patches. We show through extensive experimental evaluation that our approach can handle a wide range of artefacts, while being applicable to dense point clouds, without any user intervention.

3.3 Overview

As a first step in digitizing a 3D object, we usually obtain a set of approximate point samples of the scanned surfaces. This *point cloud* is typically an intermediate result used for further processing, for example to reconstruct a mesh or to analyze properties of the scanned object. The quality of these downstream applications depends heavily on the quality of the point cloud. In real-world scans, however, the point cloud is usually degraded by an unknown amount of outliers and noise. We assume the following point cloud formation model:

$$\mathbb{P}' = \{p'_i\} = \{p_i + n_i\}_{p_i \in \mathbb{P}} \cup \{o_j\}_{o_j \in \mathbb{O}}, \tag{3.1}$$

where $\mathbb{P}'$ is the observed noisy point cloud, $\mathbb{P}$ are perfect surface samples (i.e., $p_i \in \mathcal{S}$ lying on the scanned surface $\mathcal{S}$), n_i is additive noise, and $\mathbb{O}$ is the set of outlier points. We do not make any assumptions about the noise model n or the outlier model $\mathbb{O}$. The goal of our work is to take the low-quality point cloud $\mathbb{P}'$ as input, and output a higher quality point cloud closer to $\mathbb{P}$, that is better suited for further processing. We refer to this process as *cleaning*. We split the cleaning into two steps: first we remove outliers, followed by an estimation of per-point displacement vectors that denoise the remaining points:

$$\tilde{\mathbb{P}} = \{p'_i + d_i\}_{p'_i \in \mathbb{P}' \setminus \tilde{\mathbb{O}}}, \tag{3.2}$$

where $\tilde{\mathbb{P}}$ is the output point cloud, d are the displacement vectors and $\tilde{\mathbb{O}}$ the outliers estimated by our method. We first discuss our design choices regarding the desirable properties of the resulting point cloud and then how we achieve them.

Approach. Traditional statistical scan cleaning approaches typically make assumptions about the scanned surfaces or the noise model, which need to be manually tuned by the user to fit a given setting. This precludes the use of these methods by non-expert users or in casual settings. One desirable property of any cleaning approach is therefore

robustness to a wide range of conditions without the need for manual parameter tuning. Recently, deep learning approaches applied to point clouds [183, 184, 216, 107] have shown a remarkable increase in robustness compared to earlier hand-crafted approaches. Most of these methods perform a *global* analysis of the point cloud and produce output that depends on the whole point cloud. This is necessary for global properties such as the semantic class, but is less suited for tasks that only depend on local neighborhoods; processing the entire point cloud simultaneously is a more challenging problem, since the network needs to handle a much larger variety of shapes compared to working with small local patches, requiring more training shapes and more network capacity. Additionally, processing *dense* point clouds becomes more difficult, due to high memory complexity. In settings such as ours, local methods such as PCPNet [107] perform better. Both steps of our approach are based on the network architecture described in this method, due to its relative simplicity and competitive performance. We adapt this architecture to our setting (Section 3.4) and train it to perform outlier classification and denoising.

While our cleaning task is mainly a local problem, the estimated displacement vectors d need to be consistent across neighborhoods in order to achieve a smooth surface. With a local approach such as PCPNet, each local estimate is computed separately based on a different local patch. The difference in local neighborhoods causes inconsistencies between neighboring estimates that can be seen as *residual noise* in the result (see Figure 3.3). We therefore need a method to coordinate neighboring results. We observed that the amount of difference in local neighborhoods between neighboring estimates correlates with the noise model. Thus, the resulting residual noise has a similar noise model as the original noise, but with a smaller magnitude. This means we can iterate our network on the residual noise to keep improving our estimates. See Figure B.8 for an overview of the full denoising approach. We will provide extensive experiments with different numbers of denoising iterations in Section 6.4.

Desirable properties of a point cloud. The two stages (i.e., outlier classification and denoising) of our method use different loss functions. The properties of our denoised point cloud are largely determined by these loss functions. Thus, we need to design them such that their optimium is a point cloud that has all desirable properties. We identify two key desirable properties: First, *all points should be as close as possible to the original scanned surface*. Second, *the points should be distributed as regularly as possible on the surface*. Note that we do not want the denoised points to exactly undo the additive noise and approximate the original perfect surface samples, since the component of the additive noise that is tangent to the surface cannot be recovered from the noisy point cloud. Section 3.5 describes our loss functions, and in Section 6.4, we compare several alternative loss functions.

3.4 Cleaning Model

As mentioned above, our goal is to take a noisy point cloud $\mathbb{P}'$ and produce a cleaned point cloud $\tilde{\mathbb{P}}$ that is closer to the unknown surface that produced the noisy samples. We treat

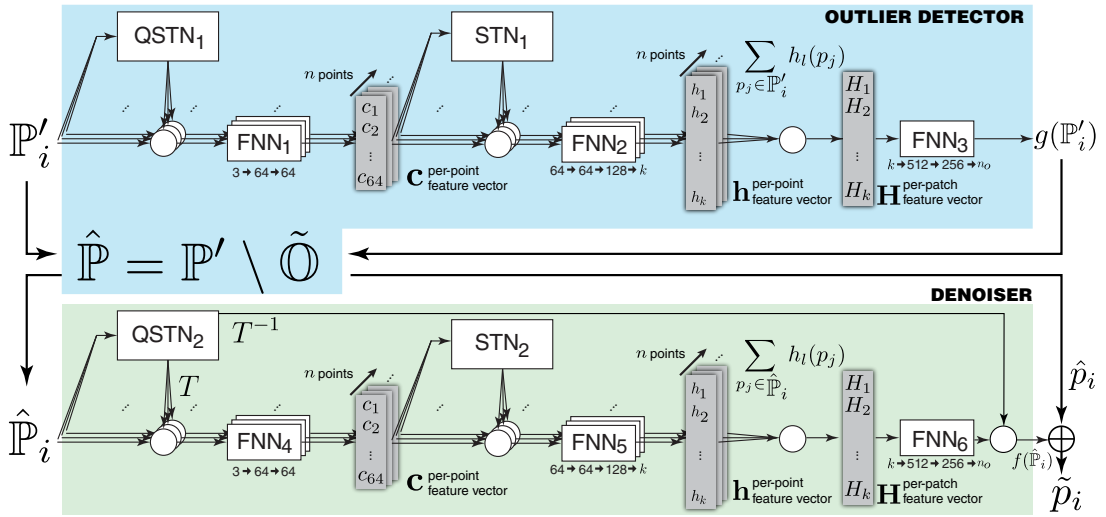

Figure 3.2 – Our two-stage point cloud cleaning architecture. (Top) Given a noisy point cloud $\mathbb{P}'$, we first apply a local outlier detection network that uses an architecture based on PointNet [183] and PCPNet [107] to detect and remove outliers to obtain $\hat{\mathbb{P}}$ (bottom). We then apply a second network, with a similar architecture, but a different loss, aimed at reducing the noise level in $\hat{\mathbb{P}}$ by estimating correcting displacement vectors, which results in the denoised point cloud $\tilde{\mathbb{P}}$. FNN and (Q)STN stand for fully connected and (Quaternion) Spatial Transformer networks[129], similar to their definition and use in PCPNet [107].

denoising as a local problem: the result for each point $p'_i \in \mathbb{P}'$ only depends on a local neighborhood $\mathbb{P}'_i$ of radius r around the point. Focusing on local neighborhoods allows us to handle dense point clouds without losing local detail. Increasing the locality (or scale) radius r provides more information about the point cloud, at the cost of reducing the capacity available for local details. Unlike traditional analytic denoising approaches, a single neighborhood setting is robust to a wide range of noise settings, as we will demonstrate in Section 6.4. In all of our experiments we set r to 5% of the point cloud's bounding box diagonal.

We assume the point cloud formation model described in Equation (3.1), i.e., the noisy point cloud consists of surface samples with added noise and outliers. We then proceed in two stages: first, we train a non-linear function g that removes outliers:

$$\tilde{o}_i = g(\mathbb{P}'_i),$$

where $\tilde{o}_i$ is the estimated probability that point p'_i is an outlier. We add a point to the set of estimated outliers $\tilde{\mathbb{O}}$ if $\tilde{o}_i > 0.5$. After removing the outliers, we obtain the point cloud $\hat{\mathbb{P}} = \mathbb{P}' \setminus \tilde{\mathbb{O}}$. We proceed by defining a function f that estimates displacements for these remaining points to move them closer to the unknown surface:

$$d_i = f(\hat{\mathbb{P}}_i).$$

The final denoised points are obtained by adding the estimated displacements to the

remaining noisy points: $\tilde{p}_i = \hat{p}_i + d_i$. Both f and g are modeled as deep neural networks with a PCPNet-based architecture. We next provide a short overview of PCPNet before describing our modifications.

A major challenge when applying deep learning methods directly to point clouds is achieving invariance to the permutation of the points: all permutations should produce the same result. Training a network to learn this invariance is difficult due to the exponential number of such permutations. As a solution to this problem, PointNet [183] proposes a network architecture that is order-invariant by design. However, PointNet is a global method, processing the whole point cloud in one forward iteration of the network. This results in a degraded performance for shape details. PCPNet [107] was proposed as a local variant of PointNet that is applied to local patches, gives better results for shape details, and is applicable to dense point clouds, possibly containing millions of points. We base our denoising architecture on PCPNet.

Creating a local patch. Given a point cloud $\mathbb{P} = \{p_1, \ldots, p_n\}$, the local patch $\mathbb{P}_i$ contains all the points within the constant radius r inside a ball centered around p_i. Using this patch as input, we want to compute the outlier probability $\tilde{o}_i$ and a displacement vector d_i, for the remaining non-outlier points. We first normalize this patch by centering it and scaling it to unit size. The PCPNet architecture requires patches to have a fixed number of points; like in the original paper, we pad patches with too few points with zeros and take a random subset from patches with too many points. Intuitively, this step, makes the network more robust to additional points.

Network architecture. An overview of our network architecture is shown in Figure B.8. Given the normalized local patch $\mathbb{P}_i$, the network first applies a spatial transformer network [129] that is constrained to rotations, called a quaternion spatial transformer network (QSTN). This is a small sub-network that learns to rotate the patch to a canonical orientation (note that this estimation implicitly learns to be robust to outliers and noise, similar to robust statistical estimation). At the end of the pipeline, the final estimated displacement vectors are rotated back from the canonical orientation to world space. The remainder of the network is divided into three main parts:

- a *feature extractor* $h(p)$ that is applied to each point in the patch separately,
- a *symmetric operation* $H(\mathbb{P}_i) = \sum_{p_j \in \mathbb{P}_i} h(p_j)$ that combines the computed features for each point into an order-invariant feature vector for the patch, and
- a *regressor* that estimates the desired properties d_i and $\tilde{o}_i$ from the feature vector of the patch.

Following the original design of PointNet [183], the feature extractor is implemented with a multi-layer perceptron that is applied to each point separately, but shares weights between points. Computing the features separately for each point ensures that they are invariant to the ordering of the points. The feature extractor also applies an additional spatial transformer network to intermediate point features.

In our implementation, we add skip connections to the multi-layer perceptrons, similar to ResNet blocks. Empirically, we found this to help with gradient propagation and improve training performance. The regressor is also implemented with a multi-layer perceptron. Similar to the feature extractor, we add skip connections to help gradient propagation and improve training performance. We use the same network width as in the original PCPNet (please refer to the original paper for details). However, the network is two times deeper as we replace the original layers with two layers ResBlocks. This architecture is used to compute both outlier indicators and displacement vectors. We change the number of channels of the last regressor layer to fit the size of the desired output (1 for outlier indicators and 3 for displacement vectors).

Importantly, for a each point p_i in the point cloud, we compute its local neighborhood $\mathbb{P}_i$ and only estimate the outlier probability and displacement vector for the center point p_i, i.e., we do not estimate outlier probabilities or displacement vectors for other points in the patch. Thus, each point in the original point cloud is processed independently by considering its own neighborhood and indirectly gets coupled by the iterative cleaning, as described next.

Iterative cleaning. At test time, after applying the displacement vectors computed from a single iteration of the architecture, we are left with residual noise. The residual error vectors from denoised points $\tilde{p}_i$ to the target surface that are introduced by our method do not vary smoothly over the denoised points. Empirically, we found that this residual noise has a similar type, but a lower magnitude than the original noisy points. Intuitively, this can be explained by looking at the content of input patches for points that are neighbors in the denoised point cloud. As shown in Figure 3.3, input patches that are far apart have different content, resulting in different network predictions, while patches that are close together have similar content, and similar predictions. The distance of these input patches correlates with the noise model and the noise magnitude, therefore the network predictions, and the denoised points, are likely to have noise of a similar type, but a lower magnitude than the original noisy points. This allows us to iterate our denoising approach to continue improving the denoised points.

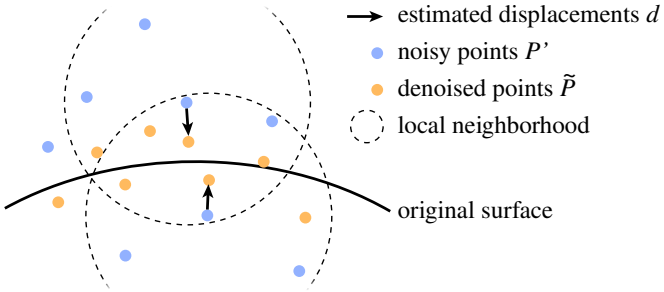

Figure 3.3 – Residual noise. Different local neighborhoods for adjacent denoised points cause slightly different results, which can be seen as residual noise in the denoised points. Iterating the denoising approach improves the results.

In practice, we observed shrinking of the point cloud after several iterations. To counteract this shrinking, we apply an inflation step after each iteration, inspired by Taubin smoothing [201]:

$$d'_i = d_i - 1/k \sum_{p_j \in N(p_i)} d_j, \tag{3.3}$$

where d'_i are the corrected displacements vectors and $N(p_i)$ are the k nearest neighbours of point p_i, we set $k = 100$. Note that this step approximately removes the low-frequency component from the estimated displacements.

3.5 Training Setup

To train the denoising model, we use a dataset of paired noisy point clouds and corresponding clean ground truth point clouds. We do not need to know the exact correspondences of points in a pair, but we assume we do know the ground truth outlier label for each noisy point. Using a point cloud as ground truth instead of a surface description makes it easier to obtain training data. For example, a ground truth point cloud can be obtained from a higher-quality scan of the same scene the noisy point cloud was obtained from. Since we work with local patches instead of entire point clouds, we can train with relatively few shapes. To handle different noise magnitudes, and to enable our iterative denoising approach, we train with multiple noise levels. This includes several training examples with zero noise magnitude, which trains the network to preserve the shape of point clouds without noise.

Loss function Choosing a good loss function is critical, as this choice has direct impact on the properties of the cleaned point clouds. For the outlier removal phase, we use the L_1 distance between the estimated outlier labels and the ground truth outlier labels:

$$L_o(\tilde{p}_i, p_i) = \|\tilde{o}_i - o_i\|_1, \tag{3.4}$$

where $\tilde{o}_i$ is the estimated outlier probability and o_i is the ground truth label. We also experimented with the binary cross-entropy loss, but found the L_1 loss to perform better, in practice.

In the denoising setting, designing the loss function is less straight-forward. Two properties we would like our denoised point clouds to have are proximity of the points to the scanned surface, and a regular distribution of the points over the surface. Assuming the ground truth point cloud has both of these properties, a straight-forward choice for the loss would be the $L2$ distance between the cleaned and the ground truth point cloud:

$$L_c(\tilde{p}_i, p_i) = \|\tilde{p}_i - p_i\|_2^2, \tag{3.5}$$

where $\tilde{p}_i$ and p_i are corresponding cleaned and ground truth points in a patch. Note that, for simplicity of notation, we have unrolled the displacement vector expressions directly in terms of point coordinates. However, this assumes knowledge of a point-wise correspondence between the point clouds; and even if the correspondence is known, we can in general not recover the component of the additive noise that is tangent to the

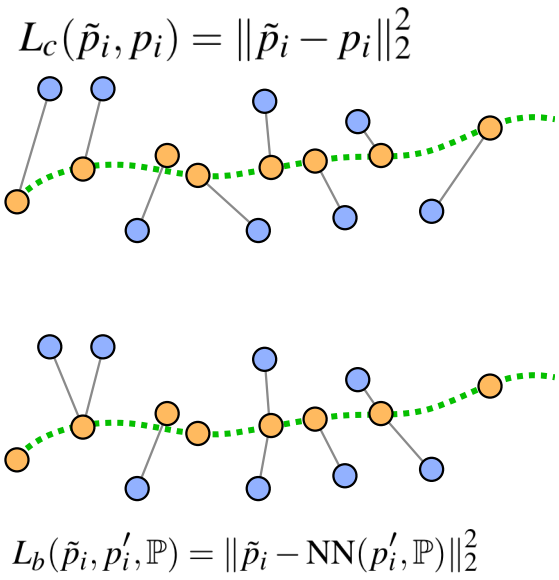

Figure 3.4 – Alternate loss functions that result in comparatively worse performance (see Figure 3.12). Dotted green line denotes the underlying scanned surface, orange points denote original points, and blue points denote the noisy points. The error function L_c (top) tries to learn denoising as denoised points $\tilde{p}_i$ going back to the original ground truth points p_i; while, the error function L_b tries to learn denoising as denoised points $\tilde{p}_i$ going to the closest point in the cleaned point set $\mathbb{P}$, i.e., $\mathrm{NN}(p_i', \mathbb{P})$.

surface. The minimizer of this loss is therefore an average between all potential candidates the noisy point may have originated from. This average will in general not lie on the surface, and lead to poor overall performance. Figure 3.4, top, illustrates this baseline loss. Fortunately, we do not need to exactly undo the additive noise. There is a large space of possible point clouds that satisfy the desired properties to the same degree as the ground truth point cloud, or even more so.

We propose a main loss function and an alternative with a slightly inferior performance, but simpler and more efficient formulation. The main loss function has one term for each of the two properties we would like to achieve: *Proximity to the surface* can be approximated as the distance of each denoised point to its nearest neighbour in the ground truth point cloud:

$$L_s(\tilde{p}_i, \mathbb{P}_{\tilde{p}_i}) = \min_{p_j \in \mathbb{P}_{\tilde{p}_i}} \|\tilde{p}_i - p_j\|_2^2. \tag{3.6}$$

For efficiency, we restrict the nearest neighbor search to the local patch $\mathbb{P}_{\tilde{p}_i}$ of ground truth points centered at $\tilde{p}_i$. Originally, we experimented with only this loss function, but noticed a filament structures forming on the surface after several denoising iterations, as shown in Figure 3.5. Since the points are only constrained to lie on the surface, there are multiple displacement vectors that bring them equally close to the surface. In multiple iterations, the points drift tangent to the surface, forming clusters. To achieve a more

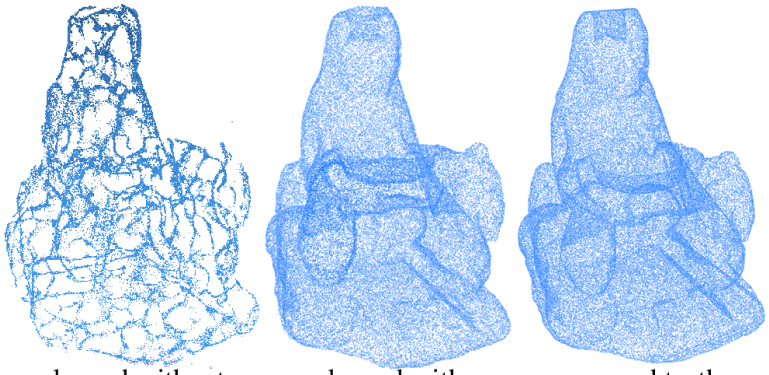

Figure 3.5 – Omitting the regularity term from the loss causes the points to cluster into filament structures on the surface after multiple iterations (left). Compare to results with the regularity term (center) and the ground truth point cloud (right).

regular distribution on the surface, we introduce a regularization term:

$$L_r(\tilde{p}_i, \mathbb{P}_{\tilde{p}_i}) = \max_{p_j \in \mathbb{P}_{\tilde{p}_i}} \|\tilde{p}_i - p_j\|_2^2. \tag{3.7}$$

By minimizing this term, we minimize the squared distance to the *farthest* point in the local patch $\mathbb{P}_{\tilde{p}_i}$. Intuitively, this keeps the cleaned point centered in the patch and discourages a drift of the point tangent to the surface. Assuming the noisy point clouds are approximately regularly distributed, this results in a regular distribution of the cleaned points since, in this case Eq. 3.7 promotes the clean point to lie in the barycenter of the points in its patch. With this term, we want to avoid the excessive clustering of points (for example, into filament structures), which is especially important when applying our approach iteratively. The full loss function is a weighted combination of the two loss terms:

$$L_a = \alpha \; L_s + (1 - \alpha) \; L_r. \tag{3.8}$$

Since the second term can be seen as a regularization, we set α to 0.99 in our experiments.

Importantly, the loss defined in Eq. (3.8) depends on the current point cloud, so that the point searches in Equations (3.6) and (3.7) need to be *updated in every training epoch.* Alternatively, these target points can be fixed. Thus, our alternative loss function uses an explicit ground truth for the cleaned point that can be precomputed:

$$L_b(\tilde{p}_i, p_i', \mathbb{P}) = \|\tilde{p}_i - \mathrm{NN}(p_i', \mathbb{P})\|_2^2, \tag{3.9}$$

where $\mathrm{NN}(p_i', \mathbb{P})$ is the closest point to the initial noisy point p_i' (before denoising) in the ground truth point set $\mathbb{P}$. Figure 3.4, bottom, illustrates this loss. Since both p_i' and the ground truth point cloud are constant during training, this mapping can be precomputed, making this loss function more efficient and easier to implement. Additionally, the fixed target prevents the points from drifting tangent to the surface. However, this loss constrains the network more than L_a and we observed a slightly lower performance.

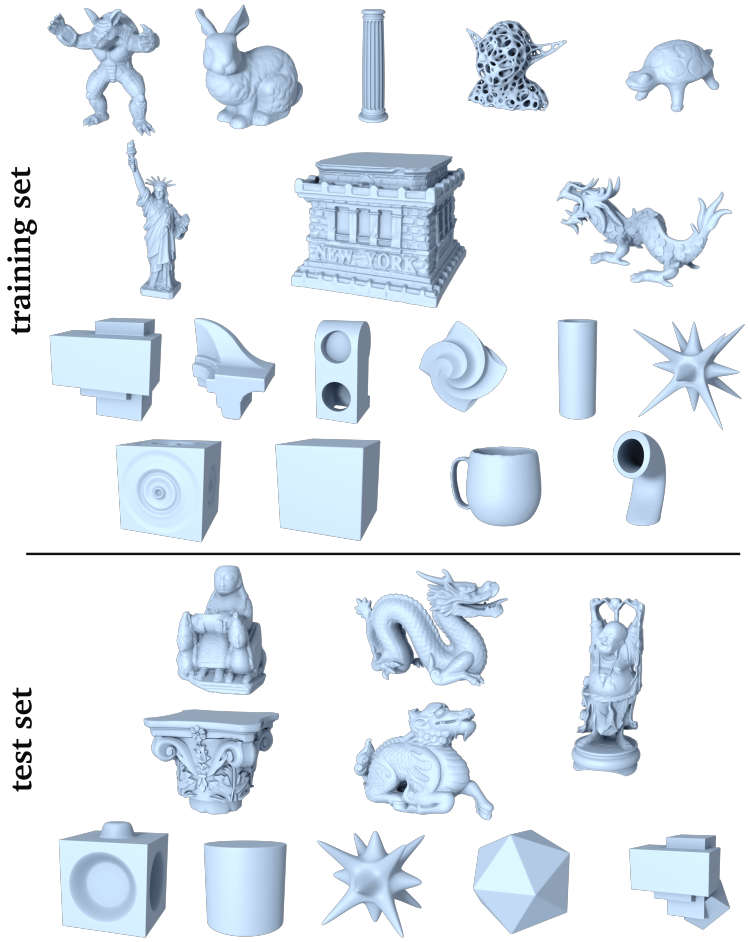

Figure 3.6 – The shapes used for the PointCleanNet training and test sets.

For the outlier removal network we use a learning rate of 10^{-4} and uniform Kaiming initialization [114] of the network weights. When training the denoising network, we observed that network weights converge to relatively small values. To help convergence, we lower the intial values of the weights to uniform random values in $[-0.001, 0.001]$ and decrease the learning rate to 10^{-8}. This improves convergence speed for the denoising network and lowers the converged error.

3.5.1 Relation to PCPNet

While being directly based on PCPNet [107], our approach has several characteristics, specifically adapted to the point cloud denoising and outlier detection problem:

Loss. We use an adapted loss, summarized in Eq. (3.6) and (3.7), which, importantly, not only includes a regularization via the distance to the farthest point, but is also

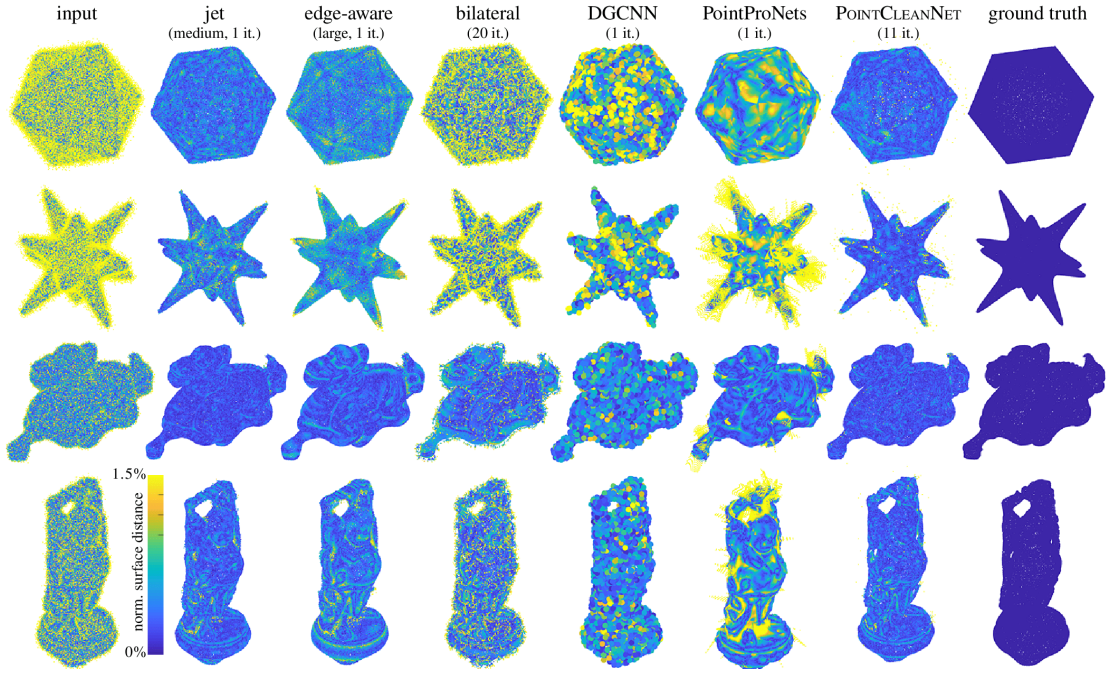

Figure 3.7 – Qualitative comparison to the state-of-the-art. We compare simple shapes in the top row and increase shape complexity towards the bottom. DGCNN can only handle small point clouds, thus we use a sparser sampling for this method. Colors illustrate the denoising error, we use the distance-to-surface for each denoised point.

updated at every training iteration, through the change of the corresponding points. We have experimented with several alternatives such as the loss described in Eq. (3.9) and found them to perform consistently worse than ours.

Iterative deep network. Importantly, we apply our network *iteratively* for improved noise reduction. While perhaps non-standard, this results in very significant improvement in our setting. Moreover, we found that a straightforward implementation might not converge, while with the proper loss and with an inflation term, the network can both stabilize and achieve higher accuracy.

Practical applicability. Finally, we remark that PointCleanNet, as a specialized adaptation of an existing network, both simplifies its integration in practice and establishes its applicability for point cloud denoising and outlier removal.

3.6 Results

We first describe our dataset and evaluation metric in Sections 3.6.1 and 3.6.2. Based on this dataset and metric, we compare the denoising performance (Sec. 3.6.3) and the outlier detection performance (Sec. 3.6.4) of our method to several baselines and state-of-the-art

methods, including the recent learning-based approaches PointProNets [190] and an adapted version of Dynamic Graph CNNs [216] among others. Experiments on additional datasets with different noise distributions, including simulations of non-uniform scanner noise, and noise from real world scans are presented in Section 3.6.5.

3.6.1 Datasets

Our main dataset contains 28 different shapes, which we split into 18 training shapes and 10 test shapes. See Figure 3.6 for a gallery of all shapes. From the original triangle meshes of each shape, we sample 100K points, uniformly at random on the surface, to generate a clean point cloud.

For the *denoising task*, noisy point clouds are generated by adding Gaussian noise with the standard deviation of 0.25%, 0.5%, 1%, 1.5% and 2.5% of the original shape's bounding box diagonal. In total, the denoising training set contains 108 shape variations, arising from 6 levels of noise (including the clean points) for each of the 18 shapes.

For the *outlier removal task*, we use the same training and test shapes, however we use only clean point clouds and with a larger sample count of 140k points per shape. To generate outliers, we added Gaussian noise with standard deviation of 20% of the shape's bounding box diagonal to a random subset of points. The training set contains point clouds with proportions starting at 10% until 90% in intervals of 10% of the points converted to outliers. Only the outliers that are farther from the surface than the standard deviation of the noise distribution are selected. In total, the outlier removal training set contains 432 example shapes, arising from 6 outlier densities and 4 levels of noise for each of the 18 training shapes. The test set contains point clouds with 30% of outliers points. To test the generality of our outlier removal, we added a second method to generate outliers to our test set only. In this setting, outliers are distributed uniformly inside the shape's bounding box that has been scaled up by 10%.

In Section 3.6.5 we also evaluate our method on point clouds generated with alternative methods, including simulated non-uniform noise and noise from real real acquisition devices.

PointCleanNet training datasets for denoising and outlier removal are available on our project page.

3.6.2 Evaluation Metric

The evaluation metric should be sensitive to the desired properties of the point cloud described earlier: point clouds should be close to the surface and have an approximately regular distribution. If we assume the ground truth point clouds have a regular distribution, the following *Chamfer measure* [84, 2], a variant of the Chamfer distance [14], measures both of these properties:

$$c(\tilde{\mathbb{P}}, \mathbb{P}) = \frac{1}{N} \sum_{p_i \in \tilde{\mathbb{P}}} \min_{p_j \in \mathbb{P}} \|p_i - p_j\|_2^2 + \frac{1}{M} \sum_{p_j \in \mathbb{P}} \min_{p_i \in \tilde{\mathbb{P}}} \|p_j - p_i\|_2^2.$$

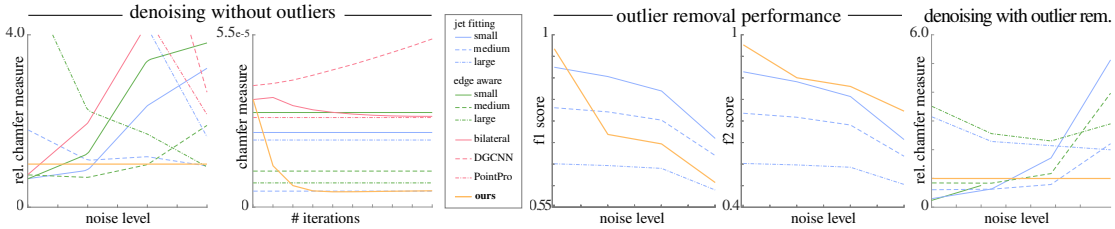

Figure 3.8 – Quantitative comparison. We compare the performance of our model to jet smoothing [41], edge-aware denoising [123], the bilateral point cloud filter by Digne et al. [73], Dynamic Graph CNNs [216], and PointProNets [190]. The two plots on the left are evaluated on our test set without outliers, the two following plots compare the outlier removal performance using the f1 and the f2 scores, and the right-most plot shows the denoising performance after outlier removal.

Here N and M are the cardinalities of the cleaned $\tilde{\mathbb{P}}$ and ground truth $\mathbb{P}$ point clouds, respectively. Note that the first term measures an approximate distance from each cleaned point to the target surface, while the second term intuitively rewards an even coverage of the target surface and penalizes gaps. All our point clouds are scale-normalized to have a unit bounding box diagonal, making the point distances comparable for different shapes.

For a dataset with simulated scanner noise that we will describe in Section 3.6.5, the clean point set has a non-uniform point distribution. For this dataset we use only the root mean square distance-to-surface ($RMSD$) of each point as evaluation metric:

$$d(\tilde{\mathbb{P}}, \mathbb{P}) = \sqrt{\frac{1}{N} \sum_{p_i \in \tilde{\mathbb{P}}} \min_{p_j \in \mathbb{P}} \|p_i - p_j\|_2^2}.$$

3.6.3 Evaluating Denoising

We first evaluate the denoising task alone, without outlier removal. We compare the results of our method on different noise levels to several state-of-the-art techniques for point cloud denoising.

We first consider a qualitative evaluation of our results in Figure 3.7, showing the denoised point clouds for four different input noisy point clouds (Icosahedron, Star smooth, Netsuke, Happy) with two different noise intensities, 1% and 2.5% of the original shape bounding box diagonal. The distances from each of the denoised points to the ground truth surface are color-coded. In the same figure, we can also compare the performance of our method to other successful algorithms.

We compare against five other methods, as described next. It is important to note that in most of these methods, it is necessary to tune some parameters, such as the neighborhood size, to adjust to the different noise levels, while our method works across all noise levels with the same hyper-parameters. When applicable, we manually adjusted parameters for best performance. Also, in some algorithms, we allowed multiple parameter settings (small, medium, large) to handle different levels of noise.

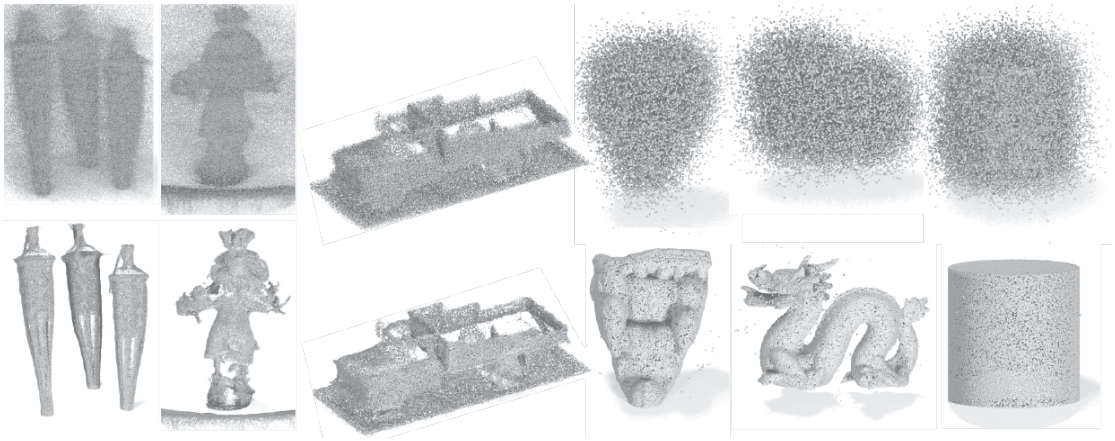

Figure 3.9 – Example input outlier and noise corrupted pointclouds (top) and their corresponding cleaned output (bottom) produced by PointCleanNet. From left to right: torch, scarecrow, tanks-and-temple, galera, dragon, cylinder. The left two examples are corrupted with real-world scanning noise and outliers, the other examples with synthetic noise and outliers.

(i) *Polynomial fitting with osculating jets* [41, 42]: Osculating jet-fitting performs well if the right neighborhood size is chosen for the given noise level. Otherwise, neighborhood sizes that are too small overfit to strong noise (Figure 3.7, first row), and neighborhood sizes that are too large do not preserve detailed features (Figure 3.7, second and fourth row).

(ii) *Edge-aware point set resampling* [123]: Edge-aware point set resampling has larger errors near detailed features (Figure 3.7, third and fourth row), while obtaining good results near sharp edges, like the edges of the icosahedron.

(iii) *Bilateral filtering for point clouds* [73]: bilateral filtering performs poorly in strong noise settings (Figure 3.7, first two rows).

(iv) *Dynamic Graph CNN (DGCNN)* [216]: Note that Dynamic Graph CNNs were not designed for local operations, such as denoising. We modify the segmentation variant of this method to output a displacement vector per point instead of class probabilities. For the loss, the displacements are added to the original points and the result is compared to the target point cloud using the same Chamfer measure used as the error metric in our evaluation. Since the whole point cloud is processed in a single go, we need to heavily sub-sample our dense point clouds before using them as input for DGCNN. We also restrict DGCNN to a single iteration as we found the result set to diverge over iterations. Similar to bilateral filtering, DGCNN also performs poorly in strong noise settings (Figure 3.7, first two rows).

(v) *PointProNets* [190]: PointProNets requires oriented normals during training. Where available, these are obtained from the ground truth source meshes, or estimated

with PCPNet [107] otherwise. Differently from the original method, we also use *ground truth normals* to orient the predicted height maps, instead of trying to estimate an orientation, as we found the in-network estimation used in the original method to be unstable for our datasets. Note that this provides an upper bound for the performance of PointProNets. The denoised patches do not accurately reconstruct detailed surfaces, presumably due to the smoothing effect of the image convolutions, and suffer from artefacts caused by the smoothing of the height map at the boundaries of a patch.

In contrast, our method works on local patches directly in the point domain, can apply several iterations of denoising to improve results, and is robust to a large range of noise levels with the same choice of hyper-parameters. This results in lower residual error, especially in detailed surface regions.

We also present quantitative comparisons that summarize the performance of each method on the entire dataset. The previously described Chamfer measure is used as evaluation metric that captures both the distance from denoised points to the ground truth surface and the regularity of the points. Results are shown in Figure 3.8 left (without outlier removal). We can observe that PointCleanNet performs noticeably better under mid to high noise level and using multiple iterations compared to all the other methods. The performance of our method is also more stable to changes in noise levels, while most other methods perform well only for a specific level of noise.

Comparison to the alternative loss. As shown in Figure 3.12, our alternative loss performs slightly worse than our main loss. However, it is more efficient and easier to implement, so the choice of loss function depends on the setting.

3.6.4 Evaluating Outlier Removal

Figure 3.8 shows the performance of our outlier removal method (right). For the purpose of cleaning dense data, a model should prioritize classifying outlier points correctly (true positives) over limiting the number of false positives. Therefore we consider that recall has more importance than precision for this task. The F_β score conveys the balance between recall and precision. Plots 3 and 4 compare our method to jet-fitting [41] using F_1 and F_2 scores. We observe that when recall and precision are weighted equally, our method has the best performance when removing outliers on clean point clouds while remaining effective on the other noise levels. PointCleanNet performs the best for all noise levels when using F_2 score which gives larger weight to recall.

The last plot in Figure 3.8 compares our approach to jet-fitting and edge-aware filtering [123] with both outlier removal and denoising on the test set. In this experiment, we first removed outliers using an outlier classification technique and then denoised the point clouds from our test set. We show the results for different noise levels from zero to 2.5% of the shape bounding box diagonal. Finally, we make two observations: first, PointCleanNet outperforms edge-aware and jet-fitting techniques with outlier removal and denoising on medium to large noise levels; and second, on smaller noise levels, our

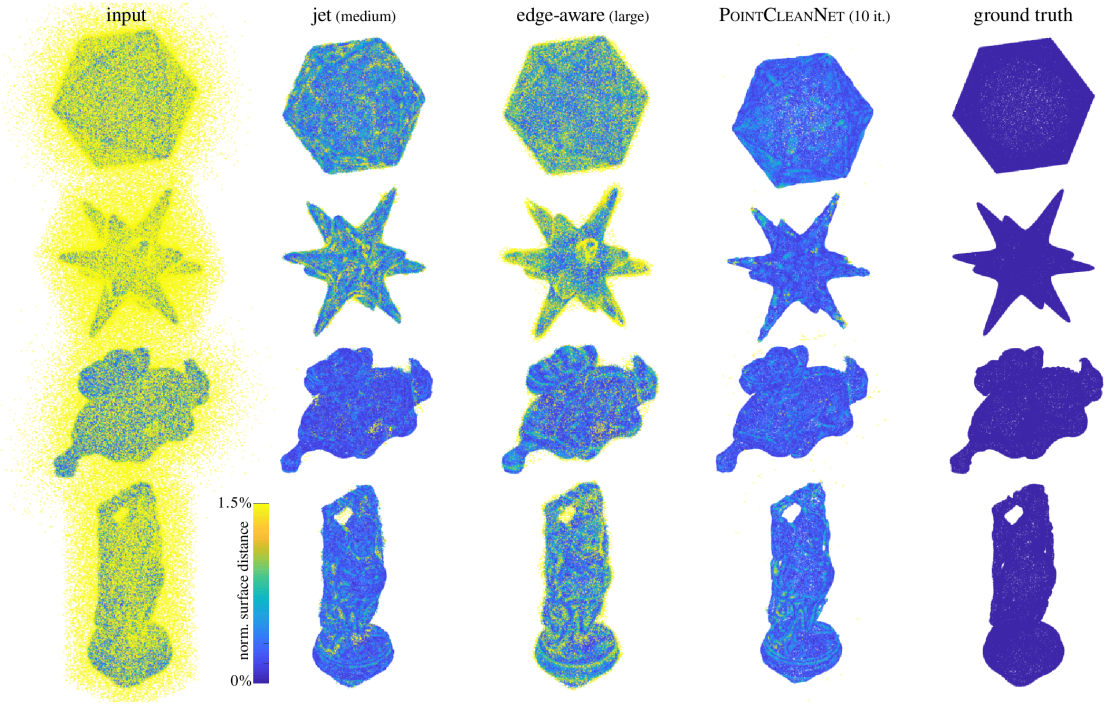

Figure 3.10 – Qualitative comparison on our outlier test set. Here the task is to remove points that are not part of the original surface. Note that analytic methods with a large setting for the radius (third column) fail to remove outliers hidden inside small details, such as the arms of the statue, while a smaller setting (second column) results in a lot of residual noise. Since PointCleanNet can learn to adapt to the feature to produce a result with less noise.

model still outperforms a few of the different tuning variations of the related techniques. Recall that our model does not require parameter tuning by the user.

Figure 3.10 also shows qualitative results for outlier removal on our test set compared to the related techniques mentioned before. We observe that edge-aware filtering performs worse around highly detailed regions and edges, while jet-fitting does not manage to clean the remaining outliers at scattered points. The result highlights the consistent performance of PointCleanNet across different shapes and varying level of details, contrary to the other methods which produce less consistent distances to the underlying ground truth shapes.

3.6.5 Performance under Different Noise Types

Directional noise We evaluated PointCleanNet on a synthetic dataset simulating 3D data acquisition via depth cameras. To do so we created a dataset with structured noise levels to simulate depth uncertainty of depth reconstructions. Specifically, we added noise using an anisotropic Gaussian distribution with constant covariance matrix aligned along the scanning direction. The results are shown in Figure 3.11. Note that our network

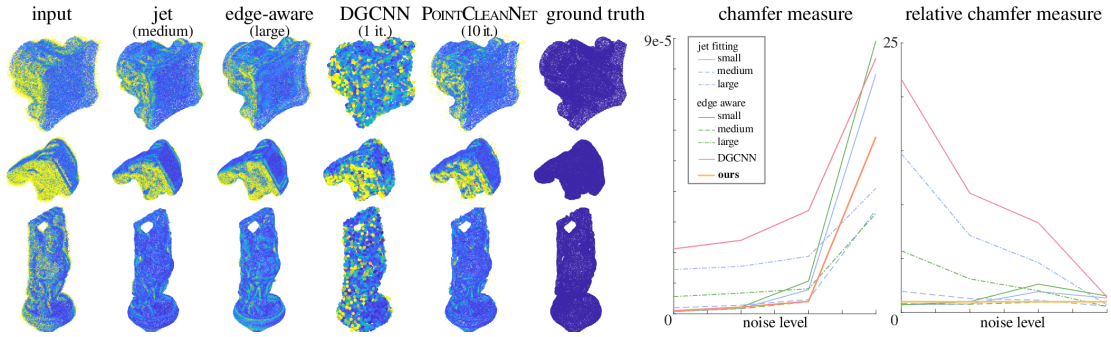

Figure 3.11 – Qualitative comparison on the directional noise test set. We show that PointCleanNet can adapt to different kinds of noise models. Here we added anisotropic noise to the shapes. Qualitative results are o the left, and quantitative plots of the absolute and relative Chamfer measure over different noise levels on the right. Even though our method was not trained on isotropic noise only, it still performs on par with the state of the art on low and medium noise levels.

was *not* retrained for this specific model.

In this setting, the non-data-driven methods, such as jet-fitting and edge-aware filtering, perform well since they are not specialized to any noise model. Even though our method was never trained on this type of noise, it still performs on par with the best methods on low to medium noise settings, and is only outperformed on high noise settings by non-data-driven methods with parameters tuned for the given noise strength.

Structured noise We evaluated our method on a simulated LIDAR dataset (*Velodyne*) generated using BlenSor [106], which models various types of range scanners. We chose to simulate a rotating LIDAR, in particular a Velodyne HDL-64E scanner. BlenSor implements two types of sensor specific error for this scanner: first, a distance bias for each laser unit; and second, a per-ray Gaussian noise. The different effects of the noise types can be observed in Figure 3.13. We use the shapes in PointCleanNet dataset for this experiment. After being normalized, each shape is scanned from $\theta \in [0°, 180°]$ with distance bias with standard deviation 0%, 0.5% and 1% and a per-ray noise of standard deviation 0%, 0.5% and 1%.

Quantitative numbers are presented in Table 3.1. Here, we use the distance to the surface instead of the Chamfer measure, since this type of scanner naturally produces non-uniform point clouds (even in the ground truth). In bold, we highlight the two best performing methods. We evaluate two versions of our method: one version trained on the unstructured noise described in Section 3.6.1, and one version re-trained on the *Velodyne* dataset. The retrained version significantly outperforms all other methods, while the non-retrained version still performs competitively. In Figure 3.14, we show a qualitative evaluation of our results compared to the two other best performing methods. We show the denoised scans for two shapes: cylinder and dragon, with distance bias (first two rows) and with distance bias and per-ray noise (last two rows). Both jet and bilateral

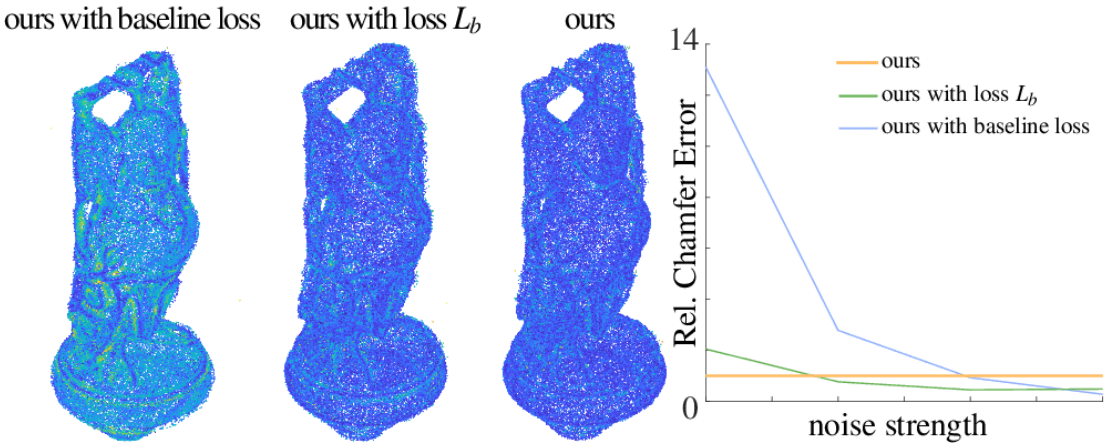

Figure 3.12 – Comparison of our two-term loss L_a with the alternative loss L_b and the baseline loss L_c. Our loss results gives large benefits over the baseline loss, and performs somewhat better than the alternative loss as well, due to being less constrained.

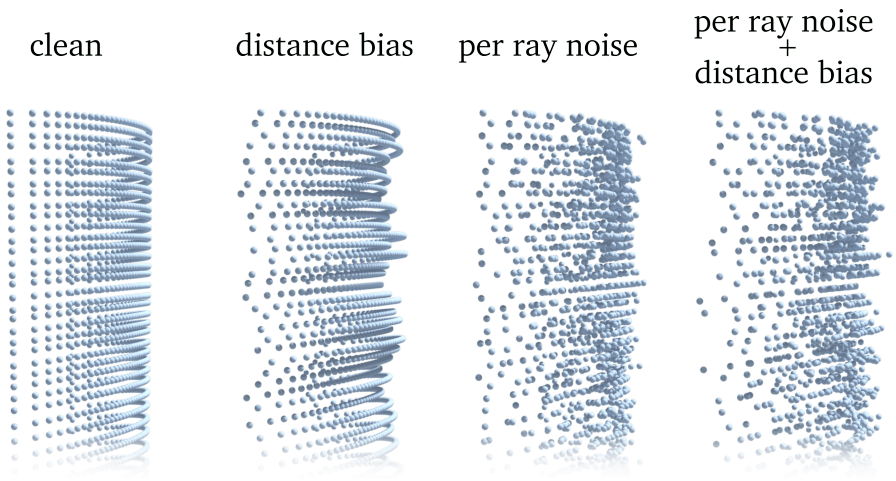

Figure 3.13 – Simulated noise of a Velodyne HDL-64E scanner. Here we show an example of the two noise types introduced by this scanner: distance bias (db) adds an error to the depth of scan lines, while ray noise (rn) adds an error to the depth of each point. Note that the points in the ground truth (left) are not distributed uniformly, so we use the distance of each point to the surface as error measure for experiments with this dataset.

methods preserve an amount of structured noise (see the cylinder shape). Jet medium produces artefact points in areas of high details especially on the dragon shape.

Table 3.2 and Figure 3.15 evaluate our model on the *Kinect v1* dataset introduced in [213]. We trained the model on the scans of the shapes David, big-girl and pyramid from this dataset, and tested on a subset of the scans from boy, girl and cone with a radius of 2.5% of the shapes' bounding box diagonals. Our trained network achieves the best performance. In Figure 3.15, we show the Poisson reconstructions [134] and distances

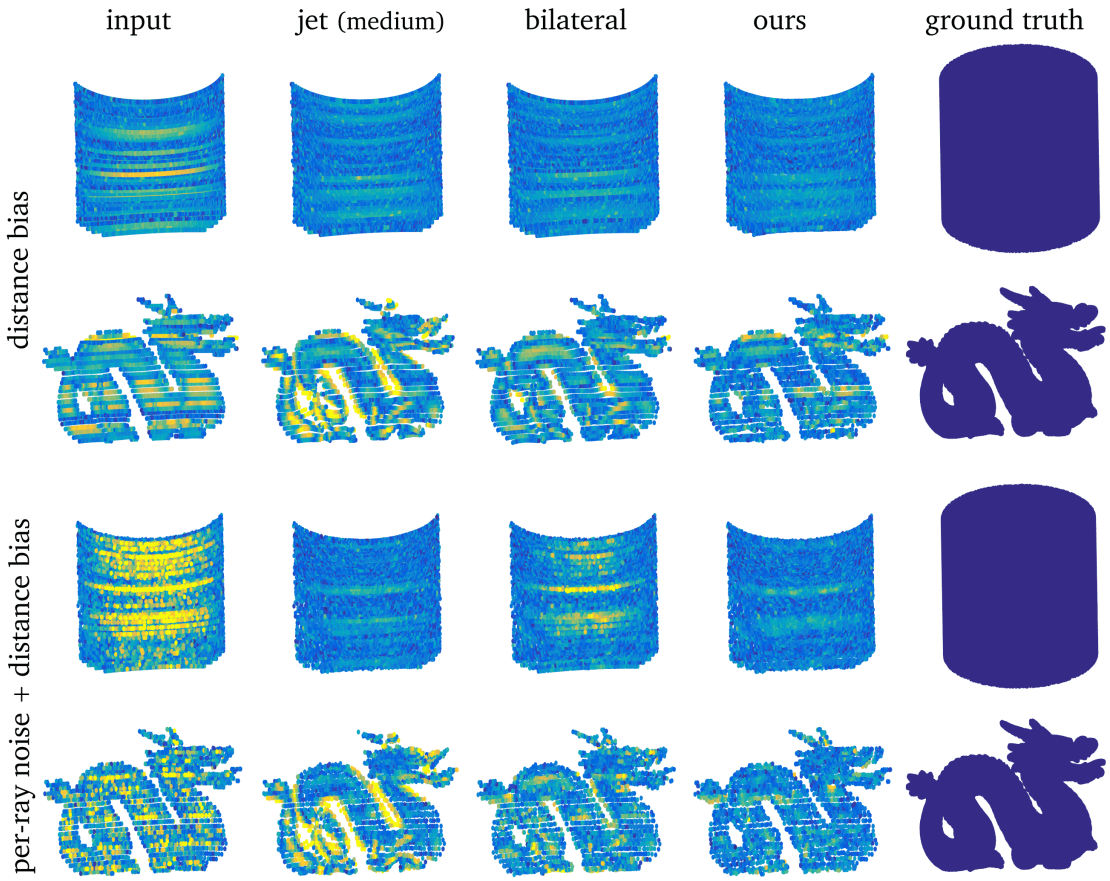

Figure 3.14 – Qualitative comparison with state-of-the-art methods on the *Velodyne* dataset. We display the normalized distance to the ground truth surface. The two top rows are evaluated on a dataset with only distance bias as noise and the two bottom rows with added per-ray noise. The simulated scanner noise has a high spatial correlation along the horizontal scan-lines, and lower correlation vertically across scan-lines. In this setting, jet fitting introduces significant error in detailed surface regions, while bilateral denoising has high residual error in the examples that have both noise types. PointCleanNet successfully learns the noise model, resulting in lower residual error.

from the ground truth for the three best performing methods: jet medium, bilateral and ours. The normals where computed using the normal estimation tool from Meshlab. Note that jet-fitting method tends to preserve the structured noise while the bilateral method tends to oversmooth shape. We also retrained PointCleanNet on the *Kinect v2* dataset described in [213]. As shown in Table 3.2 our method performs well compared to other methods.

Generalization to real-world data. Figure 6.1 and the first two results in Figure 3.9 (left) show the result of our approach on real data obtained with the plane

Table 3.1 – Comparison with state-of-the-art methods on the *Velodyne* datasets. We show the root mean square distance-to-surface (RMSD) for each method. The first column evaluates the methods on a dataset with only distance bias as noise and the second column with added per-ray noise (see Figure 3.13 for examples of the noise types). PointProNets was re-trained on the dataset, and for our method we show both a re-trained version, and a version trained on the original dataset (Section 3.6.1).

	Velodyne (db)	*Velodyne* (db+rn)
jet small	5.46	5.78
jet medium	4.91	5.18
jet large	9.68	9.67
edge-aware small	5.50	6.36
edge-aware med.	5.48	5.77
edge-aware large	11.31	11.53
bilateral	**4.53**	**4.99**
PointProNets	17.47	22.02
ours	5.83	7.03
ours retrained	**4.07**	**4.27**

swift algorithm [221], an image-based 3D reconstruction technique. Statue, torch, and scarecrow input point clouds each contain 1.4M points. Since in this case no ground truth is available, we only show the qualitative results obtained using our method. Note that although trained on an entirely different dataset, PointCleanNet still produces high quality results on this challenging real-world data. The next three shapes in Figure 3.9 show results on other external raw point clouds, while the last one shows a shape with sharp edges.

Table 3.2 – Comparison with state-of-the-art methods on the *Kinect v1* and *Kinect v2* datasets. We show the chamfer measure for each method. PointProNets was re-trained on each dataset, and for our method we show both a re-trained version, and a version trained on the original dataset (Section 3.6.1).

	Kinect v1	*Kinect v2*
jet small	5.10	6.36
jet medium	**4.69**	**6.16**
jet large	5.40	8.63
edge-aware small	5.21	6.38
edge-aware med.	4.78	6.53
edge-aware large	6.85	13.10
bilateral	4.72	**6.04**
PointProNets	7.39	12.81
ours	5.02	6.42
ours retrained	**4.57**	6.26

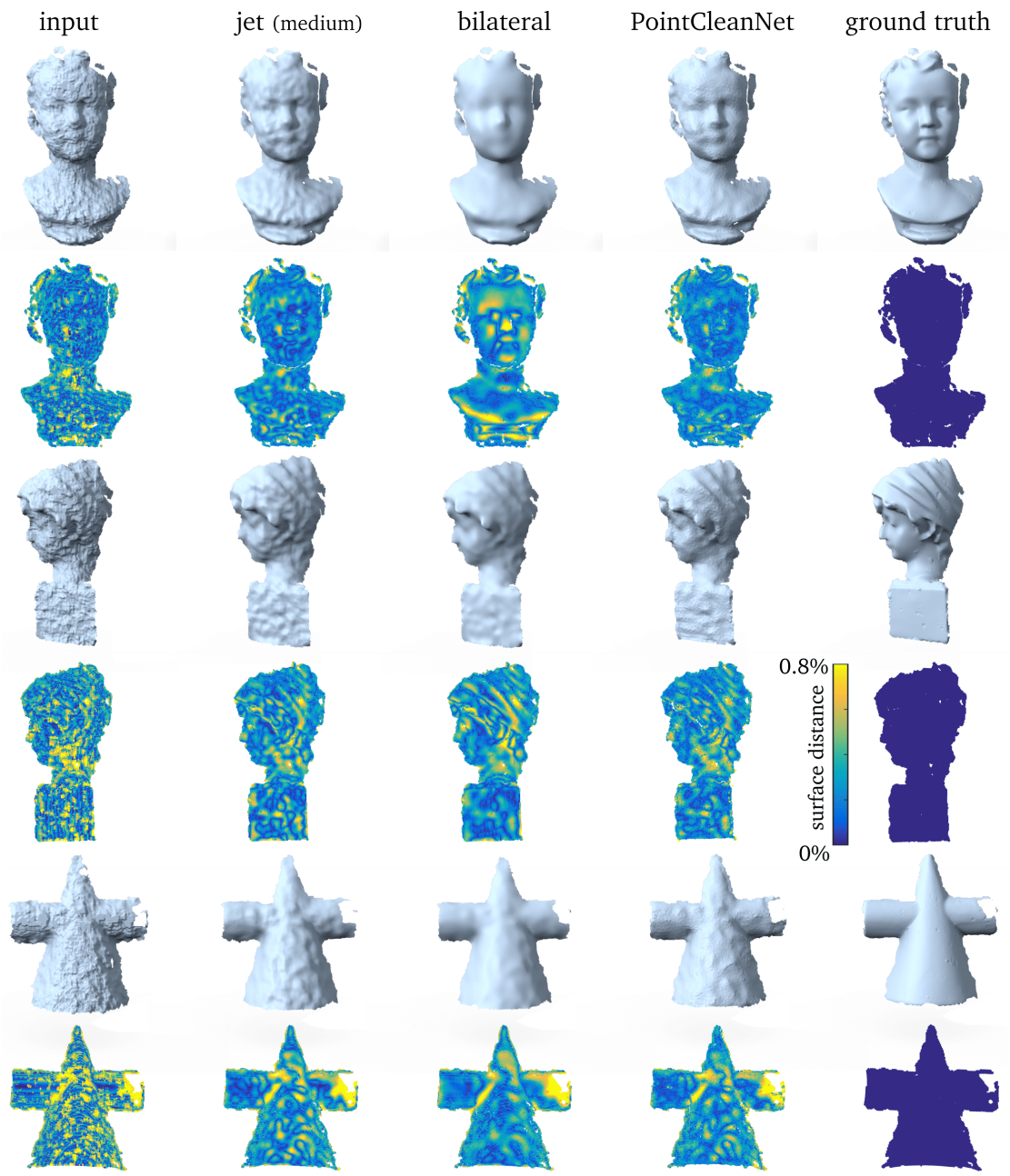

Figure 3.15 – Qualitative comparison of the three best performing methods on the *Kinect v1* dataset. For each shape, we compare a Poisson reconstruction and the normalized distance to the ground truth surface of the denoised point sets computed by each method.

3.7 Conclusion, Limitations and Future Work

We presented PointCleanNet, a learning-based framework that consumes noisy point clouds and outputs clean ones by removing outliers and denoising the remaining points with a displacement back to the underlying (unknown) scanned surface. One key advantage of the proposed setup is the simplicity of using the framework at test time as it neither requires additional parameters nor noise/device specifications from the user. In our extensive evaluation, we demonstrated that PointCleanNet consistently outperforms state-of-the-art denoising approaches (that were provided with manually tuned parameters) on a range of models under various noise settings. Given its universality and ease of use, PointCleanNet can be readily integrated with any geometry processing workflow that consumes raw point clouds. Note that in our current framework, we still need paired noisy-clean data to train PointCleanNet. An exciting future direction would be learn denoising directly from unpaired data. As a supervised learning method, our approach is also unlikely to succeed when noise characteristics during training are very different from the ones of the test data.

While we presented a first learning architecture to clean raw point clouds directly, several future directions remain to be explored: (i) First, as a simple extension, we would like to combine the outlier removal and denoising into a single network, rather than two separate parts. (ii) Further, to increase efficiency, we would like to investigate how to perform denoising at a patch-level rather than per-point level. This would require designing a scheme to combine denoising results from overlapping patches. (iii) Although PointCleanNet already produces a uniform point distribution on the underlying surface if the noisy points are uniformly distributed, we would like to investigate the effect of a specific uniformity term in the loss function (similar to [230]) to also produce a uniform distribution for non-uniform noisy points. The challenge, however, would be to restrain the points to remain on the surface and not deviate off the underlying surface. (iv) Additionally, it would be interesting to investigate how to allow the network to upsample points, especially in regions with insufficient number of points, or to combine it with existing upsampling methods such as [232] This would be akin to the 'point spray' function in more traditional point cloud processing toolboxes. (v) Finally, we would like to investigate how to train a point cloud cleanup network *without* requiring paired noisy-clean point clouds in the training set. If successful, this will enable directly handling noisy point clouds from arbitrary scanning setups without requiring explicit noise model or examples of denoised point clouds at training time. We plan to draw inspiration from related *unpaired* image-translation tasks where generative adversarial setups that have been successfully used.

Chapter 4

Intrinsic Point Cloud Interpolation via Dual Latent Space Navigation

We present a learning-based method for interpolating and manipulating 3D shapes represented as point clouds, that is explicitly designed to preserve intrinsic shape properties. Our approach is based on constructing a dual encoding space that enables shape synthesis and, at the same time, provides links to the intrinsic shape information, which is typically not available on point cloud data. Our method works in a single pass and avoids expensive optimization, employed by existing techniques. Furthermore, the strong regularization provided by our dual latent space approach also helps to improve shape recovery in challenging settings from noisy point clouds across different datasets. Extensive experiments show that our method results in more realistic and smoother interpolations compared to baselines. Both the code and our pre-trained network can be found online: https://github.com/mrakotosaon/intrinsic_interpolations.

4.1 Introduction

A core problem in 3D computer vision is to manipulate and analyze shapes represented as point clouds. Compared to other representations such as triangle meshes or dense voxel grids, point clouds are distinguished by their generality, simplicity and flexibility. For these reasons, and especially with the introduction of PointNet and its variants [183, 184, 196], point clouds have gained popularity in machine learning applications, including point-based *generative models*.

Unfortunately the flexibility of the point cloud representation also comes at a cost, as it does not encode any *topological* or intrinsic metric information of the underlying object. Thus, methods trained on point cloud data can, by their nature, be insensitive to distortion that might appear on generated shapes. This problem is particularly prominent in 3D shape interpolation, where a common approach is to generate intermediate shapes by interpolating the learned latent vectors. In this case, even if the end-shapes are realistic, the intermediate ones can have severe distortions that are very difficult to detect and correct using only point-based information. More generally, several works have observed that point cloud-based generative models can fail to capture the space of natural shapes [150, 125], making it difficult to navigate them while maintaining realism.

In this paper, we introduce a novel architecture aimed specifically at injecting intrinsic information into a generative point-based network. Our method works by learning consistent mappings across the latent space obtained by a point cloud auto-encoder and a feature encoding that captures the *intrinsic* shape structure. We show that these two

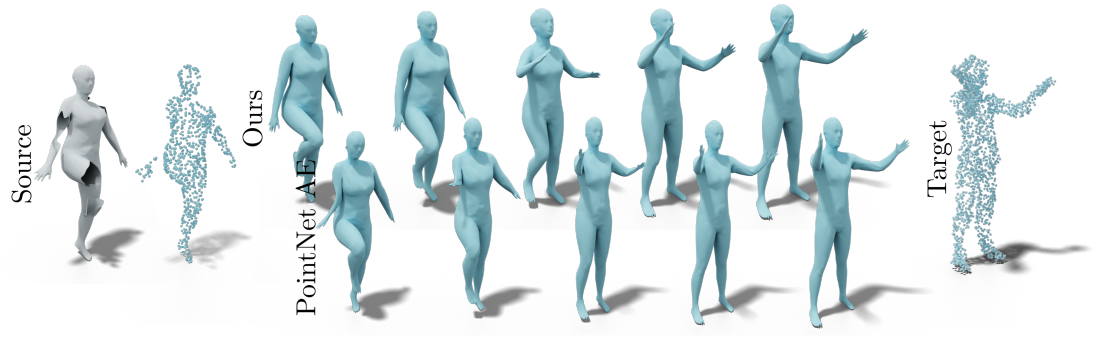

Figure 4.1 – Intrinsic point cloud interpolation between points from an incomplete scan with holes (left, reconstructed in first blue column) and points from a noisy mesh (right, reconstructed in last blue column). Our method both reconstructs the shape better and produces a more natural interpolation than a PointNet-based auto-encoder.

components can be optimized using shapes represented as triangle meshes during training. The resulting linked latent space combines the strengths of a generative latent model and of intrinsic surface information. Finally, we use the learned networks at test time on raw 3D point clouds that are neither in correspondence with the training shapes, nor contain any connectivity information.

Our approach is general and not only enables smooth interpolations, while avoiding expensive iterative optimization, but also, as we show below, leads to more accurate shape reconstruction from noisy point clouds across different datasets. We demonstrate on a wide range of experiments that our approach can significantly improve upon recent baselines in terms of the accuracy of shape recovery as well as realism and smoothness of shape interpolation.

4.2 Related Work

Shape interpolation, also known as morphing in certain contexts, is a vast and well-researched area of computer vision and computer graphics (see [144] for a survey of the early approaches). Below we review only most relevant works and focus on structure-preserving mesh interpolation, and on recent learning-based methods that operate on point clouds.

Classical methods for 3D shape interpolation have primarily focused on designing well-founded geometric metrics, and associated optimization methods that enable smooth structure-preserving interpolations. Early works in this direction include variants of as-rigid-as-possible interpolation and modeling [5, 127, 223] and various *representations* of shape deformation that facilitate specific transformation types, e.g. [209, 124, 152, 55, 208] among many others.

A somewhat more principled framework is provided by the notion of *shape spaces* [135, 167] in which interpolation can be phrased as computing a shortest path (geodesic).

In the case of surface meshes, this approach was studied in detail in [138] and then extended in numerous follow-up works, including [220, 87, 117, 115, 116] among others. These approaches enjoy a rich theoretical foundation, but are typically restricted to shapes having a fixed connectivity and can lead to difficult optimization problems at test time.

We also note a recent set of methods based on the formalism of *optimal transport* [20, 198, 32] which have also been used for shape interpolation. These approaches treat the input shapes as probability measures that are interpolated via efficient optimization techniques.

Somewhat more closely related to ours are data-driven and feature-based interpolation methods. These include interpolation based on hand-crafted features [91, 125] or on exploring various *local* shape spaces obtained by analyzing a shape collection [92, 238, 210]. Such techniques work well if the input shapes are sufficiently similar, but require triangle meshes and dense point-wise correspondences, or a single template that is fitted to all input data to build a statistical model, e.g. [113, 26, 27].

Most closely related to ours are recent generative models that operate directly on unorganized point clouds [2, 150, 155]. These methods are often inspired by the seminal work of PointNet and its variants [183, 184] and are typically based on autoencoder architectures that allow shape exploration by manipulation in the latent space. Despite significant progress in this area, however, the structure of learned latent spaces is typically not easy to control or analyze. For example, it is well-known (see e.g. [125]) that commonly-used *linear interpolation* in latent space can give rise to unrealistic shapes that are difficult to detect and rectify.

Common approaches to address these issues include extensive data augmentation [104], adversarial losses that penalize unrealistic instances [150, 19] or explicit modeling of the metric in the latent space. The latter can be done by computing the Jacobian of the decoder from the latent to the embedding space [48, 192] or using feature-based metrics at test time [143, 88]. Unfortunately, as we show below, such techniques either lead to difficult optimization problems at test time, or can still result in significant shape distortion.

Contribution In this paper, we propose to address the challenges mentioned above by building a *dual latent space* that combines a learned point-based auto-encoder with another parallel encoding that captures the intrinsic shape metric given by the lengths of edges of triangle meshes, required only during training. This second encoding exploits the insights of mesh-based interpolation techniques [138, 116, 191] that highlight the importance of *interpolating the intrinsic surface information* rather than the point coordinates. We combine these two encodings by constructing dense networks that "translate" between the two latent spaces, and enable smooth and accurate interpolation without relying on correspondences or solving expensive optimization problems at test time.

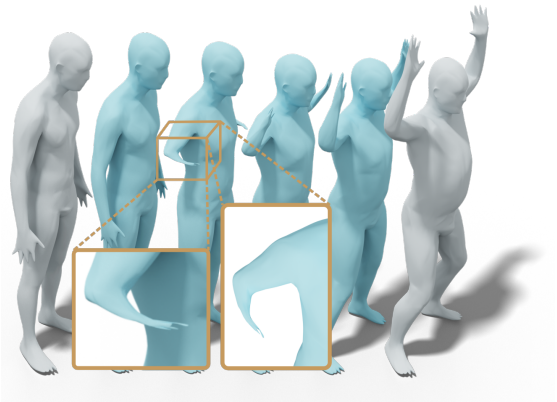

Figure 4.2 – Linear interpolation in the latent space of the shape AE produces artefacts, as the interpolation is close to linear interpolation of the coordinates.

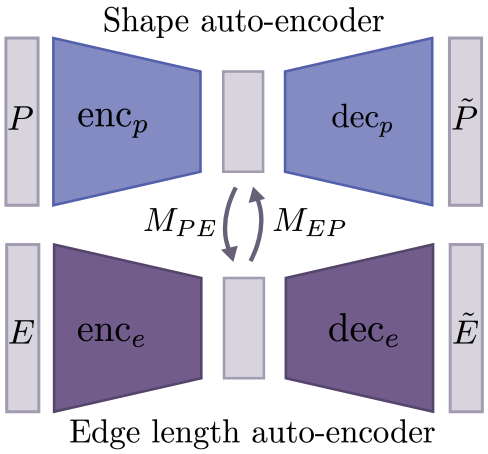

Figure 4.3 – Our overall architecture. We build two auto-encoders that capture the shape and edge length structure respectively, as well as two mapping networks M_{PE} and M_{EP} that "translate" across the two latent spaces.

4.3 Method

4.3.1 Overview

Figure B.8 gives an overview of our network. As mentioned above, it consists of three main building blocks and training steps: a shape auto-encoder, an auto-encoder of the edge lengths of the underlying mesh, and two "translation" networks that enable communication between the two latent spaces. These networks are used at test time to endow given point clouds with intrinsic information which is then used, in particular, for more accurate point cloud interpolation. We assume that the training data is given in the form of triangle meshes with fixed connectivity, while the input at test time consists of unorganized point clouds. In the following section we describe our architecture and the associated losses, while the implementation and experimental details are given in Section 6.4.

4.3.2 Architecture

Shape auto-encoder. Our first building block (Fig. B.8 top) consists of a shape auto-encoder, based on the PointNet architecture [183]. We denote the encoder and decoder networks as enc$_p$ and dec$_p$ respectively (we provide the exact implementation details and compare to a VAE in the appendix Section A.3) and Section A.6. To train this network we use the basic L_2 reconstruction loss, since we assume that the input shapes are in 1-1 correspondence. This leads to the following training loss:

$$L_{rec}(P) = \frac{1}{n}\sum_{i=1}^{n} \|P_i - \tilde{P}_i\|^2, \text{ where } \tilde{P} = \text{dec}_p\left(\text{enc}_p(P)\right). \tag{4.1}$$

Here P is a training shape, the summation is done over all points in the point cloud, and P_i represents the 3D coordinates of point i.

Importantly, our point-based encoder enc_p inherits the permutation invariance of PointNet [183], which is crucial in real applications. Specifically, this allows us to encode arbitrary point clouds at test time even if they have significantly different sampling and are not in correspondence with the training data.

Edge length auto-encoder As observed in previous works and as we confirm below, the shape AE can capture the structure of individual shapes, but often fails to reflect the overall structure of *shape space*, which is particularly evident during shape interpolation. We address this by constructing a separate auto-encoder whose latent space captures the *intrinsic* shape information, and by learning mappings across the two latent spaces.

For this, we first build an auto-encoder (enc_e, dec_e) with dense layers that aims to reconstruct a list of edge lengths. Note that since we assume 1-1 correspondence at training time, the list of lengths of edges can be given in canonical (e.g., lexicographic with respect to vertex ids) order. We therefore build an auto-encoder that encodes this list into a compact vector and decodes it back from the latent representation. Our training loss for this part consists of two components: an L_2 error on the predicted edge lengths and an additional term that promotes linearity in the learned latent space:

$$L_e(E_A) = \|\text{dec}_e(\text{enc}_e(E_A)) - E_A\|^2 \tag{4.2}$$

$$\begin{aligned} L_{lin}(E_A, E_B) = \Bigg\| &\frac{\text{dec}_e(\text{enc}_e(E_A)) + \text{dec}_e(\text{enc}_e(E_B))}{2} \\ &-\text{dec}_e\left(\frac{\text{enc}_e(E_A) + \text{enc}_e(E_B)}{2}\right)\Bigg\|^2. \end{aligned} \tag{4.3}$$

Here E_A, E_B are the lists of edge lengths corresponding to the triangle meshes A, B given during training. Our motivation for the loss L_{lin} is to explicitly encourage linear structure, which promotes smoothness of interpolated edge lengths and thus, as we show below, minimizes intrinsic distortion.

Mapping networks Given two pretrained auto-encoders described above, we train two dense mapping networks that translate elements between the two latent spaces. We use M_{PE} and M_{EP} to denote the networks that translate an element from the shape (resp. edge) latent space to the edge (resp. shape) latent space.

To define the losses we use to train these two networks, for a training mesh A we let $l_A = \text{enc}_p(A)$ denote the latent vector associated with A by the shape encoder. Recall that when training the shape AE we compare A with $\text{dec}_p(l_A)$. To train our mapping networks M_{PE} and M_{EP} we instead compare A with $\text{dec}_p\left(M_{EP}(M_{PE}(l_A))\right)$. In

other words, rather than decoding directly from l_A we first map it to the edge length latent space (via M_{PE}). We then map the result back to the shape latent space (via M_{EP}) and finally decode the 3D shape. We denote the shape reconstructed this way by $\tilde{A} = \text{dec}_p(M_{EP}(M_{PE}(\text{enc}_p(A))))$. We compare $\tilde{A}$ to the original shape A, which leads to the following loss:

$$L_{map1}(A) = d^{\text{rot}}(\tilde{A}, A). \tag{4.4}$$

Here d^{rot} is a *rotation invariant* shape distance comparing the original and reconstructed shape. We use it since the list of edge lengths can only encode a shape up to rigid motion [99]. Specifically, we first compute the optimal rigid transformation between the input shape A and the predicted point cloud $\tilde{A}$ using Kabsh algorithm [10]. We then compute the mean square error between the coordinates after alignment. As shown in [125] this loss is differentiable using the derivative of the Singular Value Decomposition.

Our second loss compares the edge lengths of the reconstructed shape $\tilde{A}$ to the edge lengths of A. For this we use the standard L_2 norm squared:

$$L_{map2}(A) = \|E_A - E_{\tilde{A}}\|_2^2, \tag{4.5}$$

where E_A denotes the list of edge lengths of shape A.

Our last loss considers a similar difference but starting in the edge length latent space, rather than the shape one. Specifically, given a shape A with the list of edge lengths E_A, we first encode it to the edge length latent space via $\text{enc}_e(E_A)$. We then translate the resulting latent vector to the shape latent space (via M_{EP}) and back to the edge length latent space (via M_{PE}), and finally decode the result using dec_e. This leads to the following loss:

$$L_{map3}(A) = \|\text{dec}_e(M_{PE}(M_{EP}(\text{enc}_e(E_A)))) - E_A\|_2^2, \tag{4.6}$$

Our overall loss is then simply a weighted sum of three terms $\alpha L_{map1} + \beta L_{map2} + \gamma L_{map3}$ for shapes given at training where γ is non-zero. We evaluate other possible losses in the appendix.

Network Training To summarize, we train our overall network architecture described in Figure B.8 in three separate steps. First we train the shape-based auto-encoder using the loss given in Eq. (A.2). Then we train the edge length auto-encoder using the sum of the losses in Eq. (A.5) and Eq. (4.3). Finally we train the dense networks M_{EP} and M_{PE} using the sum of the three losses in Eq. (4.4), Eq. (4.5), Eq. (4.6). We also experimented with training the different components jointly but have observed that the problem is both more difficult and the relative properties of the computed latent spaces become less pronounced when trained together, leading to less realistic reconstructions, please refer to Sec. A.3 for additional details.

4.3.3 Navigating the restricted latent space

After training the networks as described above, we use them at test time for shape reconstruction and interpolation. We stress that at test time we do not use the edge encoder

and decoder networks enc_e, dec_e, as they require canonical edge ordering. Instead we use the permutation invariant shape auto-encoder and the mapping networks M_{PE}, M_{EP} to better preserve intrinsic shape properties.

Interpolation Given two possibly noisy unorganized point clouds P_A and P_B we first compute their associated edge-based latent codes: $m_A = M_{PE}(\text{enc}_p(P_A))$ and $m_B = M_{PE}(\text{enc}_p(P_B))$. Here we use the permutation-invariance of our encoder enc_p allowing to encode unordered point sets. We then linearly interpolate between m_A and m_B but use the *shape decoder* dec_p for reconstruction. Thus, we compute a family of intermediate point clouds as follows:

$$P_\alpha = \text{dec}_p\left(M_{EP}\left((1-\alpha)m_A + \alpha m_B\right)\right), \ \alpha \in [0 \dots 1] \tag{4.7}$$

In other words, we interpolate the latent codes in the edge-based latent space, but perform the reconstruction via the shape decoder dec_p. This allows us to make sure that the reconstructed shapes are both realistic and their intrinsic metric is interpolated smoothly. Note that unlike the purely geometric methods, such as [138], our approach does not rely on the given mesh structure at test time. Instead, we employ the learned edge-based latent space as a proxy for recovering the intrinsic shape structure, which as we show below, is sufficient to obtain accurate and smooth interpolations.

Since the edge length auto-encoder is fully rotation invariant, it is necessary to align the output shapes at test time. We can do so easily by using the same optimal rigid transformation as used to compute Eq. (4.6).

Reconstruction Given a point cloud P_A we also use our trained architecture for shape recovery via $S = \text{dec}_p(M_{EP}(M_{PE}(\text{enc}_p(P_A))))$. Here we use the fact that the edge-length latent space helps to regularize the shape space avoiding noisy or distorted output.

4.3.4 Interpretation

Our approach can be interpreted both in terms of capturing the structure of individual 3D shapes and of the entire *shape space*. For the former, our shape and edge-length auto-encoders help to capture, respectively, the *extrinsic* and *intrinsic* information of the underlying surface. Jointly, they enable more accurate shape recovery and comparison. In this context, our approach is related to methods for reconstructing a shape from its intrinsic metric. This problem, while possible theoretically [99], is computationally challenging and error prone in practice [215, 33, 54, 52]. By using a *learned* latent space our reconstruction is both efficient and leads to realistic results.

In terms of the shape space, the latent vectors of the shape auto-encoder provide a way to parametrize the space of realistic 3D shapes while the edge-length latent space helps to impose a *distance structure* on that space. This is similar to the standard approach in Riemannian geometry [39] where the manifold structure of a space and the *metric* on it are encoded separately. We highlight this interpretation in the appendix Section A.5, and leave its complete exploration as exciting future work.

4.3.5 Unsupervised training

Our method can be adapted to the unsupervised context where the 1-1 correspondences are not provided during training. Contrary to our main pipeline, we cannot compute the edge lengths directly from the training data. However, we can encourage the model to produce a consistent mesh as described in [104]. We initialize the weights by pre-training on a selected mesh using the reconstruction loss L_{rec} described in (A.2) and train the model using Chamfer distance and regularization losses to keep the triangulation consistent. Finally, we can train the edge-length auto-encoder by using the output of the shape auto-encoder as training data. We describe this process in detail in the appendix Section A.4.

4.4 Results

Datasets We train our networks on two different datasets: humans and animals. For humans, we use the dataset proposed in [125]. The dataset contains 17440 shapes subsampled to 1k points from DFAUST [27] and SURREAL [207]. The test set contains 10 sub-collections (character + action sequence, each consisting of 80 shapes) that are isolated from the training set of DFAUST and 2000 shapes from SURREAL dataset. During training the area of each shape is normalized to a common value. For animals we sample 12000 shapes from the SMAL dataset [241]. We sample an equal number of shapes from the 5 categories (big cats, horses, cows, hippos, dogs) to build a training set of 10000 shapes and a testset of 2000 shapes. We simplify the shapes from SMAL to 2002 points per mesh. The animal dataset provides challenging shape pairs that are far from being isometric.

4.4.1 Shape interpolation

We evaluate our method on our core application of shape interpolation and compare against six different recent baselines. Namely, we compare to three data-driven methods, by performing linear interpolations in the latent spaces of auto-encoders using PointNet [183] and PointNet++ [184] architectures as well as the pre-trained auto-encoder proposed in the state-of-the-art non-rigid shape matching method 3D-CODED [104].

We also compare to three optimization-based geometric methods, by building on the ideas from [138, 192, 48]. We produce our first two baselines by initializing a linear path in latent space of our shape auto-encoder and optimizing each sample via 1000 steps of gradient descent. We use GD EL to denote the method that minimizes edge length variations over interpolation steps, and G2 L2 to denote the method that minimizes the L^2 variance over the interpolated shape coordinates as described in [192]. Finally we compare to a method simplified from [138] (GD Coord.), in which we first initialize a path by linearly interpolating the coordinates of source and target shapes. Similarly to GD EL, we minimize the discrete interpolation energy (edge length variation) using gradient descent on the point coordinates directly.

Remark that GD Coord., GD L2 and GD EL methods all rely on gradient descent to

	Direct inference				Optimization based		
	Ours	PN	3D-Coded	PN++	GD L2	GD EL	GD Coord.
EL	**0.231**	0.351	0.613	0.299	0.363	0.298	**0.034**
Area	**1.261**	1.773	3.137	1.586	1.838	1.714	**0.248**
Volume	**0.342**	1.613	1.243	335.2	1.483	1.703	**0.152**

Table 4.1 – We report the mean squared variance of the edge length (EL), per surface area and total shape volume over the interpolations of 100 shape pairs. Our method achieves lowest variance across all intrinsic features among direct inference methods. Note that GD coord. leads to interpolation with low distortion, as it optimizes the coordinates directly but produces unrealistic shapes (see Figure 4.4). We refer to PointNet as PN and PointNet++ as PN++.

compute each interpolation *at test time*. In other words, these approaches all require to solve a highly non-trivial optimization problem during interpolation, leading to additional computational cost and parameters (learning rate, number of iterations). In contrast, our method outputs a smooth interpolation in a single pass.

To evaluate the interpolations we sample 50 shapes from the DFAUST testset using farthest point sampling. We then test on 100 random pairs from those 50 shapes. We use our pipeline trained with $\alpha = 30$, $\beta = 1200$ and $\gamma = 800$ in the mapping networks loss described in Section 4.3.2. We provide an ablation study on the choice of losses in the appendix Section A.3.

Table 4.1 shows quantitative comparisons. Given an interpolation path (S_n) obtained by each method, we compute the mean squared variance of various shape features f on the path. We consider three features: lengths of edges, area of faces and overall volume enclosed by the shape (computed from the mesh embedding). For each of these, we compute the sum of the squared differences across all instances in the interpolating sequence:

$$Var_f(S_n) = \frac{1}{n-1} \sum_{i=2}^{n} \|f(S_i) - f(S_{i-1})\|^2. \tag{4.8}$$

Intuitively, we expect a good interpolation method to result in smooth interpolations which would have low variance across all of the intrinsic shape properties. When comparing with PointNet++ as it inputs normalized bounding boxes, we normalize the total area of each output. The large volume variance of this baseline is primarily due to bad reconstruction quality of the input shapes.

As shown in Table 4.1 our method produces the best results among the direct data-driven methods and the best results over all the baselines except from GD Coord. This latter method is not data-driven and optimizes edge lengths directly on the coordinates without any constraints. As such, it produces shapes with low distortion but that are not realistic (see Figure 4.4). Furthermore, similarly to [138] it requires the input shapes to be represented as meshes in 1-1 correspondence.

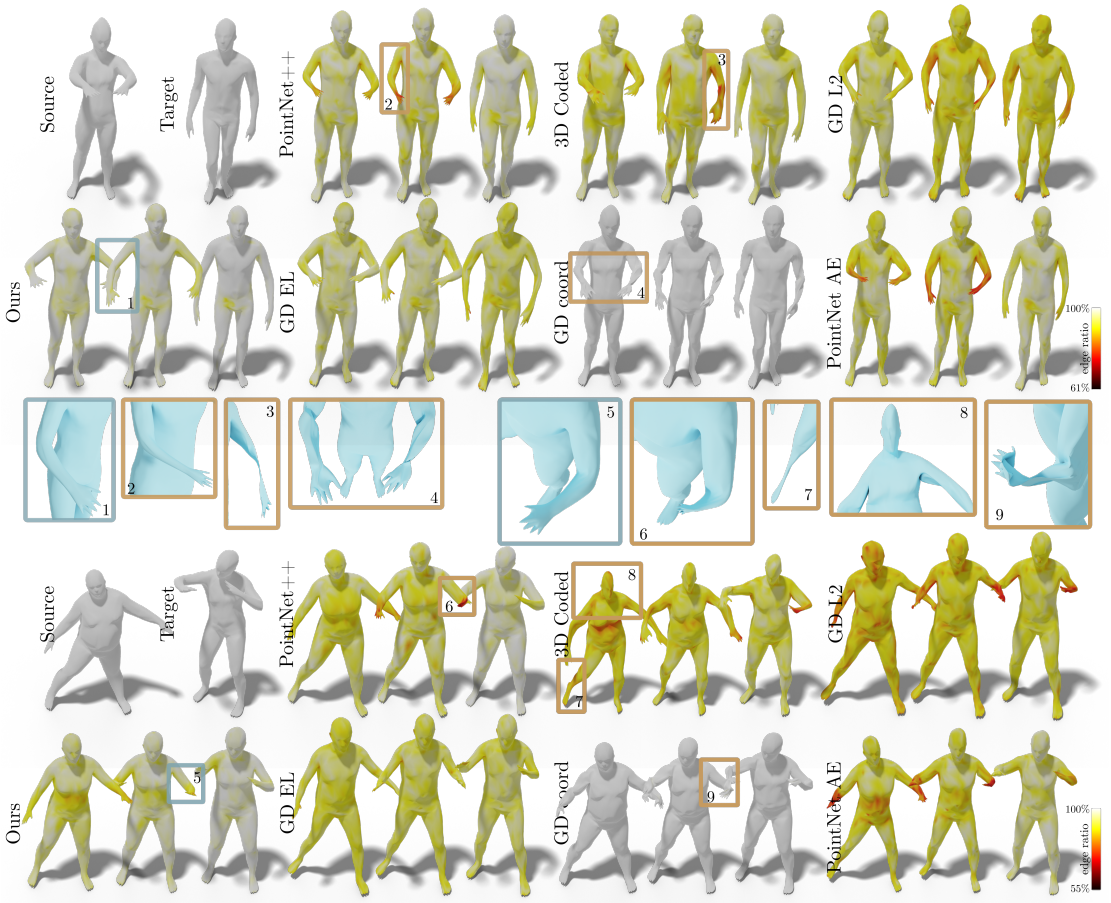

Figure 4.4 – Qualitative comparison of interpolation on DFAUST testset. We display the edge ratio between the linear interpolation of the target and source edges and the produced interpolation.

In all qualitative figures, we visualize the minimum ratio between the linear interpolation of the ground truth edge lengths and the edge lengths of the produced shapes. We color-code this ratio to highlight areas of highest intrinsic distortion (shown in red). In Figure 4.4 we illustrate the interpolated shapes between the input source and target, shown in grey. We observe that PointNet AE and PointNet++ methods tend to produce results that are closer to linear interpolation of the coordinates. As highlighted above, we notice that while GD Coord. has low variance in the interpolated intrinsic features, the reconstructed shapes do not look natural. Overall, our method presents less distortions and more smooth interpolations compared to all baselines. We present more comparisons and evaluations in in the appendix Section A.1.

We further evaluate our model on the SMAL dataset. To build the interpolation pairs from the test set, we sample 10 shapes per category by farthest points sampling. We then choose 100 random pairs from that dataset. In Figure 4.5 we show results of

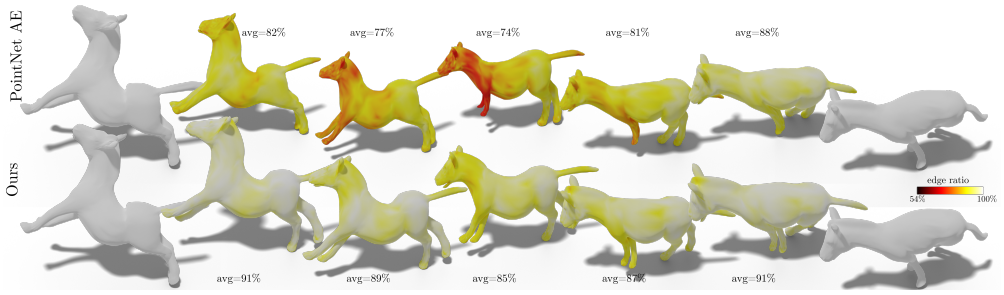

Figure 4.5 – Interpolation of two horses from the SMAL dataset.

	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
PointNet AE	3.023	**2.120**	2.454
Edge Length AE	3.127	-	-
Ours	1.641	2.572	1.562

Table 4.2 – Mean squared reconstruction losses on the humans testset. Edge length reconstruction loss (EL), Point cloud coordinates reconstruction loss (PC) and per triangle area difference.

	CD (10^{-3})	volume (10^{-5})	area
Shape AE	4.703	30.851	0.1382
Ours	4.135	9.47	0.047

Table 4.3 – Reconstruction accuracy on the SCAPE dataset. Chamfer distance (CD), mean squared total volume difference and total area difference.

interpolating between two horses. We observe that linear interpolation in the shape latent space leads to shape distortions such as shorter legs (middle) and wrong shape size estimation (top left). The Shape AE (resp. Ours) produces a edge variance of 2.068 (resp. 1.548). Similarly to above, our method shows improvement at interpolating intrinsic information. We provide detailed numerical evaluation of interpolations on SMAL in the appendix Section A.1.

Interpolation in the unsupervised case. The unsupervised Shape AE (resp. Ours) produces a edge variance of 0.599 (resp. 0.394). While we observe better results in the supervised setting, our method nevertheless produces quantitative and qualitative improvement over the linear interpolation in latent space. We provide further numerical and qualitative results in the appendix Section A.4.

4.4.2 Shape reconstruction

We also evaluate the accuracy of our model for shape reconstruction on the DFAUST/SURREAL testset. In Table A.2, we compare the reconstruction accuracy to the base models. We measure intrinsic features: edge length and per triangle area reconstruction loss, and extrinsic L^2 coordinates reconstruction loss. Our method reconstructs the input shape intrinsic features better that the PointNet AE while producing comparable extrinsic

reconstruction loss.

We further evaluate the generalization capacity of our network by evaluating on the SCAPE [9] dataset. For testing we sample 1000 random points from the surface of each mesh. Table 4.3 shows an improvement in the reconstruction for our method. We observe even higher relative performance when comparing the total volume and total area of the reconstructed shapes which give a sense of the perceived quality of the shapes. Shape distortions are often related to shrunk or disproportional body parts. We show qualitative results on reconstruction in the appendix Section A.2. Overall, our method produces more precise and natural reconstructions. Finally, as shown in Figure 6.1, our method is robust to high levels of noise (left), holes, and missing parts (right). We provide further reconstruction examples in the appendix Section A.2.

4.5 Conclusion, Limitations & Future Work

We presented a method for interpolating unorganized point clouds. Key to our approach is a dual latent space that both captures the extrinsic and intrinsic shape information, given by edge lengths provided during training. We demonstrate that our approach leads to significant improvement over existing methods, both in terms of interpolation smoothness and quality of the generated results. In the future, we plan to extend our method to incorporate other features such as semantic classes or segmentations. It would also be interesting to explore our dual encoding space in other applications on images or graphs.

Chapter 5

Learning Delaunay Surface Elements for Mesh Reconstruction

We present a method for reconstructing triangle meshes from point clouds. Existing learning-based methods for mesh reconstruction mostly generate triangles individually, making it hard to create manifold meshes. We leverage the properties of 2D Delaunay triangulations to construct a mesh from manifold surface elements. Our method first estimates local geodesic neighborhoods around each point. We then perform a 2D projection of these neighborhoods using a learned logarithmic map. A Delaunay triangulation in this 2D domain is guaranteed to produce a manifold patch, which we call a Delaunay surface element. We synchronize the local 2D projections of neighboring elements to maximize the manifoldness of the reconstructed mesh. Our results show that we achieve better overall manifoldness of our reconstructed meshes than current methods to reconstruct meshes with arbitrary topology. Our code, data and pretrained models can be found online: https://github.com/mrakotosaon/dse-meshing

5.1 Introduction

Surface reconstruction from a given set of points (e.g., a scan), has a long history in computational geometry and computer vision [23, 172]. A version of the problem requires triangulating a given point cloud to produce a watertight and manifold surface. A key challenge is to handle different sampling conditions while producing well-shaped triangles and preserving the underlying shape features.

A good surface reconstruction algorithm should satisfy the following requirements: (i) produce a connected, manifold and watertight triangulation; (ii) require no case-specific parameter tuning; (iii) preserve sharp features; (iv) handle point sets with non-uniform distribution; and (v) generalize to handle a variety of shapes.

A widely-used pipeline for surface reconstruction consists in first computing an implicit surface representation [133] and then extracting a triangulation using a volumetric method such as Marching Cubes [156]. Methods in this category often require additional information (e.g., oriented normals), while, crucially, the resulting triangulations may not preserve the original point set and can oversmooth sharp features. On the other hand, methods from computational geometry, e.g., alpha shapes [79], ball pivoting [24], etc., can respect the original point set, come with theoretical guarantees and produce triangulations with desirable properties (e.g., good angle distribution). These approaches, however, typically require careful parameter selection and rely on dense, uniformly sampled point sets.

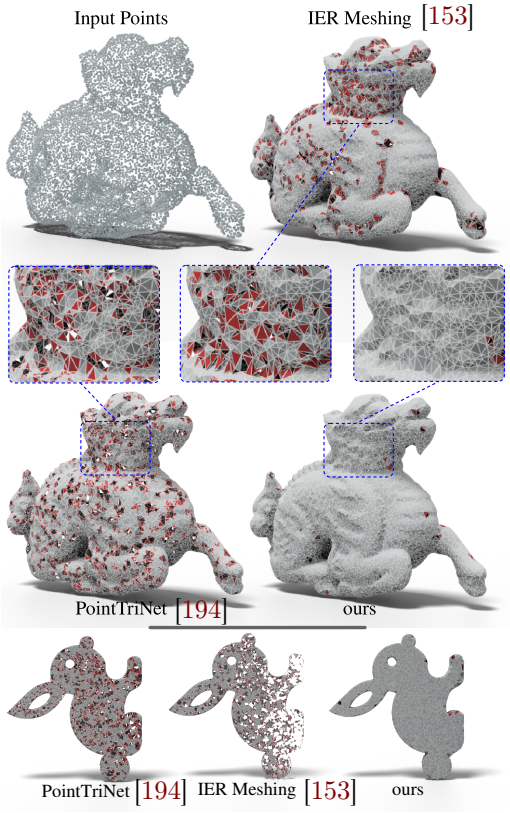

Figure 5.1 – We present a method for mesh reconstruction from point clouds. We combine Delaunay triangulations with learned local parameterizations to obtain a higher-quality mesh than the current state-of-the-art. Bad (non-manifold) triangles are shown in red. Our method is robust to uniformly (top) and non-uniformly (bottom) sampled points.

More recently, learning-based approaches have been developed to extract a triangulation without case-specific parameter selection. Most of such techniques focus on robustly predicting a signed distance field or simply an occupancy map, from which a mesh is subsequently extracted using volumetric triangulation [49, 94, 179]. Only two recent methods [194, 153] produce a triangulation while respecting the original point set, but they ignore the quality of the triangles or have trouble reconstructing sharp features.

We present a method that combines the advantages of classical methods with learning-based data priors. Our method is based on blending together *Delaunay surface elements*, which are defined by a 2D Delaunay triangulation of a local neighborhood in the point set after projecting it to a planar 2D domain. For this, we propose an approach that predicts a local projection via learned logarithmic maps and uses them to propose likely triangles using local Delaunay triangulations. Figure 6.1 shows an example reconstructions using our method. We evaluate our method on a benchmark of diverse surface point sets, and provide a comparison with both classical and learning-based methods to show the advantages of the proposed approach. Through these extensive experiments, we

demonstrate that our method generalizes across diverse test sets, is more robust than classical approaches, and produces higher-quality triangulations than recent learning-based methods.

Computing a triangulation of a given point set is one of the most fundamental problems in computational geometry, computer vision, and related disciplines. We review methods most closely related to ours and refer to recent surveys [137, 197, 23, 172] for a more in-depth discussion.

A commonly-used pipeline for surface reconstruction [120, 56] consists of computing the implicit surface representation using, e.g., a signed distance function. A mesh can then be extracted with standard methods such as Poisson surface reconstruction [133] combined with Marching Cubes [156] or Dual Contouring [132]. Such approaches work well in the presence of oriented normals and dense/uniform point sets, but do not necessarily preserve the given points in the final mesh and lead to over-smoothing or loss of details (see [23] for a detailed discussion).

We were inspired by classical methods based on Delaunay triangulations [28, 142, 30, 102, 68], alpha shapes [79] or ball pivoting [24]. Such approaches can be shown to recover the shape mesh topology [8] under certain sampling conditions (an excellent overview of such approaches is provided in [67]). Unlike implicit-based methods, approaches in this category, e.g., [24, 7, 31] typically preserve the input point set. However, they can often fail to produce satisfactory results for coarsely sampled shapes or in the presence of complex geometric features. Another more robust, but computationally more expensive, approach capable of feature preservation was introduced in [72], based on iterative optimisation using optimal transport.

5.1.1 Learning for surface reconstruction

To address the challenges mentioned above, recent methods have aimed to learn surface reconstruction priors from data. The majority of existing learning-based methods in this area use a volumetric shape representation. For example, meshes can be computed by predicting voxel grid occupancy [97, 166] or via a differentiable variant of the marching cubes [151], or more recently using generative models for explicit or implicit surface prediction [49, 94, 179, 170]. While these methods can produce accurate results they solve a different problem to ours and do not compute a mesh over the given point set. Instead, we focus on directly meshing a set of input points, which provides better control over the final shape and avoid over-smoothing, often associated with implicit surface-based techniques.

Other methods have also aimed to compute a surface by deforming a simple template while updating its connectivity [211, 178], fitting parameterized [105, 219] or mesh-aware patches [13], performing local (e.g., convex) shape decomposition. Majority of these schemes are restricted to particular shape topology or category and again do not necessarily guarantee point set preservation.

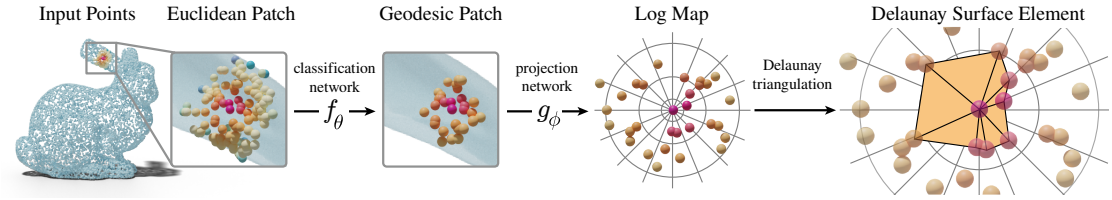

Figure 5.2 – Overview of Delaunay Surface Element (DSE) generation. For any point p_i in an input point cloud, we select the k-nearest neighbors and extract the subset of points that are in the geodesic neighborhood of p_i, using a learned classification network. A projection network then estimates a log map projection of the points into a 2D embedding, where we can apply Delaunay Triangulation to get a DSE.

5.1.2 Learning mesh connectivity

More directly, our work fits within the line of recent efforts aimed explicitly at learning the mesh connectivity for a given shape geometry. An early approach, Scan2Mesh [57] developed a graph-based formulation to generate triangles in a mesh. However, the method uses a costly volumetric representation, does not aim to produce manifold meshes, and specializes on particular shape categories.

Most closely related to ours are two very recent approaches aimed directly to address the point set triangulation problem. The first method PointTriNet [194] works on point clouds and, similarly to ours, uses a local patch-based network for predicting connectivity. However, this technique processes triangles independently and only promotes watertight and manifold structure through soft penalties. The second method was presented in [153], and estimates local connectivity by predicting the ratio between geodesic and Euclidean distances. This is a powerful signal, which is then fed into a non-learning based selection procedure, which aims to finally output a coherent mesh.

In contrast to both of these approaches [194, 153], we formulate the meshing problem as learning of (local) Delaunay triangulations. Starting from the restricted Voronoi diagram based formulation proposed in [31] we use data-driven priors to directly learn local projections to create local Delaunay patches. As a result, locally our network *guarantees* the coherence of the computed mesh. As we demonstrate below, learning Delaunay surface elements, both leads to better shaped triangles (i.e., more desirable angle distribution) and improves the overall manifold and watertight nature of the computed triangle mesh.

5.2 Method

We assume to be given an point set $P \in \mathbb{R}^{N\times 3}$ sampled from a surface $\hat{S}$. Our goal is to create a mesh $M = (P', T)$ that approximates $\hat{S}$, by choosing a new triangulation T that triangulates a subset $P' \subset P$ of the input point cloud. It is easy to obtain a high-quality triangulation for any set of points that lies in *2D*, via Delaunay triangulation [61]. However, when the set of points lies in 3D, finding a triangulation is a much harder

problem. A simple solution is to locally project points to an estimated tangent plane of the surface, resulting in local 2D embeddings where we can apply a Delaunay triangulation. However, this is problematic near complex geometry, such as edges or thin structures and is sensitive to an imperfect estimation of the tangent plane. *Logarithmic maps* [75, 118], or *log maps* for short, provide a systematic solution to this problem by providing local geodesic charts of the ground truth surface that are good local parameterizations of complex geometry.

The core idea of our method is therefore to combine Delaunay triangulations and learned log maps to create small triangulated patches that we call *Delaunay Surface Elements* (DSEs). Each DSE approximates a small part of the surface and is guaranteed to have a manifold triangulation. Since neighboring log maps may disagree, especially in regions of high curvature, we align them locally with non-rigid transformations of the 2D parameterizations of each DSE. DSEs enable us to maintain the good properties of Delaunay Triangulations, like manifoldness and high-quality triangles, within a data-driven approach, that learns to extract local geodesic patches and parameterize them with a the log map, thereby increasing robustness and reconstruction accuracy. Our approach proceeds in four steps (the first two steps are illustrated in Figure 6.2):

1. For each point $p_i \in P$, a network estimates a geodesic ball, by extracting a 3D patch $P^i \in \mathbb{R}^{k\times 3}$ made up of its k-geodesically-closest points.

2. For each 3D point patch P^i, a second network approximates the log map parameterization, to get a 2D embedding of the patch, denoted $U^i \in \mathbb{R}^{k\times 2}$.

3. We improve the consistency of neighboring patches by aligning their 2D embeddings, giving us improved patch embeddings $\hat{U}^i$, which we then use to compute the Delaunay Surface Elements.

4. The Delaunay Surface Elements *vote* for candidate triangles, which are then aggregated iteratively into a mesh.

5.2.1 Constructing Local Embeddings

The first two steps in our method are aimed at creating a patch P^i around each point p_i and a local 2D embedding U_i of the points inside the patch. These two ingredients will later be used to compute a 2D Delaunay triangulation that defines a Delaunay Surface Element.

Geodesic patch construction Given the point p_i and its K nearest neighbors Q^i, we train a network to find a subset of k points from these neighbors that are *geodesically* closest to p_i on the ground truth surface. In our experiments, we set $K = 120$ and $k = 30$. More details on the choice of k and K are provided in Section B.8 of the appendix. The network is trained to model a function $c_j := f_\theta([q^i_j, d^i_j] \mid Q^i)$ that classifies each point q^i_j in Q^i as being one of the k geodesically closest points if $c_j = 1$ or not if $c_j = 0$. We concatenate the Euclidean distance d^i_j to the center point as additional

input. The network is parameterized by θ, conditioned on the point set Q^i, and models a function from 3D position to classification value. We train this network with an L2 loss $\|c_j - \sigma(\hat{c}_j)\|^2$, where $\hat{c}_j$ is the ground classification and σ is the sigmoid function. To obtain a fixed number of k points, we select the top-k points based on their predicted labels c_j^i, giving the (geodesic) patch P^i.

Log map estimation We train a second network to compute the log map coordinates of each point in P^i, denoted as $U^i \in \mathbb{R}^{k\times 2}$. The network is trained to model a function $u_j^i := g_\phi([p_j^i, d_j^i] \mid P^i)$, where ϕ denotes the network parameters and p_j are the 3D coordinates of a point in P^i. These coordinates are concatenated with the Euclidean distance d_j^i to the center point. The network outputs the log map coordinates u_j^i in U^i, consisting of the Euclidean coordinates of the log map with an origin at the center point of the patch. Like the classification network, this network is conditioned on the input point set P^i. We use the sum of two losses: a loss that penalizes the difference to the ground truth coordinates and one that penalizes only the radial component of the coordinates, i.e. the geodesic distance to the center. Since log map coordinates are defined only up to a 2D rotation around the central point, we use the Kabsh algorithm [25] to find a 2D rotation and/or reflection that optimally aligns the predicted log map and the ground truth log map before computing the loss: $\|RU^i - \hat{U}^i\|_2^2$, where $\hat{U}^i$ is the ground truth and R is the optimal rigid transformation computed by the Kabsh algorithm. Note that the Kabsh algorithm is differentiable. Our second loss measures the error in the radial component: $\sum_j (\|u_j^i\|_2 - \|\hat{u}_j^i\|_2)^2$. This loss measures how well the network can recover geodesic distances regardless of the orientation in the patch.

Network architecture When approximating log maps with a network, continuity is an important property. If the estimated mapping from 3D space to the 2D log map parameterization is not continuous, the resulting Delaunay triangulation may have flipped or intersecting triangles. We base our architecture on FoldingNet [228] that produces continuous mappings from an input to an output domain. Unlike the original implementation, however, which maps from 2D to 3D, we want to map from 3D to 2D. Our experiments have shown that this network architecture leads to more continuous results than a PointNet-based architecture. We have also found that it improves the performance of our classification network, where we also adopt an architecture based on FoldingNet. Since we train our network on individual patches, we can train on relatively small datasets, where each shape provides a large number of patches as training samples. More details on the architecture are provided in Section B.5 of the appendix.

5.2.2 Combining Delaunay Surface Elements

At this point, we have a local 2D parameterization for each patch. We could use these local parameterizations to construct a triangulation of the patch by Delaunay-triangulating it. However, each patch may be rather inconsistent with neighboring patches, in the sense that if two patches P^i, P^j share three points a, b, c, the Delaunay triangulation of U^i may produce the triangle (a, b, c) while the triangulation of U^j may not, since the

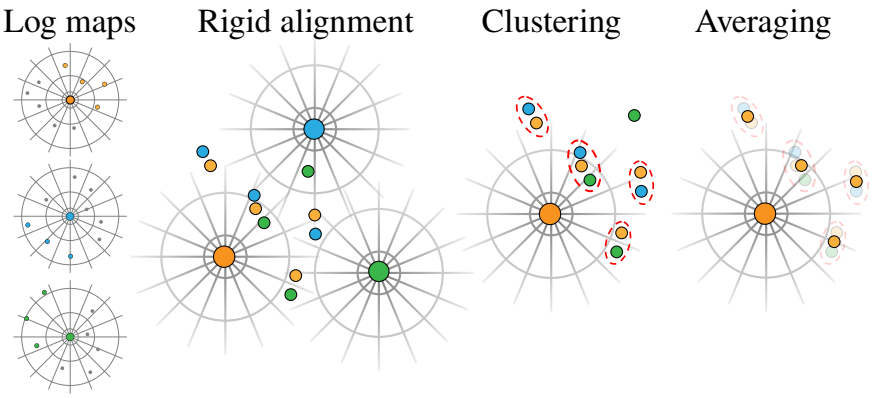

Figure 5.3 – Log map alignment. To improve the consistency of log maps, we align corresponding points in neighboring log maps with rigid transformations. The resulting sets of corresponding points are then clustered to remove outliers and averaged, giving us 2D point embeddings that are more consistent with their neighbors.

points are laid out differently in each of the two parameterization. An example is shown in Figure 5.4, right. Hence, the final pair of steps is aimed at improving the consistency between the different patch parameterizations of neighboring DSEs before combining all DSEs into the final mesh M.

Log map alignment In 2D, Delaunay triangulation are guaranteed to produce a manifold triangulation. However, we produce independent 2D parametrizations for each DSE. Large differences in the parameterization of neighboring DSEs may make their triangulations incompatible (i.e., the union of their triangles may be non-manifold). In this step, we locally align the log maps to one another to ensure better consistency, without requiring the construction of a global parameterization. Namely, a point $p_k \in P$ from the original point cloud has an image in the log maps of each patch that contains that point. We denote this set of all log map images of point p_k as R^k. We say U^i, U^j are neighbor patches if they both have a point in the same R^k. Denote the image of p_k in the log map of each of the two patches as $U^i(p_k)$, $U^j(p_k)$, respectively.

Our approach is illustrated in Figure 5.3. Considering the patch U^i, we align the neighboring patch U^j to it, by taking all corresponding points and using the Kabsch algorithm to find the rigid motion that best aligns (in the least-squares sense) the points based on their correspondences $U^i(p_k) \leftrightarrow U^j(p_k)$. Repeating this for all neighboring patches aligns them all to U^i. We then define the set R^i_k to be the set of images of the point p_k in the aligned log maps and cluster R^i_k with DBSCAN [83]. The largest cluster corresponds to the largest agreement between neighboring patches on the 2D coordinates u^i_k of point p_k in patch i. We average all 2D coordinates in the cluster to update u^i_k, and weigh the average based on the distance of each point in R^i_k to the center of its patch. Applying this process to all 2D coordinates U^i in each patch, we get a corrected log map $\hat{U}^i$ for each patch, giving us DSEs that are more consistent with the neighboring DSEs.

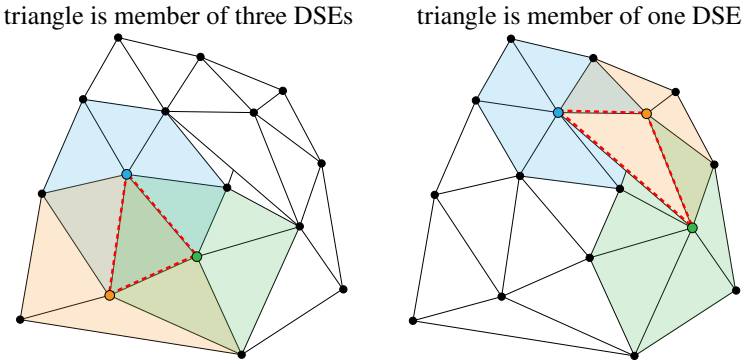

Figure 5.4 – Triangle membership count. Delaunay Surface Elements are shown as colored triangles. Triangles that are part of exactly three DSEs, like the dotted red triangle on the left, result in a manifold triangulation. Triangles that are part of less than three DSEs, like the triangle on the right, result in non-manifold triangulations. We use this property to define a triangle confidence when selecting triangles.

Delaunay triangulation Given a patch P^i and its 2D parameterization $\hat{U}^i$, we can compute a Delaunay Triangulation on the 2D points u^i_j. If $\hat{U}^i$ approximates the log map, this gives us a manifold triangulation of the 3D patch that locally approximates the ground truth surface $\hat{S}$. We define a Delaunay Surface Element $D := (P^i, T^i)$ as the set of Delaunay triangles T^i corresponding to the Voronoi cell centered at p_i. These triangles form an umbrella with p_i as its central point. We restrict our triangulation to triangles that include the central point, as triangulations are increasingly inconsistent with neighboring DSEs as the distance from the central point increases.

Triangle selection Combining the triangles of all DSEs yields a set of candidate triangles that we use in a final triangle selection step to obtain a near-manifold mesh. We base our selection criteria on our DSEs by observing that a triangulation is manifold exactly if all triangles are part of three DSEs (see Figure 5.4). Therefore, we divide our triangles into three confidence bins. Triangles that appear in three different DSEs will be considered the most likely to appear in a manifold triangulation. And triangles that appear only once are considered least likely. Finally, we use the triangle selection process proposed in [153] to produce a triangulation based on our priority queue.

5.3 Results

We evaluate our method by comparing the quality and accuracy of our reconstructed meshes to the current state-of-the-art.

Dataset Since our networks are trained on individual patches, our method is able to train successfully from a small training set of shapes. Each shape provides a large set of patches as training samples. We create a dataset with a total of 91 shapes chosen from

Table 5.1 – Quantitative results on the FAMOUSTHINGI testset. We compare the percentage of non-watertight edges (NW), the Chamfer distance (CD) and normal reconstruction error in degrees (NR).

Method	NW (%)	CD $*1^{e-2}$	NR
ball pivoting	25.7	0.524	6.59
PointTriNet [194]	17.2	0.337	6.24
RVE [31]	9.2	0.344	15.71
IER meshing [153]	5.3	0.343	6.30
α-shapes 3%	2.5	0.939	28.50
α-shapes 5%	1.7	1.064	17.69
Ours	**0.4**	**0.326**	**5.23**

Thingi10k [240] and the PCPNet [107] dataset, that we call FAMOUSTHINGI, since the PCPNet dataset contains several shapes that are well-known in the graphics and vision literature. Each shape is sampled uniformly with 10k points. We compute ground truth log maps at each point using the recent method by Sharp et al. [195]. The training set contains 56 of these shapes and the remaining shapes are used for evaluation. Example shapes and more details are given in Section B.4 of the appendix.

5.3.1 Comparison to Baselines

We compare our method to recent state of the art learning based methods for point-set triangulation, as well as to more classical methods.

Ball pivoting [24] and α-shapes [79]. These two classic techniques use the concept of rolling a ball on the surface to deduce connectivity at points of contact. For ball-pivoting, the ball radius is automatically guessed as the bounding box diagonal divided by the square root of the vertex count. For α-shapes, we report two different choices of the radius parameter α, as 3% and 5% of the bounding box diagonal.

Restricted Voronoi estimation [31] (RVE) This method is the closest existing baseline to our method. It estimates Voronoi cells restricted to the surface by projecting local patches to local tangent planes. Note that this method requires normal information that we estimate from the input point cloud.

PointTriNet [194] and IER meshing [153] We compare our method to two recent learning based methods for triangulating point clouds. We retrain PointTriNet on our dataset. Intrinsic-Extrinsic Ratio Guidance Meshing (IER meshing), however, needs a larger amount of data to train and overfits on our dataset. Since it is not patch based, it needs a larger variety of shapes to train. We use the pre-trained model provided by the authors, that was trained on the larger ShapeNet dataset.

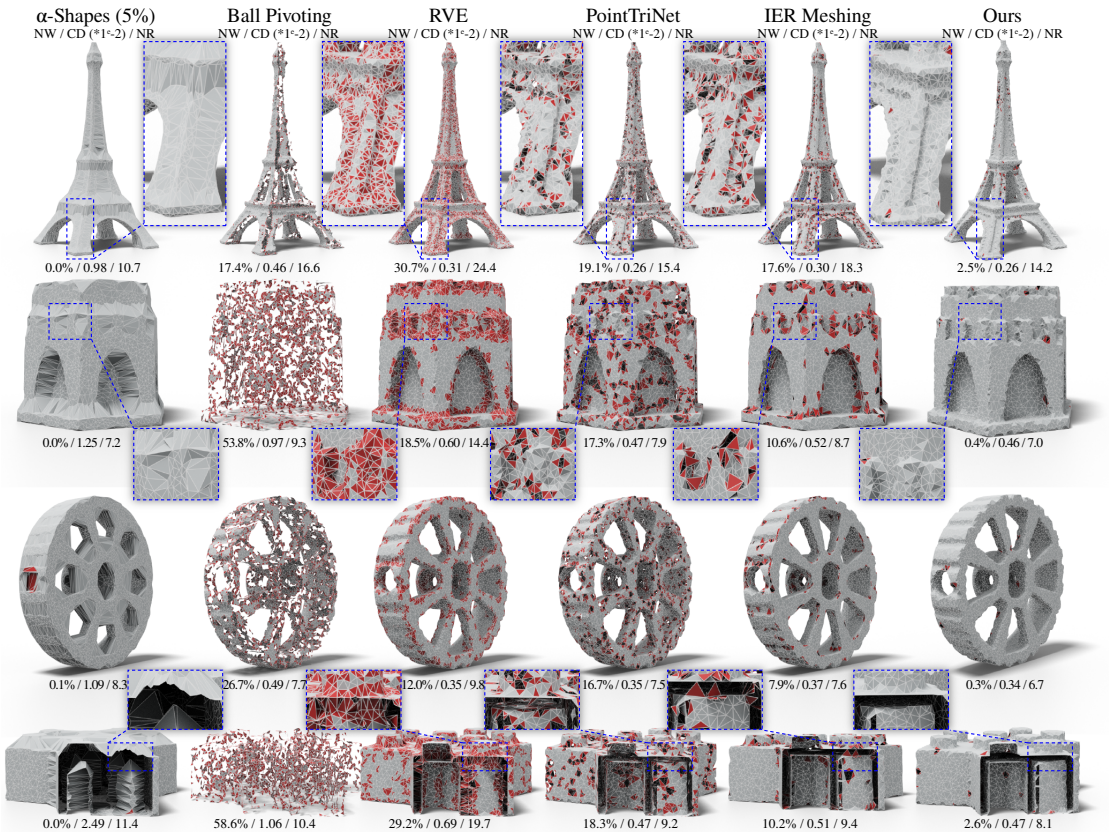

Figure 5.5 – Qualitative comparison. We compare four meshes reconstructed by our method to the results of five current methods. Non-manifold triangles are marked in red and we show both the percentage of non-watertight edges (NW) and the Chamfer distance multiplied by 100 (CD) below each shape. Note that classical non-data-driven methods struggle to separate thin surfaces and data-driven methods have significantly more non-manifold triangles.

Metrics We compare to these methods using two metrics for the mesh quality and two metrics for the mesh accuracy. As mesh quality measures, we use the percentage of non-watertight edges (NW) and the standard deviation (A_σ) of triangle angles in the mesh. Note that due to the triangle selection step, all the produced edges are manifold (have one or two adjacent triangles) but the edges can be open. An angle of 60 degrees corresponds to equilateral triangles, while skinny triangles have more extreme angles.

As a measure of the surface reconstruction accuracy, we use the Chamfer Distance [14, 84] (CD) between a dense point set P_M sampled on the reconstructed surface and a dense point set $P_{\hat{S}}$ sampled on the ground truth surface:

$$\begin{aligned}\mathrm{CD}(P_M, P_{\hat{S}}) \;=\; & \frac{1}{N}\sum_{p_i \in P_M} \min_{q_j \in P_{\hat{S}}} \|p_i - q_j\|_2 \;+ \\ & \frac{1}{N}\sum_{q_j \in P_{\hat{S}}} \min_{p_i \in P_M} \|q_j - p_i\|_2\end{aligned}$$

We also compare the normal reconstruction error (NR). At each vertex of the mesh we measure the angle difference in degrees between the ground truth normal and the normal obtained from our reconstructed mesh.

Quantitative Comparison In Table 5.1, we show a quantitative comparison between our method and the baselines. Our method yields lower chamfer distance, and less non-manifold edges, showing we both better-approximate the surface while at the same time outputting a triangulation with far less non-manifold artifacts. Indeed, only the classic technique of α-shapes manages to come close to our degree of manifoldness, at the cost of lower accuracy, due to filling in concave surface regions (see examples in Figure 5.5).

In Figure 5.7, we evaluate the quality of the generated triangles by considering the histogram of triangle angles. The standard deviation of each method is given next to its name. Our method yields superior triangle quality to all learning-based methods, and to all classic techniques except for ball-pivoting, which achieves better triangle quality by sacrificing manifoldness to a large degree.

Qualitative Comparison We show qualitative results in Figure 5.5, on 4 meshes of our FamousThingi dataset. Non-manifold triangles are visualized in red, with the percentage of non-manifold triangles, as well as the Chamfer distance error, written beneath each result. The figure gives a very clear visual insight to the numbers from Table 5.1: the classic techniques work in a non-adaptive way which enables them to produce meshes with mostly-manifold edges, but they cannot handle thin and tight structures, like the scaffolds of the tower. In contrast, the learning-based methods are more local and can handle the concavities in, e.g., the wheel, but fall short on producing manifold triangulations. Our method, combining the robustness of classic Delaunay triangulation, with modern, data-driven learning techniques, manages to produce triangulations that both respect the original fine geometry and have less non-manifoldness.

Figure 5.6 – Qualitative comparison on ShapeNet [44]. We compare with the two data-driven methods PointTriNet and IER Meshing on five shapes taken from five different categories of the ShapeNet dataset. Our approach results in more manifold meshes, especially in detailed areas like the backrest of the chair.

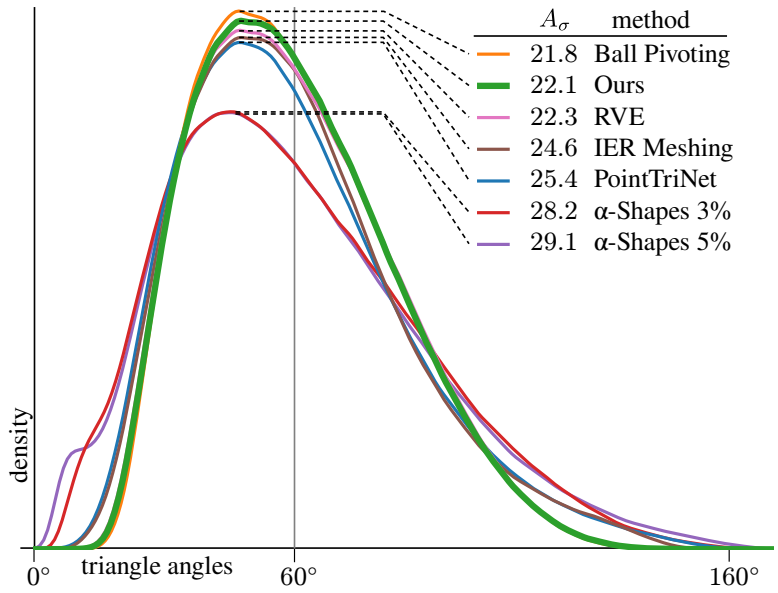

Figure 5.7 – Distribution of triangle angles in the reconstructed meshes. Our method produces better shaped triangles than all other methods except for ball pivoting which sacrifices mesh manifoldness. We show the angle variance next to each method.

We show additional results on five shapes of the ShapeNet dataset [44] in Figure 5.6. Compared to the two data-driven methods PointTriNet and IER Meshing, we improve upon the manifoldness, especially in regions with detailed geometry and high curvature, like the edge of the table, or the backrest of the chair. The results show a similar trend as in our FamousThingi dataset. Note that IER meshing is trained on the ShapeNet dataset while PointTriNet and our method are trained on FamousThingi dataset, demonstrating the ability of our method to generalize to unseen data. We show quantitative and qualitative results on the ShapeNet dataset in Section B.2 and additional qualitative results in B.3 of the appendix.

Limitations Finding a geometrically complex surface, like on parts of the Eiffel tower in Figure 5.5, can be difficult. In such cases, the geodesic neighbors or logmap networks may misclassify/misplace some points. Moreover, thin parts of a model are particularly challenging. We can handle these cases better than existing works (Lego piece of Figure 5.5, or the plane wing in Figure 5.6. More extreme cases, like the leaves of a plant, would require training on a dataset where these cases are more common.

Non uniform sampling We evaluate our method on non uniformly sampled point clouds. In particular we sample points following a probability gradient along the y-axis (horizontal). We observe in Figure 6.1 (bottom) that our method performs better than other learning-based baselines. Note that PointTriNet, IER Meshing and our method have

Table 5.2 – Ablation study over the components of our method. Log map alignment, triangle selection as well as the log map parametrization improve manifoldness in the output meshes.

Method	NW (%)	CD $*1^{e-2}$	NR
Ours w/o align, select	22.51	0.326	7.26
Ours w/o select	10.98	0.348	6.86
Ours w/o log maps	1.18	0.334	5.93
Ours w/o align	1.07	**0.325**	**5.19**
Ours	**0.40**	0.326	5.22

not been retrained on a non-uniformly sampled dataset. We provide further evaluation on non uniformly sampled point clouds in Section B.1 of the appendix.

5.3.2 Ablation study

We evaluate the impact of each step in our pipeline using an ablation study, shown in Table 5.2. We remove one component at a time and compute the percentage of non-watertight edges (NW), Chamfer distance (CD) and normal reconstruction error (NR) as described before. We first remove the *align*ment of the logmaps of the delaunay triangulation, which results in a slight degradation manifoldness. Next, we evaluate the efficacy of our triangle *select*ion process by instead creating a mesh from all triangles in our Delaunay Surface Elements. This results in a significant drop in manifoldness of the triangulation, since we do not achieve perfect alignment of our logmaps. Dropping both the alignment and the selection results in a much more significant decrease in manifoldness than just removing the selection – this hints that the alignment is indeed producing more consistent local DSE's. Lastly, we replace the *log map*s with simple 2D projections, to get the local patch parameterization along the approximated normal vector. Please note that we still use the learned geodesic neighborhood. Manifoldness deteriorates as well, showing the necessity of our specific parameterization method. In particular, the 2D projection parametrization performs poorly for complex shapes such as the Eiffel tower (NW: 5.59% (w/o logmaps), 2.48% (Ours)) or Trilego (NW: 5.59% (w/o logmaps) NW: 1.64% (Ours)) shapes. Note that the Chamfer distance is not significantly increased by the removal of any component from our pipeline, as our method's locality prevents strong errors in the surface location by design, due to considering only the learned geodesic neighborhoods of the surface. We provide additional ablation of the learned Logmap component in Section B.7 of the appendix.

5.4 Conclusion

We presented Delaunay Surface Elements for robust surface reconstruction from points sets. In the process, we combine the best of two worlds: 2D Delaunay triangulation from classical computational geometry which comes with guarantee about mesh quality and

manifoldness; and local logmaps learned using networks, followed by synchonization to get local data-driven 2D projection domains to handle non-planar regions. We demonstrated that the method can be trained with very limited training data and produces near-manifold triangulations that respect the original point set and have a higher mesh quality than the state-of-the-art.

In the proposed method, the final mesh extraction is done via a non-differentiable growing approach. In the future, it would be interesting to also learn the triangle selection via a network. This could enable a truly end-to-end optimization and allow us to optimize for context-specific point distributions accounting for data-priors (e.g., sharp edges) and scanner characteristics. Another direction would be to consider weighted Delaunay triangulations that provide additional freedom to local triangulations.

Chapter 6

Differentiable Surface Triangulation

Triangle meshes remain the most popular data representation for surface geometry. This ubiquitous representation is essentially a hybrid one that decouples continuous *vertex locations* from the discrete topological *triangulation*. Unfortunately, the combinatorial nature of the triangulation prevents taking derivatives over the space of possible meshings of any given surface. As a result, to date, mesh processing and optimization techniques have been unable to truly take advantage of modular gradient descent components of modern optimization frameworks. In this work, we present a *differentiable surface triangulation* that enables optimization for any per-vertex or per-face differentiable objective function over the space of underlying surface triangulations. Our method builds on the result that *any* 2D triangulation can be achieved by a suitably perturbed weighted Delaunay triangulation. We translate this result into a computational algorithm by proposing a soft relaxation of the classical weighted Delaunay triangulation and optimizing over vertex weights and vertex locations. We extend the algorithm to 3D by decomposing shapes into developable sets and differentiably meshing each set with suitable boundary constraints. We demonstrate the efficacy of our method on various planar and surface meshes on a range of difficult-to-optimize objective functions. Our code can be found online: https://github.com/mrakotosaon/diff-surface-triangulation.

6.1 Introduction

Triangle meshes are arguably the most predominant surface representation, both in geometry processing and computer graphics, as well as in other fields such as computational geometry and topology. The popularity of triangle meshes comes from their simplicity, flexibility, and the existence of many data structures for efficient mesh navigation and manipulation [66, 65, 29, 203]. Many methods have been developed to compute or modify triangulations of given surfaces or point clouds, while promoting properties such as alignment to shape features (e.g., ridges or creases), adapting sampling density to geometric detail, or triangle aspect ratio (see [40, 23] for an overview).

Unfortunately, as of now, no method has been proposed to enable a continuous, differentiable representation of triangulations. This is mainly due to the fact that in addition to the *continuous* spatial aspect - the position of each vertex - triangulations also have a *discrete* combinatorial component - the connectivity, i.e., the set of edges and triangles connecting the vertices. As a result, existing algorithms either optimize the mesh quality by moving the vertex locations while keeping their connectivity fixed [171], re-mesh from scratch, or iterate between updating the vertex positions and their connectivity, e.g., [121, 204].

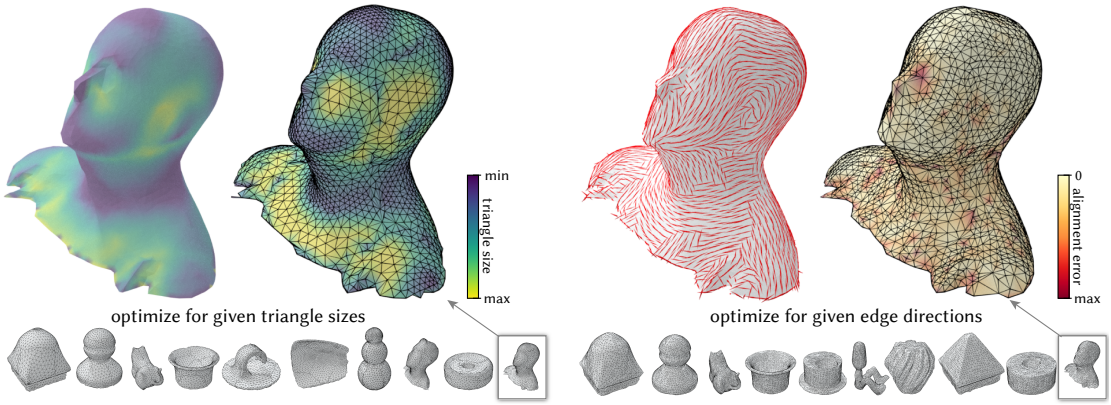

Figure 6.1 – We present a fully differentiable approach for optimizing triangle meshes both in 2D and on surfaces. Our approach allows to optimize the mesh using any differentiable objective function, based on vertex positions or shapes of triangles using continuous optimization techniques. Here we demonstrate meshes obtained on a surface by optimizing for (left) sizes of triangles to depend inversely on the mean absolute curvature value, and (right) alignment of triangle edges to the maximal principal curvature directions (i.e. triangle edges tend to follow the vector field). This optimization is done in a fully differentiable manner without any post-processing or combinatorial operations such as edge flips or vertex splits. Our framework is general and can be thus integrated within modern optimization and learning modules.

This lack of a unified differentiable representation is particularly unfortunate in light of recently-introduced gradient-based optimization frameworks such as Pytorch [180] and TensorFlow [1] for Machine Learning applications. These frameworks rely on the differentiablity of the pipeline and enable modular design. In absence of such a differentiable triangulation framework, current deep-learning pipelines either perform surface meshing during post-processing, or use formulations that are learned via proxies [151, 194, 153, 186], which typically do not give explicit access to the resulting triangle mesh structure.

In this work, we devise what we believe to be the first formulation for *differential triangulation*, enabling gradient-based optimization for per-face and/or per-vertex objectives, such as size and curvature alignment. Our approach is general, can be applied to manifolds represented in any explicit representation, is modular, and supports optimizing for any objective that can be expressed as a differentiable function with respect to triangle properties like size and angles.

The main technical challenge in devising a differentiable triangulation is developing a smooth representation that allows to control both the vertex positions and the (inherently-combinatorial) mesh structure, while also ensuring the resulting mesh is always a 2-manifold. Our core idea is to use the concept of a weighted Delaunay triangulation (**WDT**) [58]. It considers a given set of vertices, along with per-vertex weights, which define a unique triangulation using a Voronoi-like partition of space.

In this paper we propose a *differentiable* weighted Delaunay triangulation (**dWDT**),

by considering (arbitrary) triplets of vertices and whether they constitute a triangle in the triangulation defined by the weights and vertices. While in classic **WDT**, this existence receives a binary value, we generalize that definition by assigning inclusion scores to triangle membership, thus giving them a *soft* association. We demonstrate that this relaxation provides a unified control over both the vertices and the mesh structure, and can be used to directly optimize any (differentiable) objective function defined on the triangles. Intuitively, we define the triangle inclusion scores in terms of Voronoi diagram distances that represent how close a certain triangle is from inclusion into (or removal from) the triangulation. Represented as a continuous quantity, we can optimize triangle inclusion scores as a function of vertex positions and weights. Importantly, Memari et al. [165] showed that, in 2D, *any* triangulation can be represented through a perturbation of a **WDT**, in other words, *any* triangulation can be reached by adjusting vertex positions and weights, and then applying a **WDT**. Therefore, our approach is both differentiable and generic, allowing to accommodate a wide range of mesh structures.

To apply our relaxation to 3D surfaces, we decompose the source into local patches, and then perform per-patch differentiable meshing with appropriate boundary constraints. For example, in Figure 6.1 we show triangulations obtained by optimizing for different objective functions, given the same original underlying surface models. The modular nature of our approach makes it easy to switch between target objective functions. Similarly, we can triangulate different surface representations (see Figure 6.9 for a triangulation of an analytic surface defined by a function).

We evaluate our method to produce 2D and 3D meshes optimized for a mix of target objective functions such as shape/size of triangles, and alignment to given vector fields, thereby highlighting that our approach is both more flexible, and can accommodate for more diverse objectives than alternative approaches.

6.2 Related Work

Surface remeshing and triangle mesh optimization are both extremely well-studied problems in computational geometry, computer graphics, and related fields. Below we review methods most closely related to ours, and refer to recent surveys, including [137, 50, 6, 136] for a more in-depth discussion.

Simplification-based approaches A common objective for surface remeshing is reducing the number of elements in the final mesh. As a result, especially early remeshing techniques, starting with the pioneering QEM approach [93], often focused on preserving mesh quality during simplification (see [93] for a survey of local methods). Such methods are typically based on edge-collapse operation followed by vertex position optimization, and have been extended both in terms of efficiency, e.g., [126, 176], the use of various metrics [174] including feature preservation [217], and even using spectral quantities [145] during edge collapse. However, such approaches are essentially greedy and typically do not allow to optimize mesh properties based on general structural criteria.

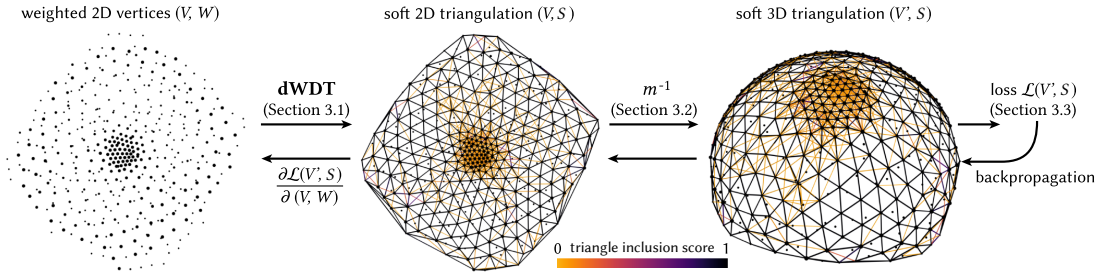

Figure 6.2 – **Overview of our approach.** We propose a differentiable weighted Delaunay Triangulation (**dWDT**) to create a soft triangulation from a set of 2D vertices V with associated associated weights W (shown as marker size). In the soft triangulation, triangles have inclusion scores S of being part of the triangulation. We illustrate triangle inclusion scores as edge colors (using the largest inclusion score of the two adjacent faces) and only show triangles with inclusion score > 0.001. The 2D vertices V are lifted to form a soft 3D triangulation on the manifold's surface using a fixed mapping m^{-1}. Since the pipeline is fully differentiable, we can propagate gradients of any differentiable loss on the 3D triangulation back to the vertex positions V and weights W. Note that by choosing appropriate weights W, our network can ignore points and produce a triangulation over a subset of points, if desired.

Local methods A related set of methods includes approaches based on local mesh modification while aiming to improve the overall mesh quality, e.g., [122, 77, 233]. In addition to edge collapse, these local operators include edge flipping, edge splitting, and vertex translation. A prominent method in this category is real-time adaptive remeshing (RAR) [77], which uses an adaptive sizing function and edge flipping to optimize the mesh quality and vertex valence. This framework was recently extended for efficient *error-bounded* remeshing [51] through a use of a range of powerful local refinement operations. Similarly, Explicit Surface Remeshing (ESR) [200] is another efficient method for remeshing based on local refinement operations coupled with angle-based smoothing. The more recent Instant Meshes [130] technique advocates using local optimization and smoothing, while aiming to optimize potentially global consistency. This results in a powerful and efficient framework, capable of handing both isotropic triangular or quad-dominant meshes. Nevertheless, as with other local techniques the topology (i.e. the connectivity between vertices) and geometry are handled separately, preventing a unified differentiable, global mesh optimization.

Delaunay and CVT-based methods Another powerful set of remeshing methods, more closely related to our approach are based on Delaunay triangulations, and centroidal Voronoi tessellations (CVT). The former category includes approaches based on triangle refinement by flipping non-locally Delaunay (NLD) edges [78] and defining an *intrinsic* Delaunay triangulations [85]. Furthermore, global optimization techniques have also been used for finding optimal Delaunay triangulations [47] under the assumption that vertex connectivity is fixed. In a different line of work, centroidal Voronoi tessellations (CVT) have been used for finding an approximately uniform vertex distributions, so that their

Voronoi diagram (and thus its dual, the Delaunay triangulation) is well-shaped, e.g., [76, 225, 214] among many others. Such methods have also been extended, for example, to explicitly penalize obtuse and sharp angles [226] and to *anisotropic* remeshing by embedding in an appropriate (e.g., feature or curvature-aware) space [148]. Nevertheless, the final shape of the triangulation is difficult to control using these methods, and it is not easy to combine multiple objective functions in a coherent optimization strategy.

Optimization-based approaches Finally we also note methods based explicitly on optimizing an objective. This includes both local, e.g., [121, 77] and global optimization, e.g., [205, 160] strategies (see also Section 4.7 in [136]). Existing optimization strategies most often rely on either smoothness energies [202, 63, 89], use sampling [90] or a variant of CVT, e.g., [224] to optimize vertex positions. In both cases, while the *positions* of the vertices can be optimized, the connectivity is only defined implicitly and updated separately, typically without explicitly taking into account the optimization objective. More fundamentally, the mesh structure is purely combinatorial, preventing the use of powerful tools based on differentiability.

In contrast to these approaches, we propose a fully differentiable framework that allows to jointly optimize for both vertex positions and triangle mesh connectivity, by using a soft version of the weighted Delaunay triangulation (**WDT**). Our method is inspired by theoretical results demonstrating that in 2D any triangulation can be represented through a perturbation of a **WDT** [165]. Importantly, the same result does not hold for the standard Delaunay triangulation, and therefore optimizing over the *weights* of the **WDT** as well as the *vertex positions* allows significantly more control over the shape of the final triangulation and even allows ignoring some input points if they are deemed unnecessary (i.e., high weights) in the final triangle mesh.

Importantly, the differentiable nature of our approach allows optimizing for a range of criteria jointly, simply by formulating a single (differentiable) objective function. Furthermore, it enables optimization of both vertex positions and criteria that depend on the connectivity in a unified framework. Finally, our differentiable meshing block can be also ultimately be used as part of a larger, differentiable shape processing or design system.

Weighted Voronoi and power diagrams. Weighted Voronoi diagrams are also known as *power diagrams*, and have been researched extensively in the context of triangulations [98]. In computer graphics, they have been used for various tasks such as computing blue noise [59], or simulating fluid dynamics [60]. [169, 100, 165] considered power diagrams in the context of formulating different triangulation *duals*. They also propose to optimize objectives on the dual. This is a different context and use-case than our differentiable formulation, which is geared towards a gradient-guided optimization of arbitrary geometric objectives on the *triangulation*.

Differentiability in computer vision Recently, with the success of deep learning in computer vision, making common operations differentiable has started to gain research interest. In particular relaxing hard condition for deep learning purposes into soft

formulations has been used for tasks such as RANSAC [36], rendering [154] or shape correspondence [161] among others. Similarly to these methods, we present a soft formulation of triangle existence.

6.3 Method

Let (V', T) represent a triangulation of a surface in 3D space, with $V' = (v'_1, \dots, v'_n), v'_i \in \mathbb{R}^3$ vertices, and T its triangular faces. A common strategy for triangulating a manifold surface is to first find a 2D parameterization that maps the surface to a planar 2D domain, then sample a set of vertices $V = (v_1, \dots, v_n), v_i \in \mathbb{R}^2$ in the 2D domain, and compute a triangulation which respects the chosen vertices. Our method relies on the ubiquitous Delaunay triangulation [50, 61] (**DT**), used for triangulating a given 2D vertex set. We denote it as $T = \mathbf{DT}(V)$. A Delaunay triangulation always includes all chosen vertices, and is, uniquely defined with respect to them, as long as the points are in general position. In order to gain more control over the triangulation, one can consider a *weighted* Delaunay triangulation [203, 11] $T = \mathbf{WDT}(V, W)$, where each vertex v_i has a scalar weight w_i, with $W = (w_1, \dots, w_n)$. Traditional methods for computing a **WDT** are typically not differentiable, as the space of all possible faces is combinatorial.

We propose a differentiable weighted Delaunay triangulation $\mathbf{dWDT}(V, W)$ that is differentiable with respect to both the vertex positions V and the weights W. In conjunction with a parameterization m that defines a bijective and piecewise differentiable mapping m from a surface in 3D to a 2D parameter space, **dWDT** enables a differentiable pipeline for triangulating 3D domains. We describe our differentiable triangulation approach in two parts (see Figure 6.2). First, in Section 6.3.1, we describe the differentiable weighted 2D Delaunay triangulation **dWDT**. In this part, we first focus on the definition of **DT** and the existence of triangles, w.r.t. vertex positions and weights, and then replace the binary triangle existence function with a smooth *triangle inclusion score*, again defined w.r.t the vertex positions and weights, in a way that naturally follows from the definition of **DT**. This yields a *soft* and differentiable notion of a triangulation that can easily be generalized to a weighted Delaunay triangulation. Then, in Section 6.3.2, we describe a parameterization m that maps between a manifold surface and our 2D Delaunay triangulation to obtain **dWDT** on 3D surfaces, before describing the losses and optimization setup in Section 6.3.3.

6.3.1 Differentiable Weighted Delaunay Triangulation

Assume we are given a set of vertices $V = \{v_1, ..., v_{|V|}\}$ with $v_j \in \mathbb{R}^2$. Consider the set of all possible triangles defined over these vertices, i.e., all possible triplets of vertices:

$$T^* = \{(v_j, v_k, v_l) \,|v_j, v_k, v_l \in V\}\,. \tag{6.1}$$

Any triangulation of the vertices V is a subset of all possible triangles $T \subset T^*$ on V, and we can consider the triangulation's *existence* function $e : T^* \to \{0, 1\}$, defined for any

triplet $t_i \in T^*$ as

$$e_i = \begin{cases} 1 & t \in T \\ 0 & t \notin T. \end{cases} \tag{6.2}$$

From this perspective, the binary and discrete existence function is the cause of the combinatorial nature of the triangulation problem. Hence, our main goal is to define a *smooth* formulation in which this function is differentiable as to enable gradient-based optimization. We achieve this by extending **WDT** to the smooth setting.

Towards gaining intuition into **WDT**, let us first consider the classic, non-weighted *Delaunay triangulation* $\mathbf{DT}(V)$ of a given set of vertices V. This triangulation is defined by considering each possible triangle $t \in T^*$ and deeming it as part of the triangulation $\mathbf{DT}(V)$ if and only if its circumcenter is the shared vertex of the three Voronoi cells centered at the triangle's vertices (see Figure 6.3 for an illustration). The Voronoi cell of vertex v_j is defined as the set of points in $\mathbb{R}^2$ closer to v_j than to any other vertex $v_k \in V$.

Said differently, each pair of vertices (v_j, v_k) divides the 2D plane into two half-spaces: the set of points closer to v_j, denoted as $H_{j<k}$, and the set of points closer to v_k, denoted as $H_{k<j}$. The Voronoi cell a_j centered at v_j is defined as the intersection of half-spaces $a_j \bigcap_{k \neq j} H_{j<k}$. The triangle circumcenter is the intersection point of the three half-space boundaries between the three vertex pairs that define its edges. Hence, we can define the existence function of the Delaunay triangulation for a triplet of vertices, $t_i = (v_j, v_k, v_l)$ with circumcenter c_i as

$$e_i = \begin{cases} 1 & \text{if } \; c_i = a_j \cap a_k \cap a_l \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

Parameterizing Triangle Existence with respect to V. We are interested in how the triangulation T changes as the vertex positions are changed - namely, we aim to analyze the range of vertex positions that do not change its membership function e_i of a triangle t_i.

For any triangle $t_i = (v_j, v_k, v_l)$, we consider the three *reduced* Voronoi cells $a_{j|i}$, $a_{k|i}$, $a_{l|i}$ respectively around the triangle's vertices v_j, v_k, v_l, where we define a reduced Voronoi cell $a_{j|i}$ centered at the triangle vertex v_j as the Voronoi cell created by ignoring the two other vertices of the triangle, v_k and v_l (see Figure 6.3). The triangle t_i is part of the triangulation T as long is its circumcenter c_i remains inside the reduced Voronoi cells around its vertices. Similarly, t_i is not part of T as long as its circumcenter remains outside its three reduced Voronoi cells. Note that, by construction, the circumcenter simultaneously enters or exits the three reduced Voronoi cells. Thus, we can re-formulate the triangle existence e_i as:

$$e_i = \begin{cases} 1 & \text{if } \; c_i \in a_{x|i} \text{ for any } x \in \{j, k, l\} \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

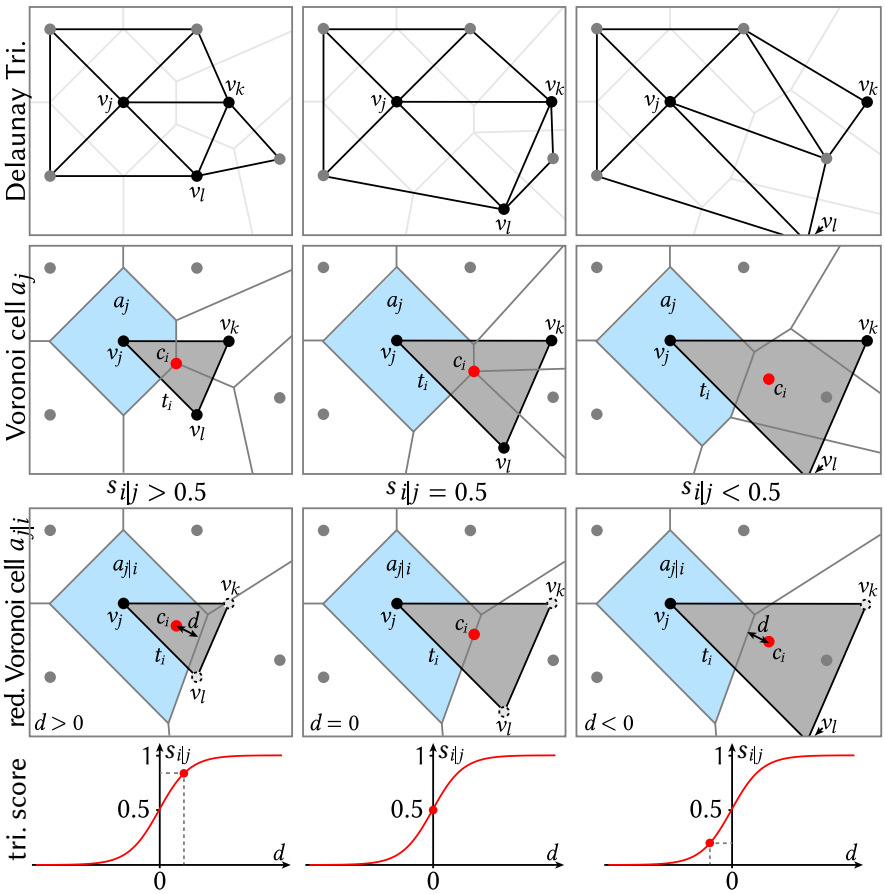

Figure 6.3 – **Triangle inclusion score and reduced Voronoi cell.** A reduced Voronoi cell for a given triangle t_i at a vertex v_j is constructed from the point set that excludes the two other vertices of the triangle. A triangle t_i exists in the Delaunay triangulation, as long as its circumcenter c_i remains inside the reduced Voronoi cell. We base triangle inclusion scores $s_{i|j}$ on the signed distance from c_i to the boundary of the reduced Voronoi cell.

Continuous Triangle Inclusion Scores We now turn to making **DT** differentiable by *relaxing* the binary existence function e_i defined in Equation (6.4) into a continuous inclusion score function, s_i, denoting the inclusion score of a triangle $t_i \in T^*$ to exist as a member of the triangulation T, defined with respect to vertex positions. The inclusion scores are based on the signed distance of the triangle circumcenter to the boundary of the reduced Voronoi cells at the triangle vertices: considering a single vertex v_j of the triangle t_i, and its reduced Voronoi cell $a_{j|i}$, the inclusion score is defined as:

$$s_{i|j} := \sigma\big(\alpha \; d(c_i, a_{j|i})\big), \tag{6.5}$$

where d is the signed distance (positive inside, negative outside) from a point c_i to the boundary of a reduced Voronoi cell $a_{j|i}$, and α is a scaling factor for the width of the

Sigmoid σ (we use $\alpha = 1000$ in all experiments). The Sigmoid gives a smooth transition from an inclusion score close to 1 inside the reduced Voronoi cell to an inclusion score close to 0 outside, with an inclusion score 0.5 if the circumcenter lies on the boundary of the reduced Voronoi cell, i.e., exactly when the discrete triangle membership changes. The triangle inclusion score s_i can then be defined as the average over the three inclusion scores at its vertices v_j, v_k, and v_l:

$$s_i = \frac{1}{3}(s_{i|j} + s_{i|k} + s_{i|l}). \tag{6.6}$$

Note that since the circumcenter simultaneously enters/exits the reduced Voronoi cells around each vertex, all three inclusion scores equal 0.5 at a discrete membership transition.

For each triangle t_i, we store the triangle inclusion score s_i and the three inclusion scores $s_{i|j}$, $s_{i|k}$ and $s_{i|l}$ defined for its three vertices, yielding a *soft* 2D triangulation (V, S) with inclusion scores S. We store the inclusion scores $s_{i|j}$ in addition to the triangle inclusion scores s_i, since most losses that we use are defined on vertices where using a triangle's vertex inclusion scores is more convenient. We can, subsequently, convert this soft triangulation into a discrete 2D triangulation (V, T), by selecting all triangles where $s_i > 0.5$. This gives us the same results as the discrete **DT**, the final triangulation is guaranteed to be manifold.

Since the number of all possible triangles T^* grows cubically with the vertex count, we reduce the number of triangles under consideration by observing that vertices in the triangles of a Delaunay triangulation are typically within the k-nearest neighbors of each other, for some small k (we use $k = 80$ in all experiments). Thus, at each Voronoi cell a_j, we only consider triangles that are within the k-nearest neighbors of v_j and set all other triangle inclusion score implicitly to 0. Note that since the 2D vertices V change positions during optimization, we recompute nearest neighbours after each iteration of our algorithm.

Weighted Delaunay triangulation Our relaxed formulation of the Delaunay triangulation can naturally be extended to the *weighted* Delaunay triangulation **WDT**, where weights are associated to each vertex. The weights allow shifting the boundary between the two half-spaces $H_{j<k}$ and $H_{k<j}$, by the relative weights w_j and w_k of the two vertices. The weighted half space $H_{j<k}$ is defined as the set of points $x \in \mathbb{R}^2$ where

$$\|x - v_j\|_2^2 - w_j^2 \leq \|x - v_k\|_2^2 - w_k^2 \tag{6.7}$$

so that a larger weight pushes the boundary away from the vertex. This allows generalizing the definition of the Voronoi cell to a weighted Voronoi cell. As a result, the existence function Equation (6.4) and the inclusion score $s_{i|j}$ in Equation (6.5) can be used as-is, with the modified definition of the half-planes, and considering the *weighted* circumcenter of the triangle. This makes the inclusion scores S of the soft triangulation (V, S) a function of both the vertex position V and their weights W.

Thus, weights enable further control over the resulting triangulation, by enabling modifications the Voronoi cells (and therefore, the triangulation itself). In fact, note that

it is even possible for a vertex to be excluded from a **WDT** (i.e., not be part of any triangle), if the weight difference to any other vertex is so large that the boundary line between the two half-spaces shifts past one of the vertices - a property not possible with classical Delaunay triangulation. We will make use of this property to allow our method to ignore vertices deemed unnecessary, hence producing triangulations with a reduced number of vertices. In the following, we consider the weighted triangle circumcenters, denoted by c_i, and the weighted Voronoi cells, denoted by a_i.

6.3.2 3D Surface Parameterization

So far we have defined differentiable triangulations of 2D sets of vertices. In order to apply our **dWDT** on a 3D surface $\mathcal{M}$, we reduce the problem to a set of 2D (triangulation) problems.

First, we construct a bijective piecewise differentiable mapping m between the manifold and the 2D plane, i.e., a 2D parameterization. Next, we elaborate on the computation of this parameterization. As a pre-process, since we are not concerned with the original triangulation but only the underlying surface it represents, we initially remesh input models using isotropic explicit remeshing [53] to yield meshes constituting between $3.5 - 4.5$K triangles. We normalize each model to unit area. We then decompose the manifold into a set of separate patches $\{\mathcal{P}_1, \mathcal{P}_2, \ldots\}$ that can be individually parameterized with less distortion than the whole shape. Individual patches are found with a spectral clustering approach [173], using the adjacency matrix for affinity. We used 10 patches in all experiments.

Then, we construct a low-distortion mapping m between the surface of a patch $\mathcal{P}_i$ and the 2D plane using Least-Squares Conformal Maps [149] (LSCM). To lower the distortion of the mapping for patches that are far from developable, we first measure the distortion as the deviation of the local scale factor from the global average. Patches with high distortion are cut along the shortest geodesic between the area of maximum distortion and any existing boundary. This process is repeated until the maximum distortion of all patches, measured as the ratio between local scale and global average is above 15% and the mapping is bijective. Finally, we normalize the 2D parametrization of each patch to have equal average edge length. We compute this mapping once, as a preprocess, and reuse it in all steps of the optimization.

Differentiable 3D Surface Triangulation Given the mapping m, we can pull back the computed 2D triangulation (V, T) to a part of the 3D surface $\mathcal{P}_i$ using the inverse mapping m^{-1}. Thus, our differentiable triangulation of a 3D surface patch is defined as:

$$(V', S) := \big(\big(m^{-1}(v_1), \ldots, m^{-1}(v_n)\big),\ \mathbf{dWDT}(V, W)\big), \tag{6.8}$$

which gives us the soft 3D triangulation (V', S) that consists of a set of 3D vertices V' and triangle inclusion scores S. Note that the inclusion scores are differentiable functions of the 2D vertices V and their weights W, and that we can obtain a manifold discrete mesh at any time by selecting all triangles with inclusion scores > 0.5. Since the mapping m is piecewise differentiable, any loss $\mathcal{L}$ can be applied directly to the 3D vertices V' and

triangle inclusion scores S, allowing gradients to propagate back to the parameters V and W that define V' and S. We highlight that similarly to Leaky ReLU activations, the piecewise differentiability does not significantly impact optimization. We discuss the losses we use in our experiments in Section 6.3.3.

Boundary preservation Special care must be taken to preserve the boundary of each patch, so that putting the patches back together does not result in gaps or overlaps. We use a two-part strategy to ensure pieces fit back together. First, we define a loss that repels vertices from the boundary of a patch, which we describe in Section 6.3.3. Second, we perform a post-processing step that cuts the 2D mesh (V, S) along the 2D boundary, based on a triangle flipping strategy along the boundary. Namely, we use the simple strategy described in [193] between consecutive boundary points of the optimized patches. The boundary between patches is therefore kept fixed before and after the optimization step.

6.3.3 Losses and Optimization

Our differentiable triangulation allows us to optimize a triangular mesh on a surface in 3D using any differentiable loss defined on the 3D vertex positions V and triangle inclusion scores S. We experiment with several different losses, combinations of which are useful for both traditional applications, as well as novel ones, as we experimentally show in Section 6.4.

The *triangle size loss* $\mathcal{L}_s$ encourages triangles to have a specified area:

$$\mathcal{L}_s(V', S) := \frac{1}{\sum_{i,j} s_{i|j}} \sum_{i,j} s_{i|j} \left(0.5 \, \|(v'_k - v'_j) \times (v'_l - v'_j)\|_2 - A(v_j)\right)^2, \tag{6.9}$$

where v'_j, v'_k, and v'_l are the 3D vertices of triangle t_i, and $A(v_j)$ is the target area at vertex v_j, where A is defined as a continuous function over the 3D surface. This loss allows us, for example, to coarsen a triangulation, when used in conjunction with other losses. Note that the size of the triangles is not constrained by the initial number of vertices - due to the **WDT** our optimized result can contain fewer vertices than the initial triangulation.

The *boundary repulsion loss* $\mathcal{L}_b$ encourages vertices to stay inside the 2D boundary of the patch $\mathcal{P}$ during the optimization:

$$\mathcal{L}_b(V, \mathcal{P}) := \frac{1}{|V|} \sum_j e^{\epsilon - \min\left(\epsilon, \, (v_j - b_j) n_j^b\right)}, \tag{6.10}$$

where b_j is the point on the boundary closest to the vertex v_j and n_j^b is the 2D boundary normal at that point (pointing inward). The repulsion loss is non-zero below a (signed) distance ϵ from the boundary as we show in Figure 6.4. We set ϵ to 0.01 in our experiments. Note that we do not use our triangle inclusion scores in this loss, since we want all vertices to remain inside the boundary, irrespective of inclusion scores. We note that since $\mathcal{L}_b$

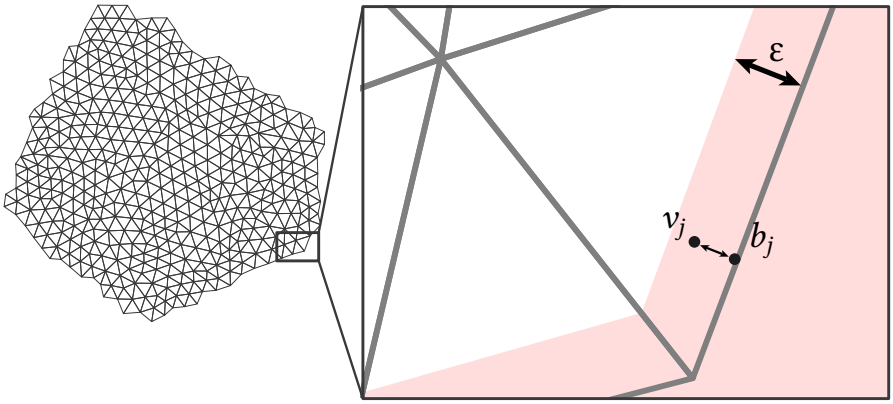

Figure 6.4 – **Boundary repulsion loss.** The repulsion loss is non-zero below a (signed) distance ϵ from the boundary. Non-boundary vertices inside the red region are pushed towards the center of the patch.

has a local effect and does not rely on global properties of the patch, patches can be non-convex.

The *angle loss* $\mathcal{L}_a$ encourages triangles to be equilateral.

$$\mathcal{L}_a(V', S, \mathcal{P}) := \frac{1}{\sum_{i,j} s_{i|j}} \sum_{i,j} s_{i|j} \;\; \left|\cos(\angle_j) - \cos(\pi/3)\right|, \tag{6.11}$$

where $\angle_j$ is the corner angle of triangle t_i including vertex v_j. Note that this loss can be modified to produce isosceles triangles.

The *curvature alignment loss* $\mathcal{L}_c$ encourages two edges per vertex to align to the two directions of the minimum principal curvature vector field C. We define it as,

$$\begin{aligned}\mathcal{L}_c(V', S, \mathcal{P}, C) :=& \frac{-1}{\sum_{i,j} s_{i|j}} \sum_j \Big(\\ & \mathrm{LSE}(\cup_{i\in\mathcal{N}_j}\{C(v_j)\cdot h_{jk}\; s_{i|j}, C(v_j)\cdot h_{jl}\; s_{i|j}\}) \\ & + \mathrm{LSE}(\cup_{i\in\mathcal{N}_j}\{-C(v_j)\cdot h_{jk}\; s_{i|j}, -C(v_j)\cdot h_{jl}\; s_{i|j}\})\Big) \\ \text{with } h_{jm} =& (v'_j - v'_m)/\|v'_j - v'_m\|_2,\end{aligned} \tag{6.12}$$

where $\mathcal{N}_j$ are the triangles adjacent to v'_j, and v'_j, v'_k, v'_l are the 3D vertices of triangle t_i. LSE denotes the smooth maximum function LogSumExp over the weighted alignment scores of all edges adjacent to vertex v'_j, where each triangle contributes two edges corresponding to h_{jk} and h_{jl}. Intuitively, we want to maximize the alignment of the best-aligned edge in a star of each vertex, for both the positive and negative target guidance direction $C(v_j)$, which is the principal curvature field evaluated at v_j.

Optimization Given a loss $\mathcal{L}$, as a sum of a selection of the terms above, we optimize the 3D mesh M, parameterized by the 2D vertex positions V and vertex weights W.

Since our framework is completely differentiable, we use the Adam [139] optimizer. We initialize all vertex weights with random values and use the mapping of the input mesh vertices to 2D as the initial 3D vertex positions. We use a learning rate of 0.0001 in all experiments. Please refer to the supplementary video for evolving triangulations over optimization iterations.

6.4 Results

We next describe experiments that highlight the key advantage of our method - differentiability, which enables plugging in and mixing any combination of differentiable losses, circumventing the need to design a specialized optimization method for each loss combination. Practically, the experiments show the efficacy of our method, and its ability to produce superior results than state-of-the-art methods that are specifically tailored to those specific applications. Code of our method is available at github.com/mrakotosaon/diff-surface-triangulation.

6.4.1 Customized Triangulation

Most triangulation tasks are formulated via user-provided requirements that are imposed on the resulting triangulation, such as desired triangle sizes or edge alignment. We employ our differentiable losses in two common scenarios, shown in Figures 6.1 and 6.5. We evaluate our method in both scenarios on 140 randomly selected meshes (among those with genus 10 or less) sampled from Thingi10k [240].

(i) Triangle size. We first optimize the triangulation to match a given distribution of triangle sizes, represented as a scalar field over the surface. We chose to assign sizes that are the reciprocal of the mean absolute curvature value, so that high curvature regions receive a finer tessellation than lower-curvature regions. We sum the losses $\mathcal{L}_s$, $\mathcal{L}_b$, and $\mathcal{L}_a$ with weights 0.5, 500, and 10^7, respectively, in order to scale each loss to the same range. Qualitative results are shown in the left half of Figure 6.5.

As evaluation metric, we take the absolute difference between the resulting triangle size and the target size distribution. Since we are interested in the distribution of relative triangle sizes rather than the absolute sizes, we normalize the triangle sizes per model to have zero mean and unit standard deviation. To compare our triangle sizes to the continuous target size distribution, triangle size at each vertex is defined as the average size of all adjacent triangles. In Figure 6.5, normalized triangle sizes are shown as colors while the numbers below each result show the RMSE over all vertices.

We compare our method to the remeshing method of Loseille [157], a state-of-the-art method for remeshing that can be guided by a given triangle size field, and show a qualitative comparison on a subset of shapes in Figure 6.5. In most cases our method can reproduce the target size distribution more accurately. Note, for example, the size distribution on the top of the pawn, on the heads, on the rim of the hat and on the cat's hind.

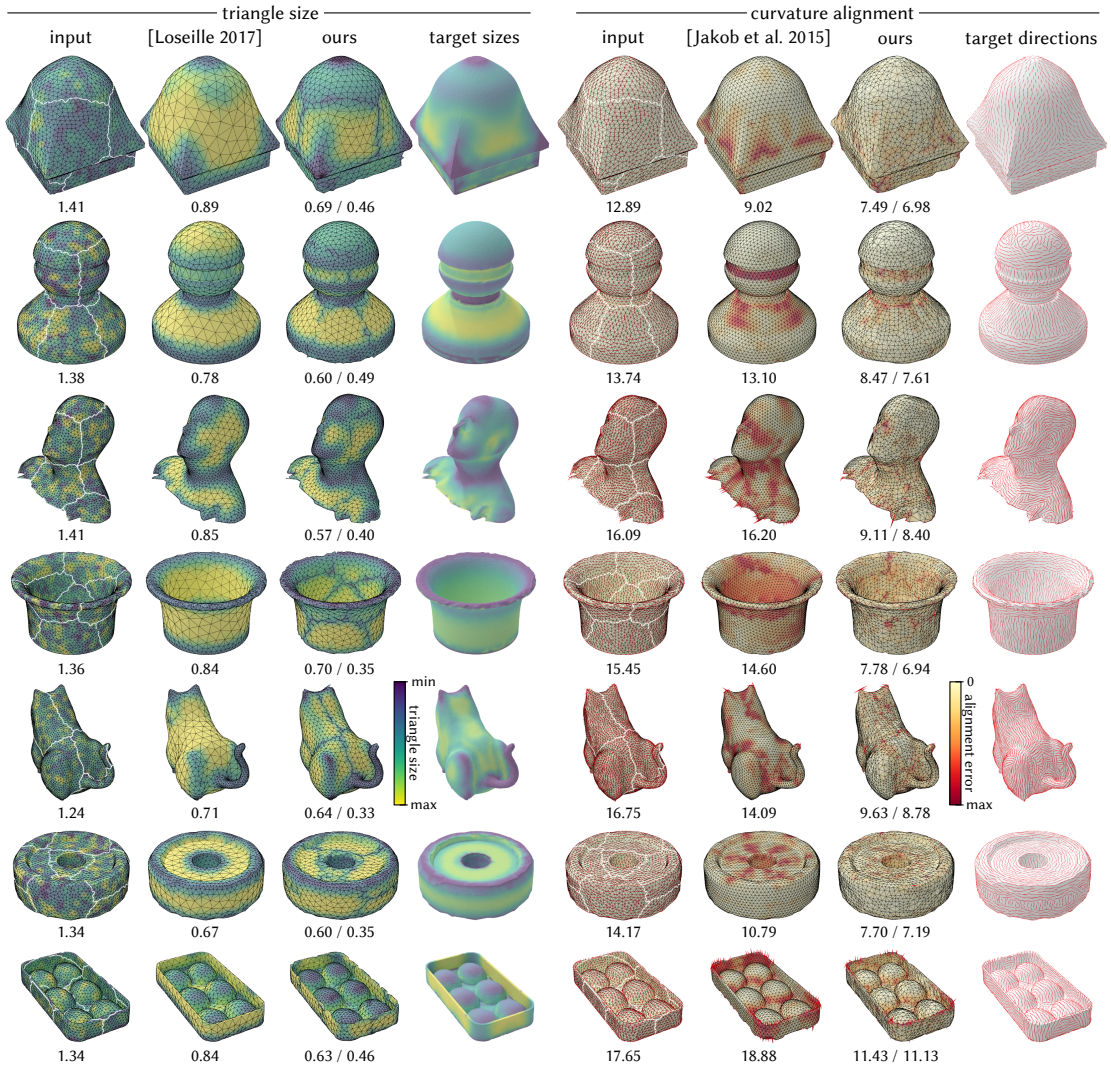

Figure 6.5 – **Qualitative Results.** We show two applications of our approach. In the left half of the figure we optimize for given target triangle sizes, and compare with a state-of-the-art remeshing method [157] (triangles are colored according to size). In the right half, we optimize for edges that are aligned to the principal curvature directions and compare with Instant Meshes [130] (colors illustrate alignment errors). Boundaries of the patch decomposition are shown as white lines on the input meshes. The average error is given below each result - for our method we give the error with / without faces adjacent to a patch boundary. Note that our differentiable triangulation more accurately satisfies the target triangle sizes or edge directions.

(ii) Vector-field alignment. In our second scenario, we optimize 3D meshes with the loss $\mathcal{L}_c$ that encourages edges to align with a given vector field. We chose to use minimum principal curvature directions to encourage meshes which edges that adhere

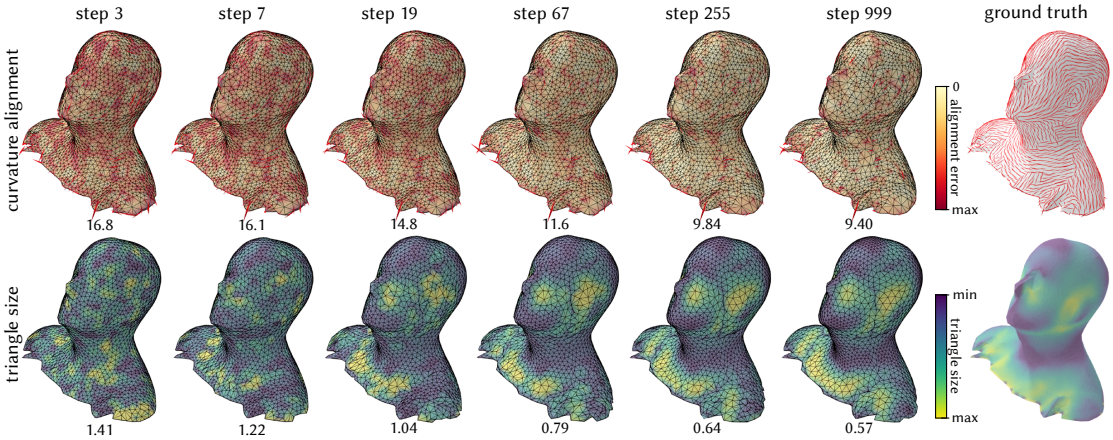

Figure 6.6 – **Optimization steps.** We show our results at different optimization steps for curvature alignment (top row), and triangle size (bottom row).

to ridge lines and geometric features. At the same time, we emphasize that any other user-prescribed field could be used as well. We minimize the loss $\mathcal{L}_c$ combined with the boundary repulsion loss $\mathcal{L}_b$ with weights of 1 and 500, respectively. Qualitative results are shown in the right half of Figure 6.5.

As evaluation metric, we take the absolute angular difference between both the positive and negative prescribed curvature direction at each vertex and the best-aligned edge. We compare with Instant Meshes [130], a method specialized to creating feature-aligned equilateral triangulations. Since Instant Meshes is designed to align to sharp features and is not well defined near flat regions or umbilical points, we weight the per vertex-alignment error using the following term:

$$w_j = \frac{|k_1^j - k_2^j|}{0.5 * (|k_1^j| + |k_2^j|)}, \tag{6.13}$$

where k_1^j and k_2^j are the signed principal curvature magnitudes at vertex j. Intuitively, this term reduces the influence of regions that are nearly flat or umbilical, so as to not penalize the baseline in those regions unfairly. In Figure 6.5, edge alignment errors are shown as colors while the numbers below each result show the RMSE over all vertices.

Our general-purpose triangulation achieves significantly better alignment, as can be seen by the significantly lower color-coded and average error on all models. While the baseline method of [130] generates triangles that are very close to equilateral, the alignment with the curvature directions suffers, as can be seen on the lower part of the pawn, where none of the edges align well with the curvature directions. Similarly, for the cylindrical hat, our method generates edge-loops "hugging" the cylinder, while Instant Meshes does not present such edge-loops. On more organic models, such as the cat and human, lack of alignment is even more evident, e.g., on the human's brow.

Quantitative evaluation We further evaluate our method on our complete dataset of 140 meshes taken from Thingi10k [240]. The quantitative results in Table 6.1 show the RMSE of the metrics described above over all vertices and all shapes in the dataset. Since the vertices at the boundary of our patches cannot fully be optimized with our approach, we provide errors computed both with and without the vertices at the patch boundaries. In both cases and in both applications, our method approximates the correct triangle sizes and edges directions significantly better than the state-of-the-art methods [157] and [130].

Table 6.1 – **Quantitative results.** We compare both the triangle size and curvature alignment applications to state of the art remeshing methods. For our results, we provide values computed both with and without the boundary triangles.

input mesh	[157]	ours w/o bound.	ours
1.320	0.865	0.499	0.686

triangle size

input mesh	[130]	ours w/o bound.	ours
14.043	11.850	7.919	8.4617

curvature alignment

6.4.2 Optimization

Choice of optimizer. We evaluate the effect of different optimization methods, comparing ADAM, LBFGS, and Simulated Annealing. In Figure 6.7 we show results on a 2D triangulation example where we optimize for both triangle sizes, and alignment to a custom vector field. We run each optimizer for 1000 steps and observe that while LBFGS can achieve better performances on some patches, ADAM produces good results more consistently, and hence we opted to use it in all our experiments. We use Simulated Annealing (SA) with the discrete mesh representation instead of our formulation as SA does not handle gradients. After computing the non differentiable weighted Delaunay triangulation, we minimize the discrete version of our losses: for instance we align existing edges to the curvature vector field and fit the area of existing triangles to the target area function. Both gradient-based methods perform significantly better than the non-gradient based method, Simulated Annealing, suggesting that our search space is typically too complex to allow for a more random search strategy that is not guided by gradients. We included the comparison to the non-gradient-based simulated annealing to show gradient-based methods are more apt for this problem; however, putting performance aside, we note that simulated annealing cannot accomplish the main goal of our work, which is to devise a triangulation module that can be used within differentiable optimization frameworks (e.g., PyTorch [180]).

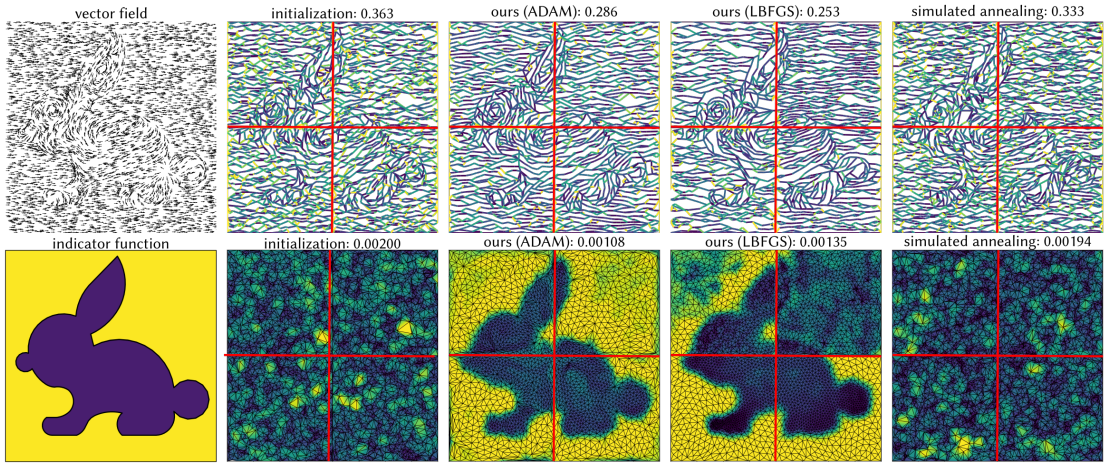

Figure 6.7 – **Comparing different optimizers.** We compare ADAM, LBFGS and Simulated Annealing on a 2D mesh. We start from a 2D mesh with random vertices. In the top row, we optimize edges to align with a given vector field. The best-aligned edges are color-coded according to the alignment error (blue is lowest error, yellow largest error). The average alignment error is shown at the top. In the bottom row, we optimize triangle areas to align with a given size field. Vertices are color-coded according to the average neighboring triangle area (blue are smaller triangles, yellow larger triangles), with RMSE shown at the top. Note how the two gradient-based optimizers ADAM and LBFGS perform significantly better than the gradient-less simulated annealing.

Optimization process. In Figure 6.6 we show the evolution of the triangulation through the optimization steps. The gradual change shows that indeed our differential triangulation enables gradient-based optimization which smoothly decreases the energy towards a local minimum. Please refer to the supplemental video for more detailed visualizations of the optimization process.

6.4.3 Loss blending

As an important advantage, our method naturally enables blending and interpolating the relative weights placed on different loss terms, such as triangle size and adherence to equilateral triangles. We show the plot of energies with respect to such a blending in Figure 6.8 using the aggregated loss term defined as,

$$\mathcal{L}(V', P) := t \times \mathcal{L}_a + (1 - t) \times \mathcal{L}_s \tag{6.14}$$

with t being the blending weight. We evaluate over 7 values of the weight on a subset of 7 models from our dataset. This allows us to easily trade off between characteristics for the triangulation. Note that this was not previously possible for specialized methods targeted towards individual tasks.

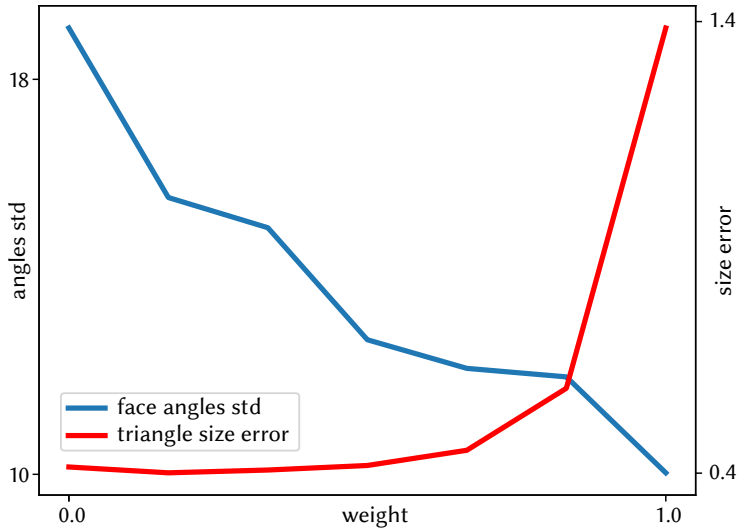

Figure 6.8 – **Loss blending.** We blend the triangle sizing loss $\mathcal{L}_s$ and the equilateral triangle loss $\mathcal{L}_a$ on 7 models of the dataset. We show the error in triangle distribution and the standard deviation of face angles. Note that given a triangulation, the average angle value is 60°. We observe that we can combine the two losses to obtain a trade off between the desired properties.

6.4.4 Method of vertex initialization

We compare our method with an alternative vertex initialization technique in Table 6.2. Given fixed per-patch boundary vertices, we uniformly sample the remaining vertices on the 3D surface using rejection sampling. We evaluate both initialization methods on the same subset of shapes from Section 6.4.3. We observe that the alternative initialization method produces initial triangulations with higher errors. While our method is not completely insensitive to the initialization strategy, it can significantly decrease the loss in both cases.

6.4.5 Runtime and memory

We show the average runtime and maximum memory usage of our method for multiple values of k in Table 6.3 and multiple vertices count per patch in Table 6.4. The runtime is for a typical optimization with 1000 iterations. Both time and memory are linear in the number of vertices and cubic in the number of neighbors k. In our experiments, we typically optimize for 1000 steps for the curvature alignment task and 1500 steps for the triangle size task.

Table 6.2 – **Vertex initialization methods.** We compare an initialization based on remeshed 3D vertices to an initialization based on a uniform distribution over the 3D surface. The uniform initialization has significantly higher initial error, but our method can still decrease the loss significantly.

init. method	input mesh	ours
remeshed model vertices	1.354	0.634
uniform	1.429	0.791

triangle size

init. method	input mesh	ours
remeshed model vertices	13.809	9.037
uniform	19.119	11.028

curvature alignment

Table 6.3 – **Runtime and memory usage w.r.t. k.** We report the average runtime for an optimization with 1K iterations and maximum memory usage per patch.

k	70	80	90
runtime (sec)	132	185	252
memory (GB)	7.6	9.47	13.2

Table 6.4 – **Runtime and memory usage w.r.t. number of vertices per patch.** We report the average runtime for an optimization with 1K iterations and maximum memory usage per patches of varying number of vertices. Time and memory are linear in the number of vertices. Note that we can adjust the size of our patches as needed to avoid memory limitations.

n vertices	500	700	900	1100
runtime (sec)	266	364	477	581
memory (GB)	9.0	12.6	16.3	21.1

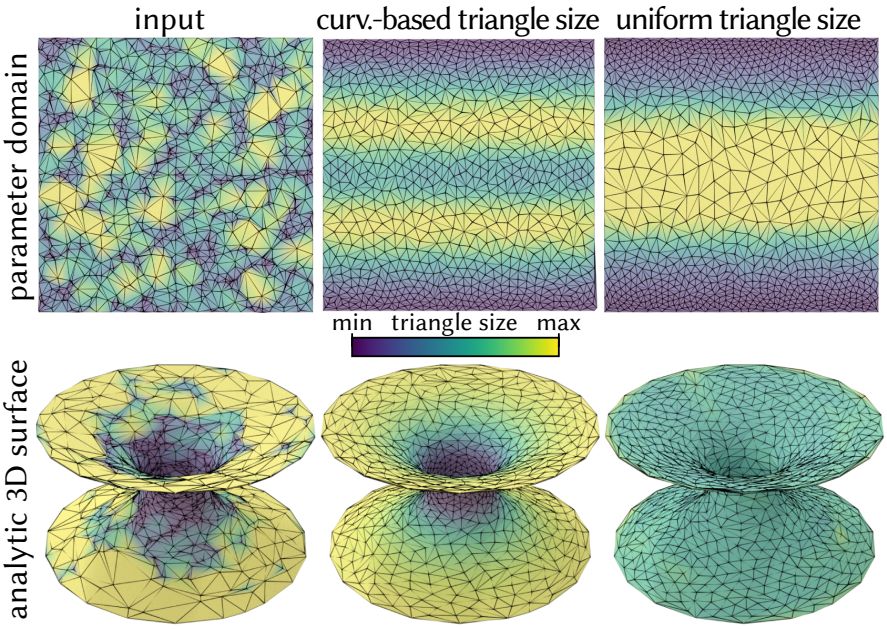

Figure 6.9 – **Analytic surfaces.** Our method can triangulate surfaces given in any representation: here we triangulate an analytic surface (a catenoid), with the parameteric domain shown on the top row, and the 3D surface on the bottom row. Starting from randomly distributed vertices (left), our approach successfully triangulates the analytic surface with curvature-based triangle sizes (middle) and equal triangle sizes (right).

6.4.6 Analytic Surfaces

Our differentiable triangulation method can be applied to any kind of 3D surface, as long as bijective piecewise differentiable parameterization of the surface is available. In Figure 6.9, we experiment with an analytically defined 3D surface, a catenoid [69]. This surface is defined as a function over a 2D parameter domain (thus, the analytical function itself is our mapping m). We start with randomly distributed vertices in the parameter domain and optimize for either triangle sizes based on the curvature magnitude that we compute analytically or for equal-sized triangles in the 3D domain. Curvature values were computed analytically from the surface definition. We can see that our approach successfully optimizes these objectives on the analytic surface.

6.4.7 Discussion on performance

We observe that our method presents an overhead compared to task-specific methods in terms of running time and the size of processed meshes. But with this overhead, our method buys generality and differentiability. We can minimize multiple different objectives (like the objectives of [157], [130]), or can easily combine multiple objectives, without modifying our pipeline and our method can be used as a component in a differentiable framework. Our numerical results are on par with specifically tailored remeshing methods. Note that several recent learning-based methods work with even smaller point counts

(1024 points in PointNet [183], 2250 Edges in MeshCNN [111], 2048 points in [158]), but these restrictions are quickly decreasing with improvements in GPU hardware and methodological improvements.

6.5 Conclusion, Limitations & Future Work

The framework presented in this paper is the first, to the best of our knowledge, to enable approaching surface triangulation from a differentiable point of view. As shown in the experiments, differentiability enables a generic and flexible framework, which can handle various geometric losses, along with their combinations, while taking advantage of modern optimization frameworks. We believe it is the first step towards a black-box, differentiable triangulation module in deep learning frameworks such as PyTorch and TensorFlow where it can be immensely helpful in devising a trainable pipeline, e.g., learning to triangulate models based on deformation sequences.

Our method has two main limitations, the first of which is that the surface needs to be segmented into patches before triangulating. The boundaries of these patches do not participate in the optimization and hence some visible artifacts exist across boundaries. Nevertheless, we note that as shown in the experiments, even with this limitation, our approach achieves significantly better results than the state of the art. A possible solution for this would be to repeat the meshing by iteratively selecting different patches and reparameterizing until convergence.

The second limitation of our method is that it cannot yet handle a large number of points (e.g., 100k+), or large patches, as we need to compute the inclusion scores over a the large space of possible triangles. As future work, we plan to consider a multiscale approach for tackling this issue.

We are excited about the possibilities our approach opens up. For one, since our method can work with surfaces represented in any explicit format (as we show in Figure 6.9), we wish to explore triangulating surfaces in other representations, such as NURBS, or neural representations such as AtlasNet [105, 168]. Extending our method further to point clouds and recovering not only an optimized triangulation but also the topological structure (i.e. the connectivity that defines a surface) could be immensely important for future applications. As an immediate application, we wish to harness differentiabilty to *train* a network to directly output vertex weights and displacements for any given surface in a single forward pass, and thus avoid test-time optimization.

Chapter 7

Conclusion

Shape processing is an important aspect of 3D computer vision. Deep learning methods on 3D data are numerous and can currently handle common problems such as shape segmentation or classification well. However, they do not allow the generation and manipulation of low-level geometric aspects. Therefore they often struggle on tasks that rely on processing the geometric properties of shapes such as point cloud denoising, deformation and meshing. The main contribution of this thesis lies in building systematic deep learning-based approaches that are able to exploit and generate geometric content. We achieved this by building a patch-based pipeline that can retrieve clean data from a noisy point cloud (Chapter 3), therefore recovering the underlying clean surface. We enabled surface retrieval from noisy and incomplete point clouds as well as shape interpolation from point clouds by explicitly learning and exploiting intrinsic information, thus significantly enhancing the typical autoencoder architecture (Chapter 4). Finally we laid the foundations to help bridge the representational gap between triangle meshes and point clouds inside learning based methods (Chapter 5 and 6).

7.1 Discussion and Future work

While we have taken a significant step towards better representations of 3D shapes as well as easy conversion between representations, several challenges still separate us from the goals of seamlessly navigating between representations, creating new 3D content and perfectly handling noisy data.

Point Cloud Denoising Follow-up works on deep denoising tasks further advance state of the art by using unpaired training data [119] or learning self-priors [112]. More recent work have started to rely on non local information [227, 82] as well. To the best of our knowledge, no method combines both local priors with global priors that correspond to higher level of understanding such as symmetries or semantic properties. As a result, denoising methods still tend to produce shapes that lose detailed information. In the future, building methods that rely on both local and global priors will be interesting to more accurately denoise input shapes.

Intrinsic shape representations and deformations In Chapter 4 we successfully combine geometric information with information learned on point clouds. Recent methods for shape deformation are starting to combine geometry processing and deep learning as well. In [17] the authors represent shapes as an atlas while encouraging as isometric as possible deformations. In [229] the authors combine a traditional cage-based deformation

technique, with learned control parameters. However, our method as well as more recent ones are still limited to specific classes of shapes, meshes or templates. A longer term goal would be to design methods that can inject geometric information to shapes from any class or dataset.

Differentiable Meshing While recent works are using a wide variety of representations of shapes [179] and are starting to combine them [175], many aspects of a more direct approach to meshing point clouds such as ours remain under-explored. Generating high quality meshes end-to-end is still an open problem and it would be highly beneficial incorporate differentiable meshing into many applications such as denoising, deformation or shape generation.

Throughout our work, we have produced methods that generalize well to new unseen data. In Chapter 3 and Chapter 5, we observed that learning from patch data helps to build robust methods that can be applied to arbitrary shape classes. Similarly, in Chapter 4 infusing geometric information leads to better generalization despite a small training set. In contrast, methods that limit themselves to specific classes from a large dataset such as ShapeNet [44] benefit from using high level understanding of parts, classes and shapes. An interesting challenge for future work is to exploit the high level understanding power of neural networks while maintaining the generalization power we have observed in this thesis, perhaps by combining geometric information with high-level features.

Finally, we have introduced alternative representations of 3D shapes: in Chapter 3 and 5 we represent shapes as a set of independent local patches that are later combined to produce a smooth surface. In Chapter 4 we represent shapes as a combination of extrinsic and intrinsic learned latent representations. In Chapter 6, we represent meshes as a combination of soft triangles. While we have investigated multiple representations they all present advantages and drawbacks. For instance while patch-based representations allow high generalization, they suffer from compatibility issues between independent neighboring patches. While our dual latent representation from Chapter 4 enables highly realistic interpolations, the applications are limited to datasets of similar classes. Our soft triangle representation in Chapter 6 is subject to costly memory requirements. Finding the right representation for specific tasks or sets of tasks is still an important problem. Addressing this problem would allow to fully exploit the power of deep learning methods on 3D shapes and go beyond the limitations of existing approaches.

Appendix A

Intrinsic Point Cloud Interpolation via Dual Latent Space Navigation: Additional Results

In Section A.1 we provide additional illustrations of our shape interpolation method. In Section A.2 we demonstrate the performance of our approach for *shape reconstruction* highlighting the utility of our dual network for strong regularization of recovering high-quality shapes from noisy point clouds, as mentioned in the main manuscript. In Section A.3 we provide an in-depth ablation study of our network design. In Section A.4 we demonstrate the performance of our approach in the unsupervised case (when the training data is not in correspondence). In section A.5, we develop intuitive connections to Riemannian geometry. Finally, in Section A.6 we provide details of our architecture.

A.1 Shape interpolation

In Figure A.1 we provide an additional qualitative comparison of the linear interpolations in the basic shape (PointNet) AE latent space and the interpolation using our method. Our method preserves body type better (row 2) and interpolates well between a pair of shapes where the end result differs highly from the linear interpolation of the coordinates (row 4).

We further compare our method to other baselines on the SMAL animals dataset. Table A.1 reports the mean-squared variance of several shape features during interpolation of 100 pairs among 50 shapes obtained by farthest points sampling on this dataset. Note that our method produces significantly better quantitative results across all shape features.

	edge length	area (10^{-3})	volume (10^{-2})
PointNet	2.068	3.742	2.754
GD L2	1.906	3.618	2.681
GD EL	1.899	3.585	2.575
3D Coded	9.359	16.922	19.969
Ours	**1.538**	**2.975**	**1.728**

Table A.1 – MS variance of various shape features obtained from interpolating 100 pairs among 50 shapes obtained by farthest points sampling on animals dataset (SMAL)

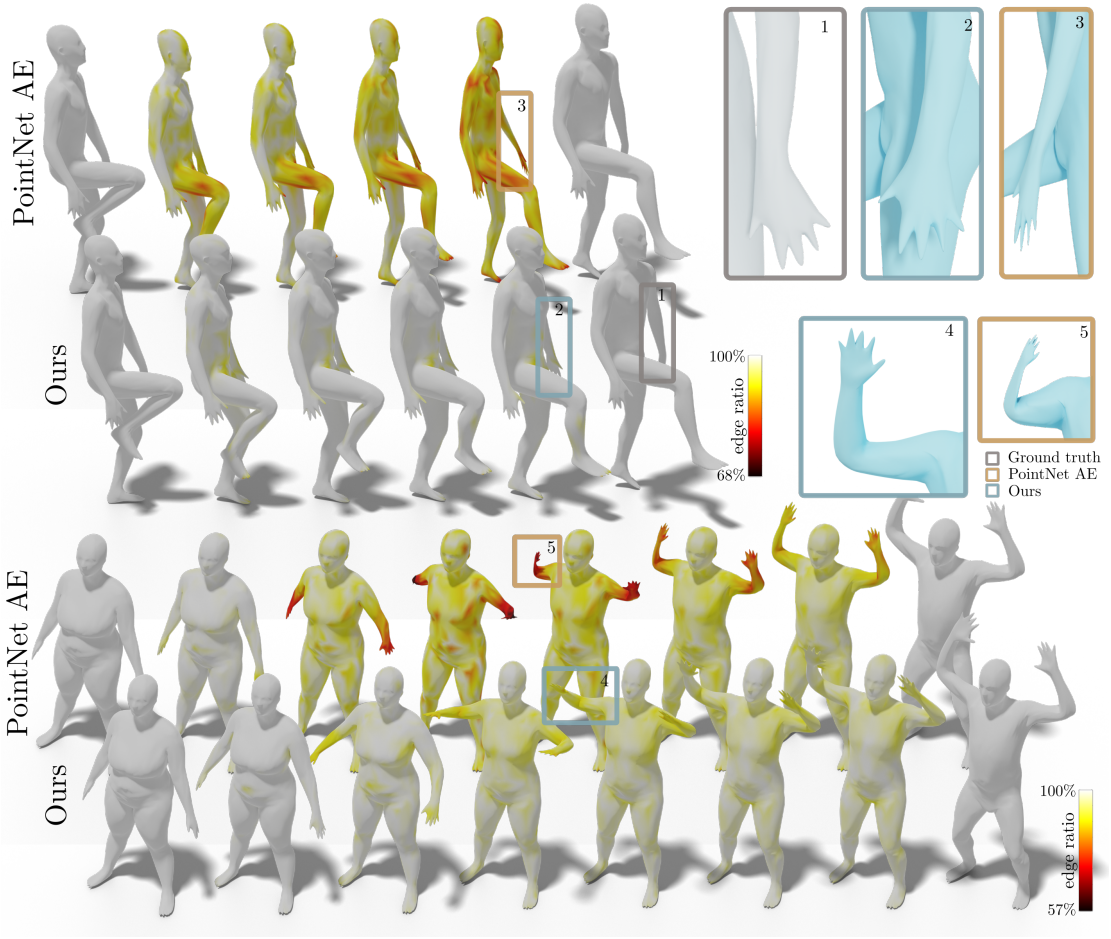

Figure A.1 – We compare linear interpolations in PointNet AE latent space and interpolation using our approach. We visualize the ratio between the linear interpolation of edge lengths and edge lengths of the computed interpolations, to help highlight problematic areas.

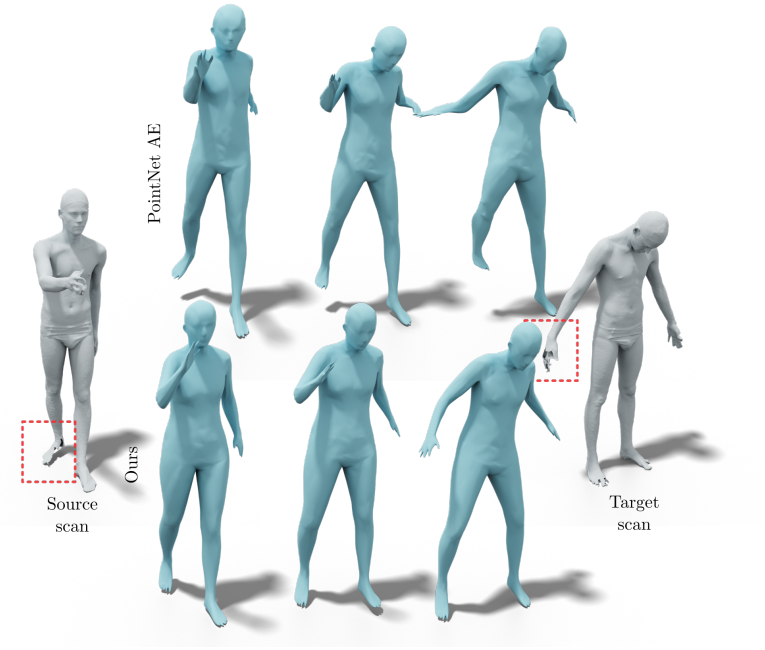

Figure A.2 – We compare linear interpolations in PointNet AE latent space and interpolation using our approach on real scans with artefacts.

We also test our method on *real scans* from the DFAUST dataset [27] in Figure A.2. We observe that our method leads to more realistic results with lower distortion.

A.2 Shape reconstruction

As mentioned in the main manuscript, our approach not only enables better interpolation, but also results in more accurate reconstructions from noisy input. Here we provide additional qualitative and quantitative evaluation of the reconstruction performance and comparison to different baseline methods.

Recall that for our method, given a noisy unordered point cloud P, we reconstruct the shapes by using the following combination of our trained networks $\text{dec}_p(M_{EP}(M_{PE}(\text{enc}_p(P))))$, which differs from the standard auto-encoder approach $\text{dec}_p(\text{enc}_p(P))$. Therefore, in this section we show that the additional regularization provided by our mapping networks M_{EP}, M_{PE} results in better shape reconstruction.

To be fair to 3D-CODED, we normalize the total area of the output shapes. We evaluate this method before (3D-CODED) and after (3D-CODED*) their additional step of Chamfer Distance minimization. Note that in the case of 3D-CODED* additional optimization *at test time* is required to recompute the latent code that best approximates the input. Our method, on the other hand, performs the reconstruction in one shot.

In all of the experiments the training data is the combination of DFAUST and SURREAL datasets, and the test data is the DFAUST test shapes, both with and without

Figure A.3 – Reconstruction of meshes from point clouds containing 1000 points, sampled from the underlying shape.

noise.

Table A.2 shows reconstruction results for several baselines on the 800 DFAUST test shapes. We report the edge length accuracy (EL), rotation-invariant point cloud reconstruction accuracy (PC) and per triangle area reconstruction accuracy (area). Note that our approach achieves the best overall reconstruction accuracy, especially on the intrinsic quantities and gives slightly worse reconstruction extrinsic loss (PC) compared to PointNet AE. We provide qualitative examples in Figure A.3. Note that our method leads to both preservation of the overall shape structure and significantly less intrinsic distortion compared to all baselines.

Table A.3 shows reconstruction performance on noisy point clouds. Note that we test using our model which was trained on clean data. Each noisy point cloud is obtained by adding Gaussian noise magnitude 5% of the scale of the mesh to each vertex coordinate. We observe that our method outperforms the other baselines for all the features. Figure A.4 shows reconstructed meshes from the noisy point clouds. Notice that our method performs better at recovering the original pose and body type than the different baselines.

Table A.4 shows reconstruction results on simplified point clouds. We randomly sample 500 points from the test shapes surfaces. We recall that the network was trained on 1000 point clouds. We observe that our method is more robust to under-sampling. In particular, and contrary to other methods, the intrinsic properties remain competitive with the performance from Table A.2.

We also demonstrate the generalization power across different datasets by showing

	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
PointNet AE	3.023	**2.120**	2.454
Edge Length AE	3.127	-	-
Ours $L_{1,2,3}$	**1.641**	2.572	**1.562**
3D-CODED	6.323	5.803	5.485
3D-CODED*	6.284	4.260	5.409
PointNet++	2.835	3.224	2.835

Table A.2 – Mean squared reconstruction losses on DFAUST testset. Edge length reconstruction loss (EL), Point cloud coordinates reconstruction loss (PC) and per triangle area difference

	Noisy dataset		
	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
PointNet AE	5.663	8.538	5.650
Ours	**3.016**	**7.329**	**2.812**
3D-CODED	8.553	10.463	7.058
PointNet++	26.837	81.379	18.23

Table A.3 – Mean squared reconstruction losses on the DFAUST testset with noise. We use 5% of the shape bounding box gaussian noise on the testset. We recall that the network was trained on 1000 point clouds. We show the edge length reconstruction loss (EL), the rotation invariant reconstruction loss (PC) and the per triangle area difference

	Undersampled dataset		
	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
PointNet AE	3.847	**3.313**	2.810
Ours	**1.854**	3.587	**1.685**
3D-CODED	6.219	6.898	5.341
PointNet++	36.223	117.824	27.541

Table A.4 – Mean squared reconstruction losses on the DFAUST testset undersampled. We randomly sample 500 points from the test shapes surfaces. We recall that the network was trained on 1000 point clouds. We show the edge length reconstruction loss (EL), the rotation invariant reconstruction loss (PC) and the per triangle area difference

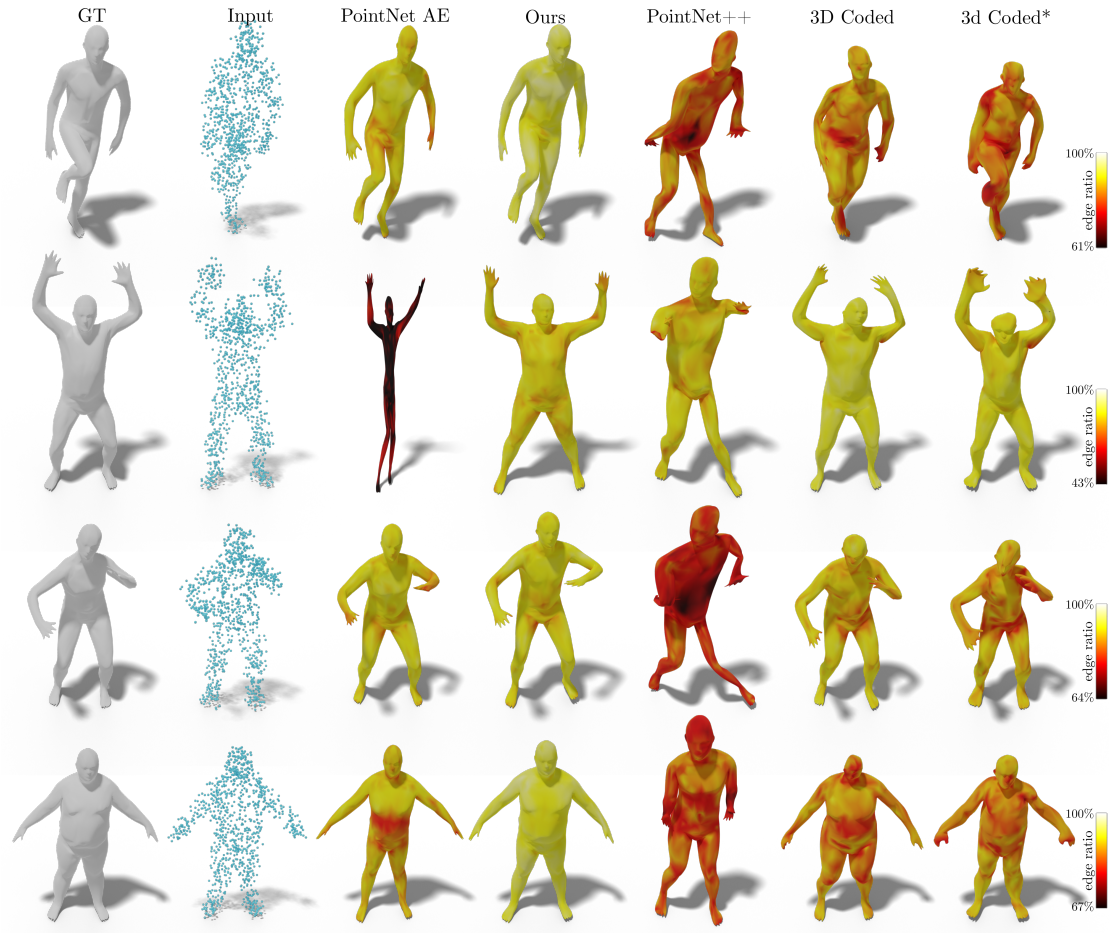

Figure A.4 – Reconstructions from point clouds with 5% of the shape scale gaussian noise.

in Figure A.6 examples of reconstructions from SCAPE dataset [9]. While the simple PointNet AE, is still able to reconstruct the overall position of the tested human, the output has distortions near the hands (left) and the legs (right). Our method generates more natural meshes even though the dataset is completely unknown with an entirely different underlying mesh, different body type and poses that are different to those seen at training. Note that we do not display the color coding as we do not have access to ground truth edge lengths.

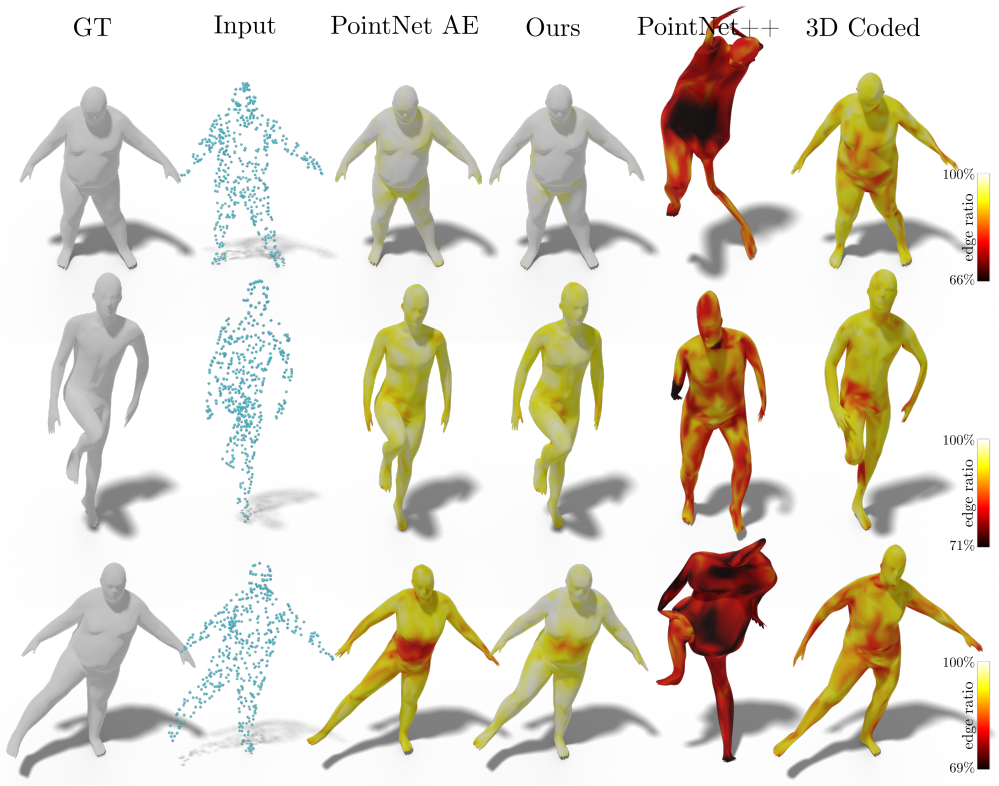

Figure A.5 – We reconstruct a mesh from 500 points sub-sampled randomly from the ground truth mesh. We use a network pre-trained on inputs of size 1000 points.

A.3 Ablation study

A.3.1 Architecture design

Importance of multiple separate networks We first test the utility of having separate networks, rather than training a single network with a combined loss. Specifically, in our study, we have observed that introducing intrinsic information directly during the training of the shape auto-encoder produces unrealistic results with significant artefacts. (Fig. A.7) We train two point-cloud AE (auto-encoders) using: a combination of edge (L_e) and point coordinate (L_{rec}) losses and edge (L_e), point coordinate (L_{rec}) and linearity losses (L_{lin})

Effect of separate networks training In our experiments, we fix the weights of the shape AE and edge auto-encoder during the training of the mapping networks. By doing so, we fix the latent space and generating capabilities of each network. We believe that if this constraint is not respected, the shape AE and edge auto-encoder can be indirectly trained for different losses and generate distortions in the generated shapes. Here, we

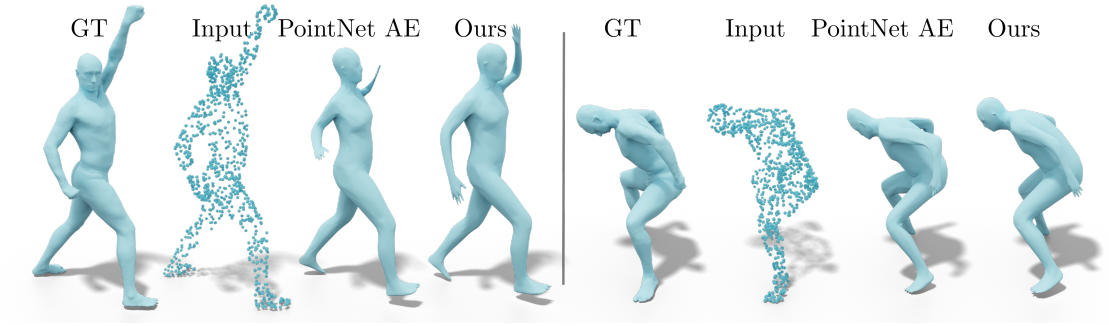

Figure A.6 – Shape reconstruction from SCAPE. We reconstruct from 1k random points on the surface.

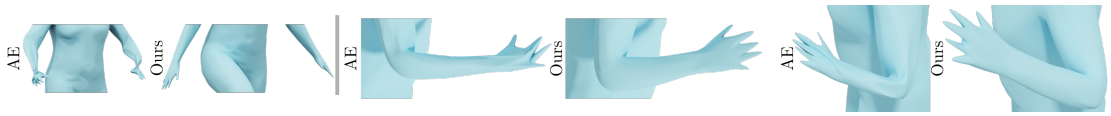

Figure A.7 – Simple AE trained with L_e and L_{rec} (left) or L_e, L_{rec} and L_{lin} (right) produces artifacts during interpolation.

train the mapping networks, edge auto-encoder and shape AE at the same time. To make the training easier, we use a pretrained shape AE and edge auto-encoder. As seen in Table A.5, the reconstruction losses are better than before. However, the shape AE can produce non natural reconstructions during interpolations as shown in Figure A.8. We believe that if the shape AE and edge auto-encoder network were not pretrained, the resulting reconstructed shapes would present even more distortions since the pretrained shape AE can already generate decent natural looking shapes on parts of the dataset.

Auto-encoder vs Variational auto-encoder During our study we compared the performances of our pipeline using either a PointNet AE or a PointNet VAE. The type of network did not result in significant differences. By instance the mean squared variance of the edge length for our architecture trained with a VAE is 0.2301 and 0.2311 when trained with a AE (respectively 0.3760 and 0.3510 for the simple VAE and AE without

	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
Ours	1.666	2.611	1.554
Ours sim. train.	1.027	1.464	1.027

Table A.5 – Mean squared reconstruction losses on the DFAUST testset. We present our main network and an alternative model where all three components are trained simultaneously. Edge length reconstruction loss (EL), Point cloud rotation invariant reconstruction loss (PC) and per triangle area difference (area).

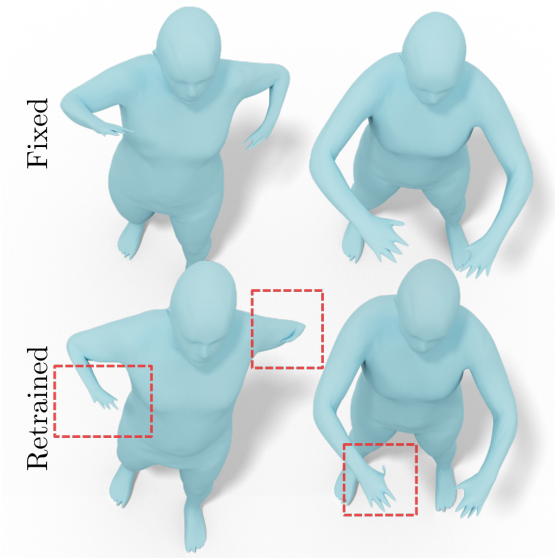

Figure A.8 – Shape distortions are appearing during interpolation if the shape AE, edge auto-encoder and mapping networks are trained at the same time.

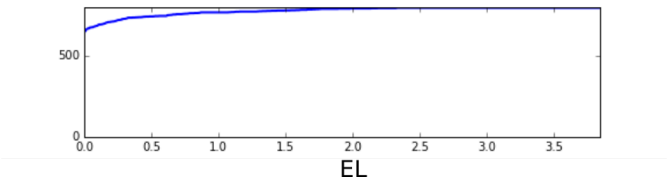

Figure A.9 – Cumulative distribution function of edge reconstruction loss on the DFAUST testset for our network trained without cycle consistency with L_{direct}.

using our pipeline).

A.3.2 Choice of losses

Importance of cycle consistency loss. We train the mapping networks with direct reconstruction losses instead of cycle consistency losses as described in section 4.2 with L_{map1}, L_{map2}, L_{map3} :

$$\begin{aligned} L_{direct}(P, E_P) = \alpha d^{rot}(\text{dec}_p(M_{EP}(\text{enc}_e(E_P))), P) \\ + \beta \|el(\text{dec}_p(M_{EP}(\text{enc}_e(E_P)))) - E_P\|^2 \\ + \|\text{dec}_e(M_{PE}(\text{enc}_p(P))) - E_P\|^2 \end{aligned} \tag{A.1}$$

In Table A.6, we observe that the quality of the map and the quality of the reconstruc-

	EL	PC	area
PointNet AE	$3.023 * 10^{-5}$	$2.120 * 10^{-4}$	$2.454 * 10^{-8}$
Ours	$1.641 * 10^{-5}$	$2.572 * 10^{-4}$	$1.562 * 10^{-8}$
Ours L_{direct}	0.1019	0.6289	$1.338 * 10^{-2}$

Table A.6 – Mean squared reconstruction losses on the DFAUST testset.

	EL (10^{-5})	PC (10^{-4})	area (10^{-8})
Ours $L_{2,3}$	1.595	14.816	1.490
Ours $L_{1,3}$	2.301	2.245	2.113
Ours $L_{1,2,3}$	1.641	2.572	1.562

Table A.7 – Ablation study on different mapping network losses. The subscripts $1, 2, 3$ refer to L_{map1}, L_{map2}, L_{map3} respectively. We show the mean squared reconstruction losses on DFAUST testset. Edge length reconstruction loss (EL), Point cloud coordinates reconstruction loss (PC) and per triangle area difference

tions are worse. In Figure A.9 we show the cumulative distribution function of the edge length reconstruction loss on the testset. While most shapes seem to have reasonable edge reconstruction quality, outlier points make the reconstruction loss explode. Since cycle consistency is not enforced, the network can map shapes onto outliers in the shape space that do not correspond to reasonable natural shapes.

Mapping losses In Table A.7 we show an ablation study of the different losses combinations (described in section 4.2 of the main manuscript) used for training the mapping networks. The subscripts $1, 2, 3$ denote the use of L_{map1}, L_{map2}, L_{map3} respectively. We observe that when trained with L_{map2}, L_{map3}, so only intrinsic features, the model produces better intrinsic reconstruction performances to the expense of the extrinsic reconstruction loss. On the contrary, when trained with only L_{map1} and L_{map3} the network produces good point coordinate reconstruction but worse intrinsic reconstruction performances. To combine the benefits of the different losses, we choose to experiment with a model trained with the 3 losses.

Cycle consistency and direct loss regularization Finally, we combine our cycle consistency loss with direct versions of L_{map1}, L_{map2}, L_{map3} described in equations from A.1. In table A.8, we observe that the models trained with cycle consistency only and cycle consistency with direct losses produce comparable results.

Linearity regularization term in edge auto-encoder. We train a version of our network without the linearity regularization term L_{lin} described in Eq. (6) of the main manuscript for training the edge auto-encoder. As seen in Table A.9, the interpolations in the latent space of the edge auto-encoder are smoother when the network is trained with

	EL	area (10^{-4})	Volume (10^{-4})
Ours $L_{1,2,3}$	0.231	1.261	0.342
Ours $L_{1,2,3}$ with L_{direct}	0.342	1.315	0.264

Table A.8 – We report the mean squared variance of the edge length (EL), per surface area and total shape volume over the interpolations of 100 shape pairs. We compare our method, and our method trained with extra direct losses.

	EL
Edge AE	0.199
Edge AE no lin. reg.	1.777

Table A.9 – We report the mean squared variance of the edge length (EL) over the interpolation in the edge length AE latent space of 100 shape pairs.

the linearity term. In Table A.10, we observe that this term is also related to smoother interpolations of shapes.

A.4 Interpolation in the unsupervised case

Our method can be adapted to an unsupervised context where the 1-1 correspondences are not provided during training. The training process can be described in 3 steps: We first train a point cloud auto-encoder that takes unordered point clouds and outputs an ordered point clouds where the order corresponds to given template T. Then we train the edge auto-encoder by using the output of the shape auto-encoder as training data. Finally, we train the mapping networks as described in the main manuscript.

We first initialize the weights by pre-training the shape AE network to output a chosen template mesh using a variant of the reconstruction loss L_{rec} described in Eq. 4

	EL	area (10^{-4})	volume (10^{-4})
Ours	0.230	1.220	0.385
Ours no lin. reg.	0.245	1.361	0.430

Table A.10 – Interpolation losses for our network where the edge auto-encoder is trained with and without linearity regularization term. We report the mean squared variance of the edge length (EL), per surface area and total shape volume over the interpolations of 100 shape pairs from the DFAUST testset.

	EL	area (10^{-4})	volume (10^{-5})
3D Coded (unsupervised)	0.982	4.140	16.054
PointNet AE (unsupervised)	0.597	3.508	5.251
Ours (unsupervised)	**0.398**	**2.752**	**4.718**

Table A.11 – We report the mean squared variance of the edge length (EL), per surface area and total shape volume over the interpolations of 100 shape pairs. We highlight, while all models produce worse results than their supervised equivalents, our method leads to better interpolations.

of the main manuscript.

$$L_{recInit}(P) = \frac{1}{n}\sum_{i=1}^{n} \|T_i - \tilde{P}_i\|^2, \text{ where } \tilde{P} = \text{dec}_p\left(\text{enc}_p(P)\right). \tag{A.2}$$

Then we train the model using Chamfer Distance (CD) from Eq. (A.3) while encouraging the network to maintain the learned triangulation from step 1 by using regularization terms similar to those used in [104] described bellow.

$$CD(\tilde{P}, P) = \frac{1}{n}\sum_{p_i \in \tilde{P}} \min_{p_j \in P} \|p_i - p_j\|_2^2 + \frac{1}{n}\sum_{p_j \in P} \min_{p_i \in \tilde{P}} \|p_j - p_i\|_2^2 \tag{A.3}$$

$$L_e^{reg}(E_{\tilde{P}}) = \|E_{\tilde{P}} - E_T\|_2^2, \text{where } \tilde{P} = \text{dec}_p(\text{enc}_p(P)) \tag{A.4}$$

$$L_{lap}^{reg}(\tilde{P}) = \|L * (\tilde{P} - T)\|_2^2, \text{where } L \text{ is the graph laplacian} \tag{A.5}$$

We compare our method to unsupervised versions of PointNet AE and 3D Coded. We report numerical evaluation of the interpolations in Table A.11. Note, that our method leads to improved shape features compared to other methods. In Figure A.10, we observe that our method produces more realistic shapes, in particular it produces better arms and heads than PointNet AE and better arms than 3D Coded.

A.5 Geodesics in non flat domains

As mentioned in the main manuscript the two auto-encoders of our architecture can be interpreted as parametrizing the space of realistic shapes and endowing this space with metric (distance) structure. Specifically, the shape auto-encoder aims to recover realistic 3D shapes, and we can *compare* shapes by computing the Euclidean distance between their associated latent vectors in the edge-length auto encoder. Below we explore the relation between linear interpolation and geodesic paths on curved surfaces.

First we note that a classical result in differential geometry (a consequence of Gauss's Theorema Egregium) states that it is impossible to parametrize a curved surface using a

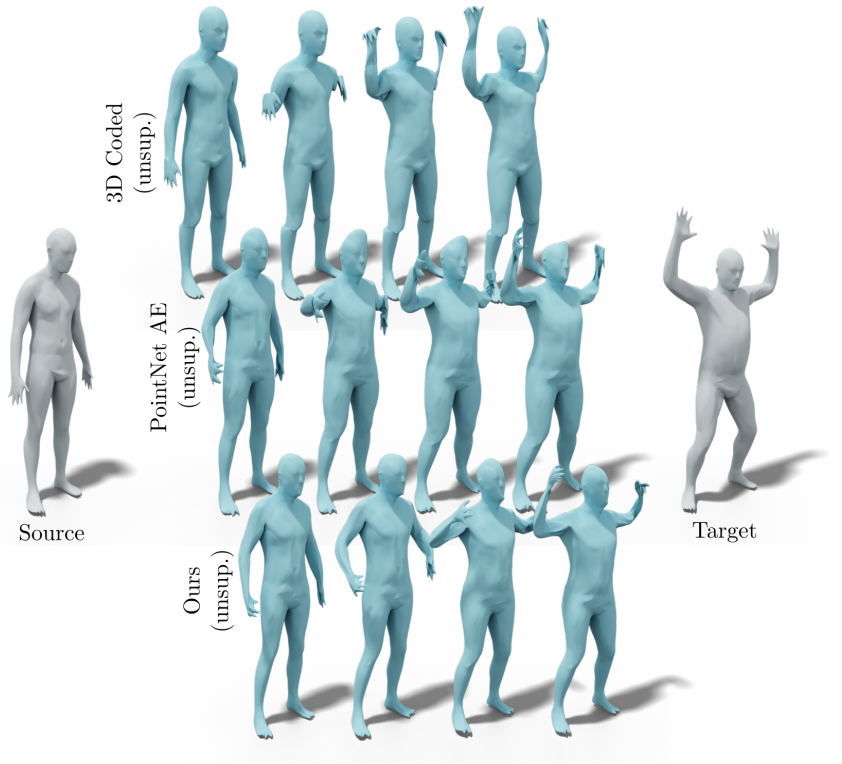

Figure A.10 – Interpolation between shapes when trained with no 1-1 correspondences at train time. Our method produces more realistic shapes.

Euclidean coordinate system while mapping geodesic paths to straight line segments [22] (Chapter 3.1). This directly implies (up to mild genericity conditions such as smoothness) that *there does not exist an auto-encoder network* that is both bijective onto some latent space and allows to recover geodesics through linear interpolation of the latent vectors. Said differently, linear interpolation in the latent space only allows to recover a *flat metric* on the space of shapes, while the intrinsic distortion metric can induce curvature in shape space [115].

Nevertheless, we observe that in certain cases linear interpolation *can* be used to recover geodesic paths even for non-flat domains, if the shape is embedded into a larger space. Specifically, consider the standard sphere S^{n-1} embedded in $\mathbb{R}^n$ and two points $p, q \in S^{n-1}$ that are not polar opposites. Now, construct a line segment linearly interpolating p, q in $\mathbb{R}^n$ and then *project* this line segment onto S^{n-1}. It is clear that the projected segment will recover the geodesic path on the sphere, despite using a linear interpolation in Euclidean space.

This simple example illustrates that if a surface is embedded in a larger space (so that the map from the surface to this space is not a parametrization as it is not invertible) points in that space can be mapped onto the surface through projection. While the projection will necessarily introduce distortion, it can nevertheless help recover geodesic

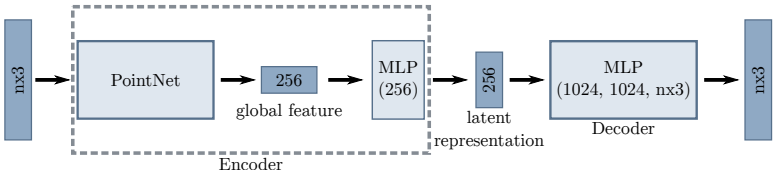

Figure A.11 – Shape AE architecture.

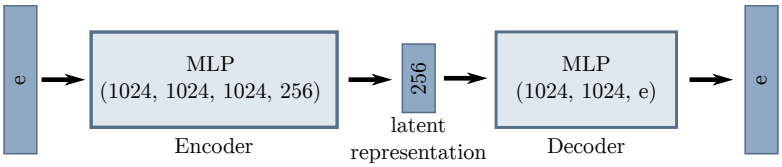

Figure A.12 – Edge length AE architecture.

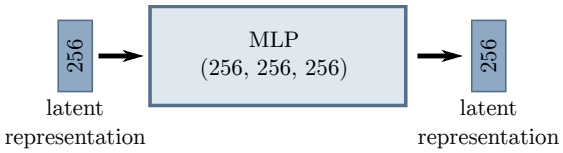

Figure A.13 – Mapping networks architecture.

paths by providing projected points informed by the metric in the embedding space. Although simple and very special, this example points at the interest in studying the relation between the latent spaces of auto-encoders and Riemannian metrics, which we leave as exciting direction for future work.

A.6 Architecture details

We present the detailed architecture of the shape AE, edge length AE and mapping networks in Figure A.11, A.12, A.13.

We implemented the presented architectures using Tensorflow and the Adam optimizer for training.

Appendix B

Learning Delaunay Surface Elements for Mesh Reconstruction: Additional Results

In this document we collect additional details about the proposed method and results that were not included in Chapter 5. We evaluate our method on a dataset with non-uniform sampling in Section B.1. We demonstrate the generalization power of our method in Section B.2 by providing additional quantitative and qualitative results on the ShapeNet dataset. In Section B.3 we illustrate our method with more qualitative results. In Section B.4 we show example shapes from our dataset. We provide additional details on the architecture of our pipeline in Section B.5, show the typical runtime of our method in Section B.6, provide ablation of the learned Logmap component in Section B.7, and show experiments with different neighborhood sizes in Section B.8.

B.1 Non-uniform sampling

We provide additional results on a non-uniformly sampled variant of the FamousThingi dataset. We sample points following a density gradient along the y-axis (horizontal in the figures), where point density correlates with the y-coordinate. A few examples are shown in Figure B.1 (bottom). We did *not* retrain on this dataset variant and evaluate the same model we used for the uniform point clouds. In Table B.1 we show that our method remains robust even with this non-uniform sampling, with only a small decrease in performance compared to uniform sampling. IER meshing takes the largest performance hit with over twice as many non-manifold triangles and significantly increased Chamfer distance. Overall our method shows a similar improvement over the baselines as in uniform sampling. The angle distribution of triangles produced by our method is compared to all baselines in Figure B.2. Our method achieves the best performance with angles more centered around 60 degrees.

We show qualitative comparison in Figure B.3. We observe that ball pivoting and IER meshing are particularly impacted by the non uniform sampling while our method achieves the best quality reconstructions.

B.2 Results on ShapeNet

We compare our method to PointTriNet and IER meshing. Both our method and PointTriNet are trained on the FamousThingi dataset, showing their generalization

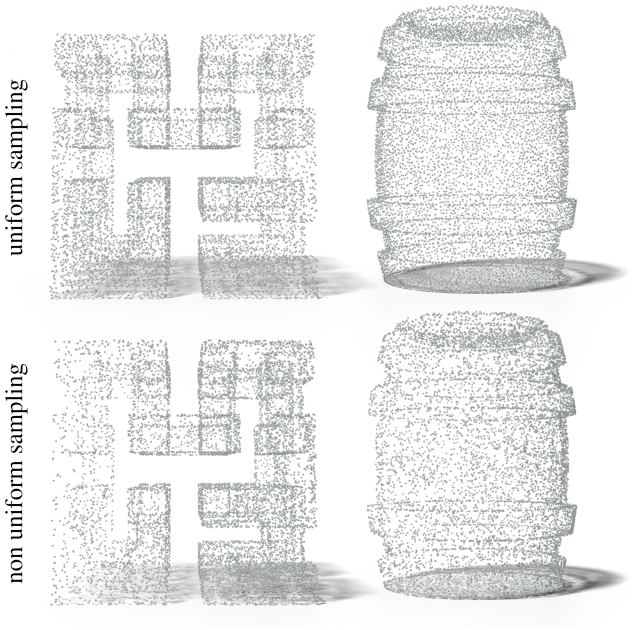

Figure B.1 – Examples of uniformly sampled point clouds (top) and non uniformly sampled point clouds. The density of points follows a gradient along the y axis (horizontal).

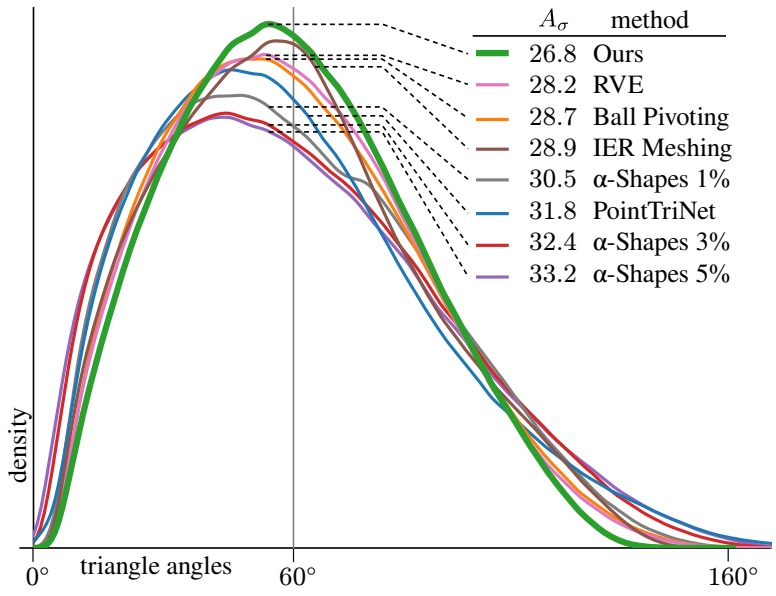

Figure B.2 – Triangle angles distribution. Our method produces triangles with angles more centered around 60 degrees and fewer very obtuse or very acute angles.

Table B.1 – Quantitative comparison the FamousThingi testset where points are sampled non-uniformly. We compare the percentage of non-watertight edges (NW), the Chamfer distance (CD), and the normal reconstruction in degrees (NR) to all baselines.

Method	NW (%)	CD $*1^{e-2}$	NR
Ball pivoting	31.5	0.396	6.84
PointTriNet	14.2	0.383	6.59
IER meshing	13.5	0.487	7.00
RVE	11.0	0.396	9.08
α-shapes 1%	3.5	3.228	63.21
α-shapes 3%	2.7	0.971	28.88
α-shapes 5%	1.7	1.061	17.71
Ours	**1.3**	**0.356**	**6.02**

Table B.2 – Quantitative comparison on 100 random shapes from ShapeNet. We compare our three main metrics to learning-based baselines. IER meshing was specifically trained on ShapeNet, while our method trained on a different dataset (FamousThingi). Even with this handicap, our method obtains better manifoldness and Chamfer distance.

Method	NW (%)	CD $*1^{e-2}$	NR
PointTriNet	22.33	0.416	10.95
IER meshing	6.96	0.456	**6.54**
Ours	**5.51**	**0.396**	9.44

performance, while IER meshing was trained on ShapeNet (since IER meshing requires more shapes for training than the other two methods). Even though this gives IER meshing an advantage, we observe in Table B.2 that our method still produces shapes with better manifoldness and Chamfer distance than other methods.

Additional qualitative results are provided in Figure B.4. We observe that our method produces meshes with better manifoldness and preserves details such as the drawer handles (row 2) or the two sides of the plane wings (row 1) more accurately. Finally, our method produces fewer large holes in the reconstructed mesh.

B.3 More Qualitative Results

We provide additional qualitative results by meshing point clouds of well-known monuments obtained from Famous Paris Buildings. Results are shown in Figure B.5. Since these shapes are geometrically more complex than the shapes in FamousThingi or ShapeNet, we uniformly sample 50k points from each monument. We do not re-train on this dataset. Our method generalizes well to unseen data and denser point clouds.

We also include a real scan reconstruction in Figure B.6. We reconstruct a point cloud with 50k sampled points and compare to other learning-based methods. Since IER meshing can not handle 50k points, we sample 12k points for comaring to IER meshing.

B.4 Dataset Examples

A few examples of shapes from our FAMOUSTHINGI dataset are shown Figure B.7. In Figure B.1, we show two examples of uniformly sampled point clouds we use as input to our method, and two non-uniformly sampled point clouds that we use in the experiments described in Section B.1.

B.5 Architecture Details

We show the detailed architecture of our geodesic patch *classification network* and the 2D log map *projection network* in Figure B.8. The classification network implements a function $c_j := f_\theta([q_j^i, d_j^i] \mid Q^i)$ that classifies if each point q_i in the euclidean patch Q^i is part of the geodesic patch P_i, while the projection network implements a function $u_j^i := g_\phi([p_j^i, d_j^i] \mid P^i)$ that projects points p_j^i in the geodesic patch P_i to their 2D log map coordinates u_j^i. Here d_i is the euclidean distance from a point to its patch center.

Both networks use the same architecture based on FoldingNet [228], except for their output dimension. They take as input a 3D point concatenated with the distance to the patch center and proceed to compute a 1024-dimensional global feature vector for the input patch with a PointNet [183]. Each input point is then augmented with this global feature vector and transformed by two blocks of per-point MLPs into a one-dimensional (classification network) or two-dimensional (projection network) per-point output.

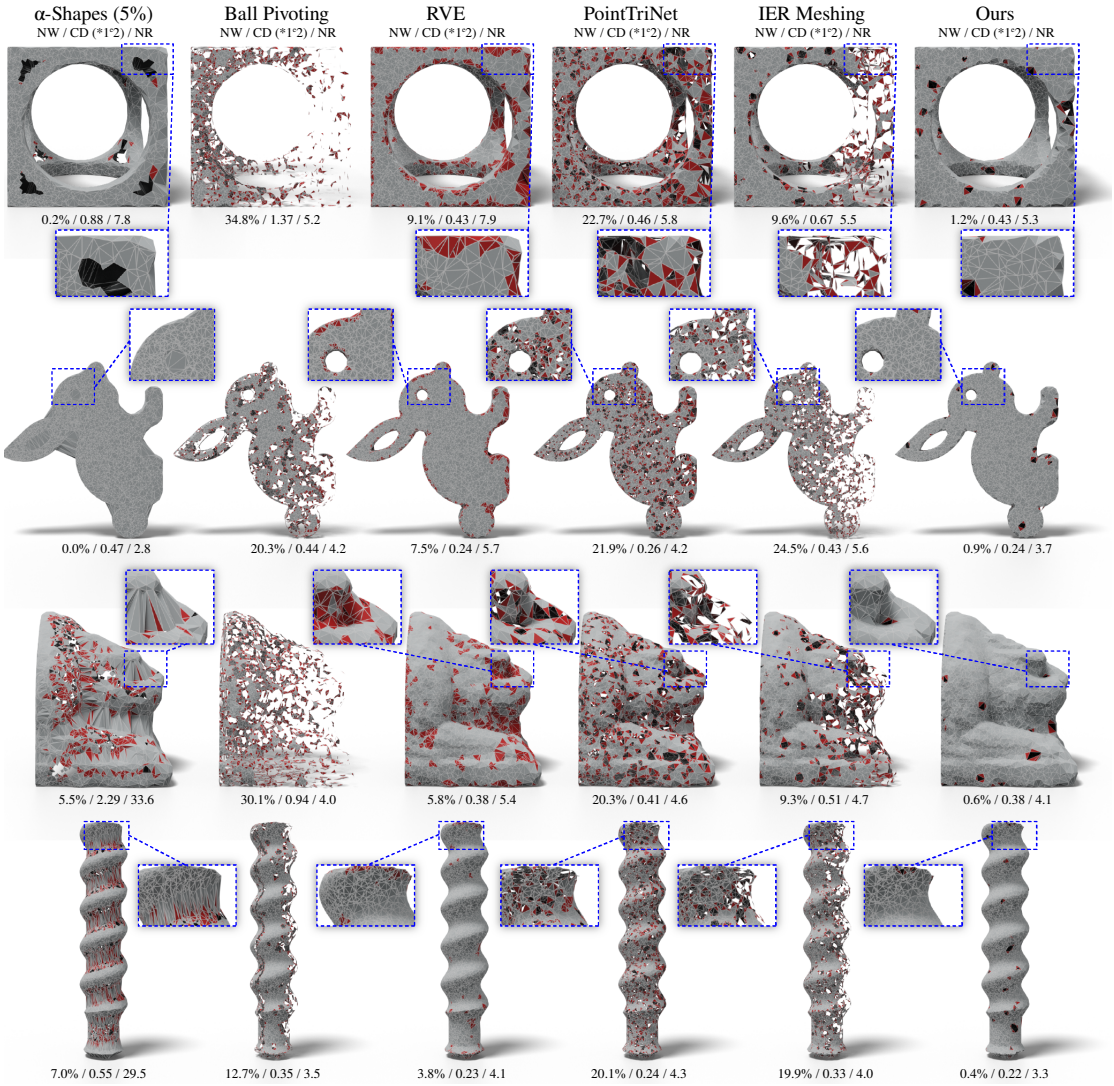

Figure B.3 – Surface reconstructions from non-uniform point clouds. Non-manifold triangles are marked in red. Shapes are sampled more densely to the left and more coarsely to the right. We can see that methods struggle to reconstruct the coarsely sampled parts of the point cloud. While our method also has slightly more errors in the coarsely sampled regions, the mesh quality drops by a much smaller amount from densely to coarsely sampled regions.

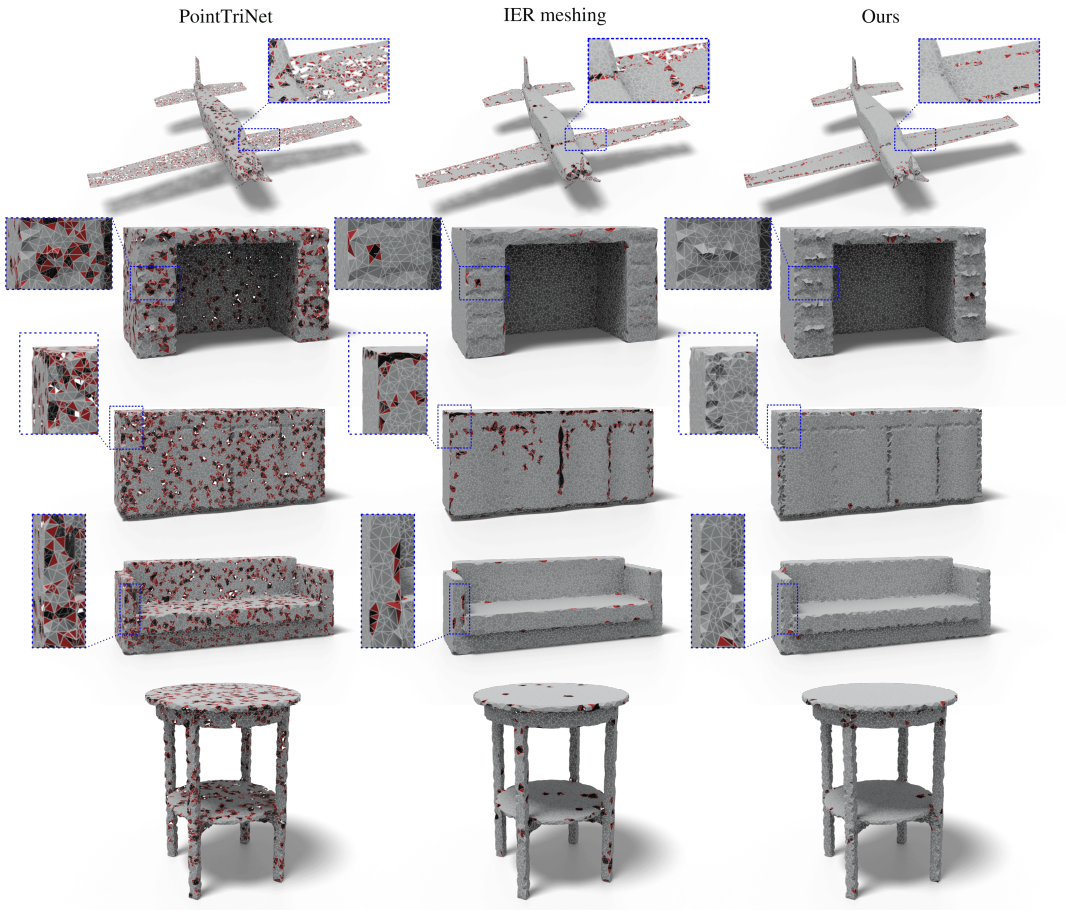

Figure B.4 – Qualitative results on ShapeNet testset. We do not retrain our method on the ShapeNet dataset while IER meshing *was* trained on this dataset. Even so, our method produces more manifold meshes and preserves details such as the drawer handles (row 2) more accurately. We better separate the two sides (top and bottom) of the plane wings (row 1). Finally, our method presents fewer large holes in the reconstructed mesh.

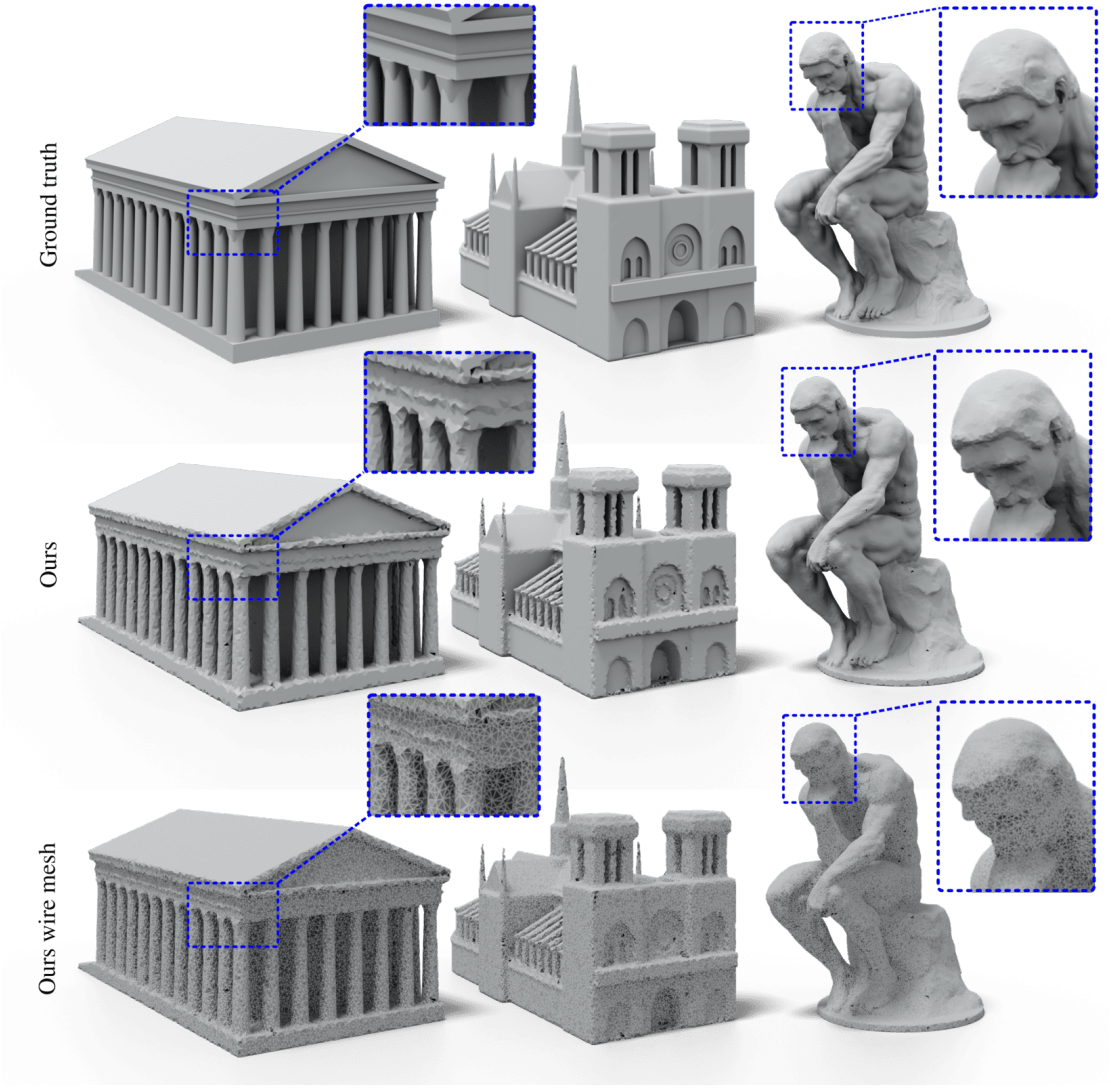

Figure B.5 – Meshing well-known monuments. We show the ground truth (top), the reconstructed mesh (middle), the reconstructed mesh with non manifold triangles colored in red (bottom). Our method generalizes well this more complex data that is also sampled more densely than our training set.

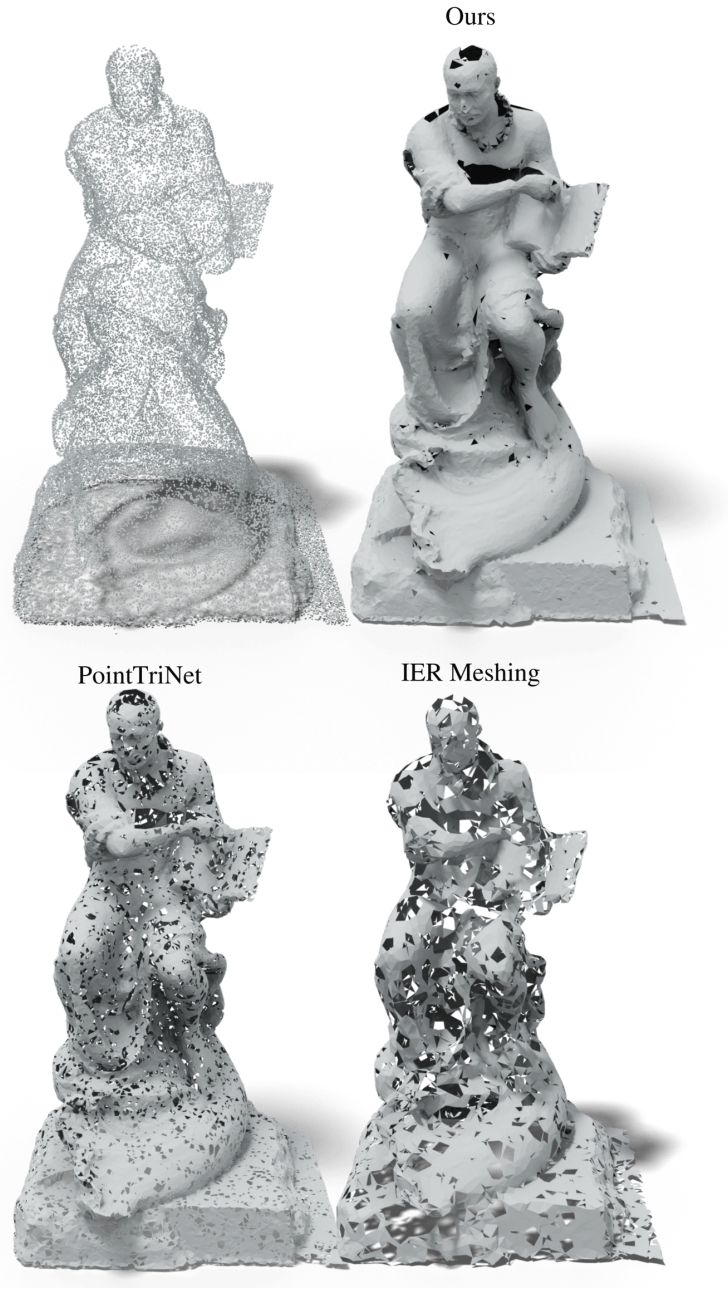

Figure B.6 – Reconstructing real scans from Tanks and Temples [141]

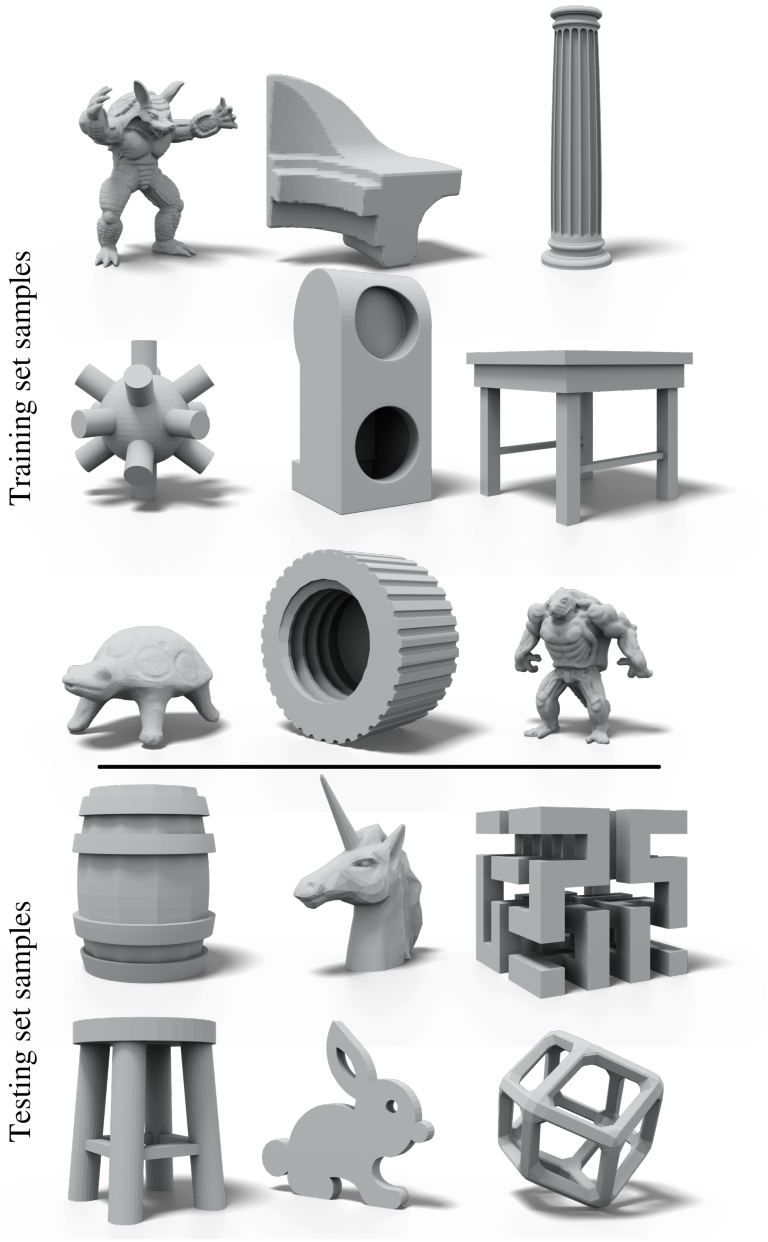

Figure B.7 – Examples from our dataset. We show ground truth meshes from both the training set and the test set.

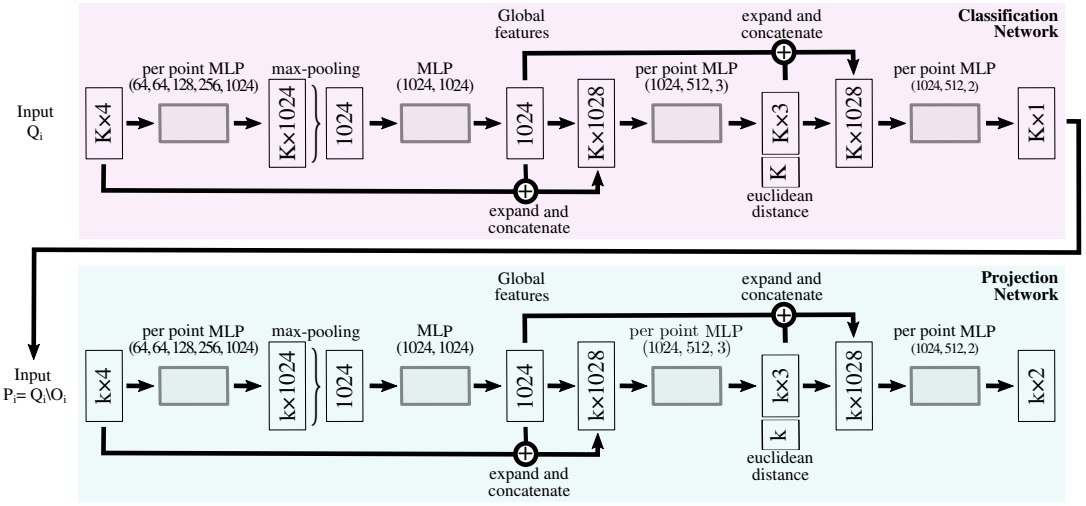

Figure B.8 – Detailed architecture of our pipeline. We first select a small geodesic patch using the classification network (purple). The projection network (blue) then applies a 2D projection to this patch that approximates a log map.

Table B.3 – Average runtime estimation per step on 10k point clouds in seconds.

Log map est.	Log map align.	Selection	Total
5.8	24.8	2.1	32.8

Table B.4 – Quantitative comparison our our learned logmap component to two logmap approximation methods.

	geodesic distance $*1^{e-3}$	2D position $*1^{e-2}$
Projection	1.943	2.627
Rotation	1.943	2.835
Ours	**0.471**	**0.835**

B.6 Runtime

We measure the average runtime of our method on point clouds of 10k vertices in Table B.3, including the runtime for each step of the method.

B.7 Ablation of the Learned Logmap

We compare the performance of our learned logmap component to two baselines. The first baseline approximates the logmap by projecting neighboring points onto the tangent plane computed from the ground truth normal. The second baseline is the approach proposed in [102], where points are rotated onto the tangent plane. Please note the both of these baselines use ground truth normal information, while our method does not. We evaluate the methods on a subset of 33 manifold shapes from our FamousThingi testset and sample 2k patches of k=30 geodesic neighbors per shape. We measure the MSE of the geodesic distance and of the 2D coordinates after patch alignment. Our method produces significantly better logmap estimates compared to other baselines as we show in Table B.4.

B.8 Ablation of Neighbor Counts k and K

We evaluate our method on different values of the geodesic neighbor count k (20, 30, and 50) and different values for the euclidean neighbor count K (80, 120, 160) in Table B.5. For each pair of (k, K) values, we train our models for 30 epochs. The choice of the geodesic neighbor count k affects the performance of our method significantly. If k is small, the Delaunay element approximation quality is affected. If k is large, it is more difficult for the logmap estimation network to produce a usable logmap. Changes in the choice of the euclidean neighbor count K lead to less significant performance drops. In our experiment we choose the parameter values $k = 30$ and $K = 120$ which produce the

Table B.5 – Ablation of different values for the geodesic and euclidean and neighbor counts k and K.

k	K	CD($*1^{e-2}$)	NW(%)	NR
20	80	0.3437	5.569	5.921
20	120	0.3394	4.381	5.845
20	160	0.3496	4.712	6.483
30	80	0.3274	0.509	5.682
30	120	0.3276	**0.485**	**5.661**
30	160	**0.3272**	0.524	5.690
50	80	0.3335	1.822	6.046
50	120	0.3282	0.667	5.856
50	160	0.3286	0.728	5.883

best results for the non-watertightness and normal reconstruction metrics. Please note that for $k = 30$ the difference in Chamfer distance values is negligible.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., *Tensorflow: A system for large-scale machine learning*, in 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 2016, pp. 265–283. (Cited on page 76.)

[2] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, *Learning representations and generative models for 3d point clouds*, in International conference on machine learning, PMLR, 2018, pp. 40–49. (Cited on pages 35 and 49.)

[3] C. C. Aggarwal, *Outlier analysis*, in Data mining, Springer, 2015, pp. 237–263. (Cited on pages 22 and 23.)

[4] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, *Computing and rendering point set surfaces*, IEEE TVCG, 9 (2003), pp. 3–15. (Cited on page 24.)

[5] M. Alexa, D. Cohen-Or, and D. Levin, *As-rigid-as-possible shape interpolation*, in Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 157–164. (Cited on page 48.)

[6] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene, *Recent advances in remeshing of surfaces*, Shape analysis and structuring, (2008), pp. 53–82. (Cited on pages 7, 14 and 77.)

[7] N. Amenta and M. Bern, *Surface reconstruction by voronoi filtering*, Discrete & Computational Geometry, 22 (1999), pp. 481–504. (Cited on page 61.)

[8] N. Amenta, M. Bern, and M. Kamvysselis, *A new voronoi-based surface reconstruction algorithm*, Proc. SIGGRAPH, (1998), pp. 415–421. (Cited on page 61.)

[9] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, *Scape: shape completion and animation of people*, in ACM transactions on graphics (TOG), vol. 24, ACM, 2005, pp. 408–416. (Cited on pages 58 and 104.)

[10] K. S. Arun, T. S. Huang, and S. D. Blostein, *Least-squares fitting of two 3-d point sets*, IEEE Transactions on pattern analysis and machine intelligence, 1 (1987), pp. 698–700. (Cited on page 52.)

[11] F. Aurenhammer, *Power diagrams: properties, algorithms and applications*, SIAM Journal on Computing, 16 (1987), pp. 78–96. (Cited on page 80.)

[12] H. Avron, A. Sharf, C. Greif, and D. Cohen-Or, *l_1-sparse reconstruction of sharp point set surfaces*, ACM TOG, 29 (2010), p. 135. (Cited on page 24.)

[13] A. Badki, O. Gallo, J. Kautz, and P. Sen, *Meshlet priors for 3d mesh reconstruction*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2849–2858. (Cited on page 61.)

[14] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, *Parametric correspondence and chamfer matching: Two new techniques for image matching*, in IJCAI, 1977, pp. 659–663. (Cited on pages 35 and 69.)

[15] M. A. Bautista, W. Talbott, S. Zhai, N. Srivastava, and J. M. Susskind, *On the generalization of learning-based 3d reconstruction*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 2180–2189. (Cited on pages 7 and 13.)

[16] J. Bechtold, M. Tatarchenko, V. Fischer, and T. Brox, *Fostering generalization in single-view 3d reconstruction by learning a hierarchy of local and global shape priors*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 15880–15889. (Cited on pages 7 and 13.)

[17] J. Bednarik, V. G. Kim, S. Chaudhuri, S. Parashar, M. Salzmann, P. Fua, and N. Aigerman, *Temporally-coherent surface reconstruction via metric-consistent atlases*, arXiv preprint arXiv:2104.06950, (2021). (Cited on page 97.)

[18] I. Ben-Gal, *Outlier detection*, in Data mining and knowledge discovery handbook, Springer, 2005, pp. 131–146. (Cited on page 23.)

[19] H. Ben-Hamu, H. Maron, I. Kezurer, G. Avineri, and Y. Lipman, *Multi-chart generative surface modeling*, ACM Transactions on Graphics (TOG), 37 (2018), pp. 1–15. (Cited on pages 9, 15 and 49.)

[20] J.-D. Benamou and Y. Brenier, *A computational fluid mechanics solution to the monge-kantorovich mass transfer problem*, Numerische Mathematik, 84 (2000), pp. 375–393. (Cited on page 49.)

[21] R. Benouini, I. Batioua, K. Zenkouar, S. Najah, and H. Qjidaa, *Efficient 3d object classification by using direct krawtchouk moment invariants*, MTA, (2018), pp. 1–26. (Cited on page 24.)

[22] M. Berger, *A panoramic view of Riemannian geometry*, Springer Science & Business Media, 2012. (Cited on page 111.)

[23] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, *A survey of surface reconstruction from point clouds*, in Computer Graphics Forum, vol. 36, Wiley Online Library, 2017, pp. 301–329. (Cited on pages 59, 61 and 75.)

[24] 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 (1999), pp. 349–359. (Cited on pages 59, 61 and 67.)

[25] K. H. Berthold and P. Horn, *Closed-form solution of absolute orientation using unit quaternions*, Journal of the optical society of America, 4 (1987), pp. 629–642. (Cited on page 64.)

[26] F. Bogo, J. Romero, M. Loper, and M. J. Black, *Faust: Dataset and evaluation for 3d mesh registration*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3794–3801. (Cited on page 49.)

[27] F. Bogo, J. Romero, G. Pons-Moll, and M. J. Black, *Dynamic FAUST: Registering human bodies in motion*, in IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), July 2017. (Cited on pages 49, 54 and 101.)

[28] J.-D. Boissonnat, *Geometric structures for three-dimensional shape representation*, ACM Transactions on Graphics (TOG), 3 (1984), pp. 266–286. (Cited on page 61.)

[29] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec, *Triangulations in cgal*, in Proc. SoCG, 2000, pp. 11–18. (Cited on page 75.)

[30] J.-D. Boissonnat and S. Oudot, *Provably good sampling and meshing of surfaces*, Graphical Models, 67 (2005), pp. 405–451. (Cited on page 61.)

[31] D. Boltcheva and B. Lévy, *Surface reconstruction by computing restricted voronoi cells in parallel*, Computer-Aided Design, 90 (2017), pp. 123–134. (Cited on pages 61, 62 and 67.)

[32] N. Bonneel, J. Rabin, G. Peyré, and H. Pfister, *Sliced and radon wasserstein barycenters of measures*, Journal of Mathematical Imaging and Vision, 51 (2015), pp. 22–45. (Cited on page 49.)

[33] D. Boscaini, D. Eynard, D. Kourounis, and M. M. Bronstein, *Shape-from-operator: Recovering shapes from intrinsic operators*, in Computer Graphics Forum, vol. 34, Wiley Online Library, 2015, pp. 265–274. (Cited on page 53.)

[34] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon mesh processing*, CRC press, 2010. (Cited on pages 7 and 14.)

[35] M. Botsch and O. Sorkine, *On linear variational surface deformation methods*, IEEE transactions on visualization and computer graphics, 14 (2007), pp. 213–230. (Cited on pages 7 and 14.)

[36] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother, *Dsac-differentiable ransac for camera localization*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6684–6692. (Cited on page 80.)

[37] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, *Geometric deep learning: going beyond euclidean data*, IEEE SPM, 34 (2017), pp. 18–42. (Cited on page 22.)

[38] A. Buades, B. Coll, and J.-M. Morel, *A non-local algorithm for image denoising*, in CVPR, vol. 2, IEEE, 2005, pp. 60–65. (Cited on page 24.)

[39] M. P. d. Carmo, *Riemannian geometry*, Birkhäuser, 1992. (Cited on page 53.)

[40] F. Cazals and J. Giesen, *Delaunay triangulation based surface reconstruction: ideas and algorithms*, PhD thesis, INRIA, 2004. (Cited on page 75.)

[41] F. Cazals and M. Pouget, *Estimating differential quantities using polynomial fitting of osculating jets*, CAGD, 22:2 (2005), pp. 121–146. (Cited on pages 22, 24, 36, 37 and 38.)

[42] ——, *Jet_fitting_3: A generic C++ package for estimating the differential properties on sampled surfaces via polynomial fitting*, PhD thesis, INRIA, 2007. (Cited on pages 22, 24 and 37.)

[43] A. Chambolle, V. Caselles, D. Cremers, M. Novaga, and T. Pock, *An introduction to total variation for image analysis*, Theoretical foundations and numerical methods for sparse recovery, 9 (2010), p. 227. (Cited on page 24.)

[44] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al., *Shapenet: An information-rich 3d model repository*, arXiv preprint arXiv:1512.03012, (2015). (Cited on pages 9, 16, 70, 71 and 98.)

[45] P. Chatterjee and P. Milanfar, *Is denoising dead?*, IEEE TIP, 19 (2010), pp. 895–911. (Cited on page 24.)

[46] F. Chazal, D. Cohen-Steiner, and Q. Mérigot, *Geometric inference for probability measures*, Foundations of Computational Mathematics, 11 (2011), pp. 733–751. (Cited on page 23.)

[47] L. Chen and M. Holst, *Efficient mesh optimization schemes based on optimal delaunay triangulations*, Computer Methods in Applied Mechanics and Engineering, 200 (2011), pp. 967–984. (Cited on page 78.)

[48] N. Chen, A. Klushyn, R. Kurle, X. Jiang, J. Bayer, and P. van der Smagt, *Metrics for deep generative models*, arXiv preprint arXiv:1711.01204, (2017). (Cited on pages 49 and 54.)

[49] Z. Chen and H. Zhang, *Learning implicit fields for generative shape modeling*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5939–5948. (Cited on pages 60 and 61.)

[50] S.-W. Cheng, T. Dey, and J. Shewchuk, *Delaunay mesh generation*, 04 2016. (Cited on pages 77 and 80.)

[51] X.-X. Cheng, X.-M. Fu, C. Zhang, and S. Chai, *Practical error-bounded remeshing by adaptive refinement*, Computers & Graphics, 82 (2019), pp. 163–173. (Cited on page 78.)

[52] A. Chern, F. Knöppel, U. Pinkall, and P. Schröder, *Shape from metric*, ACM Transactions on Graphics (TOG), 37 (2018), p. 63. (Cited on page 53.)

[53] 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, vol. 2008, Salerno, Italy, 2008, pp. 129–136. (Cited on page 84.)

[54] E. Corman, J. Solomon, M. Ben-Chen, L. Guibas, and M. Ovsjanikov, *Functional characterization of intrinsic and extrinsic geometry*, ACM Transactions on Graphics (TOG), 36 (2017), pp. 1–17. (Cited on page 53.)

[55] K. Crane, U. Pinkall, and P. Schröder, *Spin transformations of discrete surfaces*, ACM Transactions on Graphics (TOG), 30 (2011), p. 104. (Cited on page 48.)

[56] B. Curless and M. Levoy, *A volumetric method for building complex models from range images*, in Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, Proc. SIGGRAPH, 1996, pp. 303–312. (Cited on page 61.)

[57] A. Dai and M. Niessner, *Scan2mesh: From unstructured range scans to 3d meshes*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5574–5583. (Cited on page 62.)

[58] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2000. (Cited on page 76.)

[59] F. De Goes, K. Breeden, V. Ostromoukhov, and M. Desbrun, *Blue noise through optimal transport*, ACM Transactions on Graphics (TOG), 31 (2012), pp. 1–11. (Cited on page 79.)

[60] F. De Goes, C. Wallez, J. Huang, D. Pavlov, and M. Desbrun, *Power particles: an incompressible fluid solver based on power diagrams.*, ACM Trans. Graph., 34 (2015), pp. 50–1. (Cited on page 79.)

[61] B. Delaunay et al., *Sur la sphere vide*, Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 7 (1934), pp. 1–2. (Cited on pages 62 and 80.)

[62] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*, in 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255. (Cited on pages 9 and 16.)

[63] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, *Implicit fairing of irregular meshes using diffusion and curvature flow*, in Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999, pp. 317–324. (Cited on page 79.)

[64] J.-E. Deschaud and F. Goulette, *Point cloud non local denoising using local surface descriptor similarity*, IAPRS, 38 (2010), pp. 109–114. (Cited on page 24.)

[65] O. Devillers, *The delaunay hierarchy*, International Journal of Foundations of Computer Science, 13 (2002), pp. 163–180. (Cited on page 75.)

[66] O. Devillers, S. Pion, and M. Teillaud, *Walking in a triangulation*, in Proc. SoCG, 2001, pp. 106–114. (Cited on page 75.)

[67] T. K. Dey, *Curve and surface reconstruction: algorithms with mathematical analysis*, vol. 23, Cambridge University Press, 2006. (Cited on page 61.)

[68] T. K. Dey, J. Giesen, and J. Hudson, *Delaunay based shape reconstruction from large data*, in Proceedings IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (Cat. No. 01EX520), IEEE, 2001, pp. 19–146. (Cited on page 61.)

[69] U. Dierkes, S. Hildebrandt, and F. Sauvigny, *Minimal surfaces*, in Minimal Surfaces, Springer, 2010, pp. 53–90. (Cited on page 94.)

[70] J. Digne, *Similarity based filtering of point clouds*, in CVPR Workshops, IEEE, 2012, pp. 73–79. (Cited on pages 22 and 24.)

[71] J. Digne, R. Chaine, and S. Valette, *Self-similarity for accurate compression of point sampled surfaces*, in CGF, vol. 33, Wiley Online Library, 2014, pp. 155–164. (Cited on page 24.)

[72] J. Digne, D. Cohen-Steiner, P. Alliez, F. De Goes, and M. Desbrun, *Feature-preserving surface reconstruction and simplification from defect-laden point sets*, Journal of mathematical imaging and vision, 48 (2014), pp. 369–382. (Cited on page 61.)

[73] J. Digne and C. De Franchis, *The bilateral filter for point clouds*, Image Processing On Line, 7 (2017), pp. 278–287. (Cited on pages 24, 36 and 37.)

[74] J. Digne, S. Valette, and R. Chaine, *Sparse geometric representation through local shape probing*, IEEE TVCG, 24 (2018), pp. 2238–2250. (Cited on page 24.)

[75] M. P. Do Carmo, *Differential geometry of curves and surfaces: revised and updated second edition*, Courier Dover Publications, 2016. (Cited on page 63.)

[76] Q. Du, M. D. Gunzburger, and L. Ju, *Constrained centroidal voronoi tessellations for surfaces*, SIAM Journal on Scientific Computing, 24 (2003), pp. 1488–1506. (Cited on page 79.)

[77] M. Dunyach, D. Vanderhaeghe, L. Barthe, and M. Botsch, *Adaptive remeshing for real-time mesh deformation*, in Eurographics 2013, The Eurographics Association, 2013. (Cited on pages 78 and 79.)

[78] R. Dyer, H. Zhang, and T. Möller, *Delaunay mesh construction*, in Proceedings of the fifth Eurographics symposium on Geometry processing, 2007, pp. 273–282. (Cited on page 78.)

[79] H. Edelsbrunner and E. P. Mücke, *Three-dimensional alpha shapes*, ACM Transactions on Graphics (TOG), 13 (1994), pp. 43–72. (Cited on pages 59, 61 and 67.)

[80] M. Elad, *From exact to approximate solutions*, in Sparse and Redundant Representations, Springer, 2010, pp. 79–109. (Cited on page 24.)

[81] M. Elad and M. Aharon, *Image denoising via sparse and redundant representations over learned dictionaries*, IEEE TIP, 15 (2006), pp. 3736–3745. (Cited on page 24.)

[82] P. Erler, P. Guerrero, S. Ohrhallinger, N. J. Mitra, and M. Wimmer, *Points2surf learning implicit surfaces from point clouds*, in European Conference on Computer Vision, Springer, 2020, pp. 108–124. (Cited on page 97.)

[83] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., *A density-based algorithm for discovering clusters in large spatial databases with noise.*, in Kdd, vol. 96, 1996, pp. 226–231. (Cited on page 65.)

[84] H. Fan, H. Su, and L. J. Guibas, *A point set generation network for 3d object reconstruction from a single image.*, in CVPR, vol. 2, 2017, p. 6. (Cited on pages 35 and 69.)

[85] M. Fisher, B. Springborn, P. Schröder, and A. I. Bobenko, *An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing*, Computing, 81 (2007), pp. 199–213. (Cited on page 78.)

[86] S. Fleishman, D. Cohen-Or, and C. T. Silva, *Robust moving least-squares fitting with sharp features*, ACM TOG, 24 (2005), pp. 544–552. (Cited on pages 22 and 24.)

[87] O. Freifeld and M. J. Black, *Lie bodies: A manifold representation of 3d human shape*, in European Conference on Computer Vision, Springer, 2012, pp. 1–14. (Cited on page 49.)

[88] M. F. Frenzel, B. Teleaga, and A. Ushio, *Latent space cartography: Generalised metric-inspired measures and measure-based transformations for generative models*, arXiv preprint arXiv:1902.02113, (2019). (Cited on page 49.)

[89] X.-M. Fu, Y. Liu, J. Snyder, and B. Guo, *Anisotropic simplicial meshing using local convex functions*, ACM Trans. Graph., 33 (2014). (Cited on page 79.)

[90] Y. Fu and B. Zhou, *Direct sampling on surfaces for high quality remeshing*, Computer Aided Geometric Design, 26 (2009), pp. 711 – 723. Solid and Physical Modeling 2008. (Cited on page 79.)

[91] L. Gao, S.-Y. Chen, Y.-K. Lai, and S. Xia, *Data-driven shape interpolation and morphing editing*, Computer Graphics Forum, 36 (2017), pp. 19–31. (Cited on page 49.)

[92] L. Gao, Y.-K. Lai, Q.-X. Huang, and S.-M. Hu, *A data-driven approach to realistic shape morphing*, Computer graphics forum, 32 (2013), pp. 449–457. (Cited on page 49.)

[93] M. Garland and P. S. Heckbert, *Surface simplification using quadric error metrics*, in Proceedings of the 24th annual conference on Computer graphics and interactive techniques, 1997, pp. 209–216. (Cited on page 77.)

[94] K. Genova, F. Cole, D. Vlasic, A. Sarna, W. T. Freeman, and T. Funkhouser, *Learning shape templates with structured implicit functions*, arXiv preprint arXiv:1904.06447, (2019). (Cited on pages 60 and 61.)

[95] S. Giraudot, D. Cohen-Steiner, and P. Alliez, *Noise-adaptive shape reconstruction from raw point sets*, in SGP, Eurographics Association, 2013, pp. 229–238. (Cited on page 23.)

[96] R. Girshick, *Fast r-cnn*, in CVPR, 2015. (Cited on pages 7 and 13.)

[97] G. Gkioxari, J. Malik, and J. Johnson, *Mesh r-cnn*, arXiv preprint arXiv:1906.02739, (2019). (Cited on page 61.)

[98] D. Glickenstein, *Geometric triangulations and discrete laplacians on manifolds*, arXiv preprint math/0508188, (2005). (Cited on page 79.)

[99] H. Gluck, *Almost all simply connected closed surfaces are rigid*, in Geometric topology, Springer, 1975, pp. 225–239. (Cited on pages 52 and 53.)

[100] F. d. Goes, P. Memari, P. Mullen, and M. Desbrun, *Weighted triangulations for geometry processing*, ACM Transactions on Graphics (TOG), 33 (2014), pp. 1–13. (Cited on page 79.)

[101] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in NeurIPS, 2014. (Cited on pages 7 and 13.)

[102] M. Gopi, S. Krishnan, and C. T. Silva, *Surface reconstruction based on lower dimensional localized delaunay triangulation*, in Computer Graphics Forum, vol. 19, Wiley Online Library, 2000, pp. 467–478. (Cited on pages 61 and 123.)

[103] M. Gross and H. Pfister, *Point-based graphics*, Elsevier, 2011. (Cited on page 24.)

[104] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, *3d-coded: 3d correspondences by deep deformation*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 230–246. (Cited on pages 9, 15, 49, 54 and 110.)

[105] ——, *A papier-mâché approach to learning 3d surface generation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 216–224. (Cited on pages 61 and 95.)

[106] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree, *Blensor: blender sensor simulation toolbox*, in ISVC, Springer, 2011, pp. 199–208. (Cited on page 40.)

[107] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, *PCPNet: Learning local shape properties from raw point clouds*, CGF, 37 (2018), pp. 75–85. (Cited on pages 22, 24, 25, 26, 27, 28, 33, 38 and 67.)

[108] L. Guibas, D. Morozov, and Q. Mérigot, *Witnessed k-distance*, Discrete & Computational Geometry, 49 (2013), pp. 22–45. (Cited on page 23.)

[109] J. Han, L. Shao, D. Xu, and J. Shotton, *Enhanced computer vision with microsoft kinect sensor: A review*, IEEE transactions on cybernetics, 43 (2013), pp. 1318–1334. (Cited on pages 8 and 14.)

[110] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, *A review of algorithms for filtering the 3d point cloud*, Signal Processing: Image Communication, 57 (2017), pp. 103–112. (Cited on pages 8, 14, 22, 23 and 24.)

[111] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, *Meshcnn: a network with an edge*, ACM Transactions on Graphics (TOG), 38 (2019), pp. 1–12. (Cited on pages 9, 15 and 95.)

[112] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, *Point2mesh: A self-prior for deformable meshes*, arXiv preprint arXiv:2005.11084, (2020). (Cited on page 97.)

[113] N. Hasler, C. Stoll, M. Sunkel, B. Rosenhahn, and H.-P. Seidel, *A statistical model of human pose and body shape*, Computer graphics forum, 28 (2009), pp. 337–346. (Cited on page 49.)

[114] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in ICCV, 2015, pp. 1026–1034. (Cited on page 33.)

[115] B. Heeren, M. Rumpf, P. Schröder, M. Wardetzky, and B. Wirth, *Exploring the geometry of the space of shells*, in Computer Graphics Forum, vol. 33, Wiley Online Library, 2014, pp. 247–256. (Cited on pages 49 and 111.)

[116] ——, *Splines in the space of shells*, Computer Graphics Forum, 35 (2016), pp. 111–120. (Cited on page 49.)

[117] B. Heeren, M. Rumpf, M. Wardetzky, and B. Wirth, *Time-discrete geodesics in the space of shells*, in Computer Graphics Forum, vol. 31, Wiley Online Library, 2012, pp. 1755–1764. (Cited on pages 9, 15 and 49.)

[118] P. Herholz and M. Alexa, *Efficient computation of smoothed exponential maps*, in Computer Graphics Forum, vol. 38, Wiley Online Library, 2019, pp. 79–90. (Cited on page 63.)

[119] P. Hermosilla, T. Ritschel, and T. Ropinski, *Total denoising: Unsupervised learning of 3d point cloud cleaning*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 52–60. (Cited on page 97.)

[120] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, vol. 26, ACM, 1992. (Cited on page 61.)

[121] ——, *Mesh optimization*, in Proceedings of the 20th annual conference on Computer graphics and interactive techniques, 1993, pp. 19–26. (Cited on pages 75 and 79.)

[122] K. Hu, D.-M. Yan, D. Bommes, P. Alliez, and B. Benes, *Error-bounded and feature preserving surface remeshing with minimal angle improvement*, IEEE transactions on visualization and computer graphics, 23 (2016), pp. 2560–2573. (Cited on page 78.)

[123] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, *Edge-aware point set resampling*, ACM TOG, 32 (2013), p. 9. (Cited on pages 22, 24, 36, 37 and 38.)

[124] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum, *Subspace gradient domain mesh deformation*, ACM Transactions on Graphics (TOG), 25 (2006), pp. 1126–1134. (Cited on page 48.)

[125] R. Huang*, M.-J. Rakotosaona*, P. Achlioptas, L. J. Guibas, and M. Ovsjanikov, *OperatorNet: Recovering 3d shapes from difference operators*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 8588–8597. (Cited on pages 7, 9, 12, 13, 15, 19, 47, 49, 52 and 54.)

[126] M. Hussain, *Efficient simplification methods for generating high quality lods of 3d meshes*, Journal of Computer Science and Technology, 24 (2009), pp. 604–604. (Cited on page 77.)

[127] T. Igarashi, T. Moscovich, and J. F. Hughes, *As-rigid-as-possible shape manipulation*, ACM transactions on Graphics (TOG), 24 (2005), pp. 1134–1141. (Cited on page 48.)

[128] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-image translation with conditional adversarial networks*, in CVPR, 2017. (Cited on pages 7 and 13.)

[129] M. Jaderberg, K. Simonyan, A. Zisserman, et al., *Spatial transformer networks*, in NIPS, 2015, pp. 2017–2025. (Cited on pages 27 and 28.)

[130] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung, *Instant field-aligned meshes*, ACM Transactions on Graphics, 34 (2015), p. 189. (Cited on pages 78, 88, 89, 90 and 94.)

[131] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, *Deep convolutional neural network for inverse problems in imaging*, IEEE TIP, 26 (2017), pp. 4509–4522. (Cited on page 24.)

[132] T. Ju, F. Losasso, S. Schaefer, and J. Warren, *Dual contouring of hermite data*, in Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002, pp. 339–346. (Cited on page 61.)

[133] M. Kazhdan, M. Bolitho, and H. Hoppe, *Poisson surface reconstruction*, in Proceedings of the fourth Eurographics symposium on Geometry processing, vol. 7, 2006. (Cited on pages 59 and 61.)

[134] M. Kazhdan and H. Hoppe, *Screened poisson surface reconstruction*, ACM TOG, 32 (2013), pp. 29:1–29:13. (Cited on page 41.)

[135] D. G. Kendall, *Shape manifolds, procrustean metrics, and complex projective spaces*, Bulletin of the London Mathematical Society, 16 (1984), pp. 81–121. (Cited on page 48.)

[136] D. Khan, A. Plopski, Y. Fujimoto, M. Kanbara, G. Jabeen, Y. Zhang, X. Zhang, and H. Kato, *Surface remeshing: A systematic literature review of methods and research directions*, IEEE Transactions on Visualization and Computer Graphics, (2020). (Cited on pages 77 and 79.)

[137] A. Khatamian and H. Arabnia, *Survey on 3d surface reconstruction*, Journal of Information Processing Systems, 12 (2016), pp. 338–357. (Cited on pages 61 and 77.)

[138] M. Kilian, N. J. Mitra, and H. Pottmann, *Geometric modeling in shape space*, in ACM SIGGRAPH 2007 papers, 2007, pp. 64–es. (Cited on pages 9, 15, 49, 53, 54 and 55.)

[139] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015. (Cited on page 87.)

[140] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv preprint, (2013). (Cited on pages 7 and 13.)

[141] A. Knapitsch et al., *Tanks and temples: Benchmarking large-scale scene reconstruction*, ACM TOG, 36 (2017). (Cited on page 120.)

[142] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, *Spectral surface reconstruction from noisy point clouds*, in Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, ACM, 2004, pp. 11–21. (Cited on page 61.)

[143] S. Laine, *Feature-based metrics for exploring the latent space of generative models*, (2018). (Cited on page 49.)

[144] F. Lazarus and A. Verroust, *Three-dimensional metamorphosis: a survey*, The Visual Computer, 14 (1998), pp. 373–389. (Cited on page 48.)

[145] T. Lescoat, H.-T. D. Liu, J.-M. Thiery, A. Jacobson, T. Boubekeur, and M. Ovsjanikov, *Spectral mesh simplification*, in Computer Graphics Forum, vol. 39, Wiley Online Library, 2020, pp. 315–324. (Cited on page 77.)

[146] T. Lescoat, M. Ovsjanikov, P. Memari, J.-M. Thiery, and T. Boubekeur, *A survey on data-driven dictionary-based methods for 3d modeling*, in CGF, vol. 37, 2018, pp. 577–601. (Cited on page 24.)

[147] A. Levin and B. Nadler, *Natural image denoising: Optimality and inherent bounds*, in CVPR, 2011, IEEE, 2011, pp. 2833–2840. (Cited on page 24.)

[148] B. Lévy and N. Bonneel, *Variational anisotropic surface meshing with voronoi parallel linear enumeration*, in Proceedings of the 21st international meshing roundtable, Springer, 2013, pp. 349–366. (Cited on page 79.)

[149] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, *Least squares conformal maps for automatic texture atlas generation*, ACM Trans. Graph., 21 (2002), p. 362–371. (Cited on page 84.)

[150] C.-L. Li, M. Zaheer, Y. Zhang, B. Poczos, and R. Salakhutdinov, *Point cloud GAN*, arXiv preprint arXiv:1810.05795, (2018). (Cited on pages 47 and 49.)

[151] Y. Liao, S. Donne, and A. Geiger, *Deep marching cubes: Learning explicit surface representations*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2916–2925. (Cited on pages 61 and 76.)

[152] Y. Lipman, D. Cohen-Or, R. Gal, and D. Levin, *Volume and shape preservation via moving frame manipulation*, ACM Transactions on Graphics (TOG), 26 (2007), p. 5. (Cited on page 48.)

[153] M. Liu, X. Zhang, and H. Su, *Meshing point clouds with predicted intrinsic-extrinsic ratio guidance*, arXiv preprint arXiv:2007.09267, (2020). (Cited on pages 60, 62, 66, 67 and 76.)

[154] S. Liu, T. Li, W. Chen, and H. Li, *Soft rasterizer: A differentiable renderer for image-based 3d reasoning*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 7708–7717. (Cited on page 80.)

[155] X. Liu, Z. Han, X. Wen, Y.-S. Liu, and M. Zwicker, *L2g auto-encoder: Understanding point clouds by local-to-global reconstruction with hierarchical self-attention*, in Proceedings of the 27th ACM International Conference on Multimedia, ACM, 2019, pp. 989–997. (Cited on page 49.)

[156] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, ACM siggraph computer graphics, 21 (1987), pp. 163–169. (Cited on pages 59 and 61.)

[157] A. Loseille, *Unstructured mesh generation and adaptation*, in Handbook of Numerical Analysis, vol. 18, Elsevier, 2017, pp. 263–302. (Cited on pages 87, 88, 90 and 94.)

[158] S. Luo and W. Hu, *Diffusion probabilistic models for 3d point cloud generation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2837–2845. (Cited on pages 9, 15 and 95.)

[159] J. Mairal, *Sparse coding for machine learning, image processing and computer vision*, PhD thesis, Cachan, Ecole normale supérieure, 2010. (Cited on page 24.)

[160] E. Marchandise, J.-F. Remacle, and C. Geuzaine, *Optimal parametrizations for surface remeshing*, Engineering with Computers, 30 (2014), pp. 383–402. (Cited on page 79.)

[161] R. Marin*, M.-J. Rakotosaona*, S. Melzi, and M. Ovsjanikov, *Correspondence learning via linearly-invariant embedding*, in NeurIPS, 2020. (Cited on pages 12, 19 and 80.)

[162] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, *Geodesic convolutional neural networks on riemannian manifolds*, in ICCV workshops, 2015, pp. 37–45. (Cited on page 22.)

[163] E. Mattei and A. Castrodad, *Point cloud denoising via moving rpca*, in CGFm, vol. 36, 2017, pp. 123–137. (Cited on page 24.)

[164] B. Mederos, L. Velho, and L. H. de Figueiredo, *Point cloud denoising*, in SIAM Conference on Geometric Design and Computing, Citeseer, 2003, pp. 1–11. (Cited on page 24.)

[165] P. Memari, P. Mullen, and M. Desbrun, *Parametrization of generalized primal-dual triangulations*, in Proceedings of the 20th International Meshing Roundtable, Springer, 2011, pp. 237–253. (Cited on pages 77 and 79.)

[166] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, *Occupancy networks: Learning 3d reconstruction in function space*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4460–4470. (Cited on page 61.)

[167] P. W. Michor and D. B. Mumford, *Riemannian geometries on spaces of plane curves*, Journal of the European Mathematical Society, (2006). (Cited on page 48.)

[168] L. Morreale, N. Aigerman, V. Kim, and N. J. Mitra, *Neural surface maps*, 2021. (Cited on page 95.)

[169] P. Mullen, P. Memari, F. de Goes, and M. Desbrun, *Hot: Hodge-optimized triangulations*, in ACM SIGGRAPH 2011 papers, 2011, pp. 1–12. (Cited on page 79.)

[170] C. Nash, Y. Ganin, S. Eslami, and P. W. Battaglia, *Polygen: An autoregressive generative model of 3d meshes*, arXiv preprint arXiv:2002.10880, (2020). (Cited on page 61.)

[171] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, *Laplacian mesh optimization*, in Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, 2006, pp. 381–389. (Cited on page 75.)

[172] T. S. Newman and H. Yi, *A survey of the marching cubes algorithm*, Computers & Graphics, 30 (2006), pp. 854–879. (Cited on pages 59 and 61.)

[173] A. Y. Ng, M. I. Jordan, Y. Weiss, et al., *On spectral clustering: Analysis and an algorithm*, Advances in neural information processing systems, 2 (2002), pp. 849–856. (Cited on page 84.)

[174] K.-W. Ng and Z.-W. Low, *Simplification of 3d triangular mesh for level of detail computation*, in 2014 11th International Conference on Computer Graphics, Imaging and Visualization, IEEE, 2014, pp. 11–16. (Cited on page 77.)

[175] M. Oechsle, S. Peng, and A. Geiger, *Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction*, arXiv preprint arXiv:2104.10078, (2021). (Cited on page 98.)

[176] H. Ozaki, F. Kyota, and T. Kanai, *Out-of-core framework for qem-based mesh simplification*, in Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization, 2015, pp. 87–96. (Cited on page 77.)

[177] A. C. Öztireli, G. Guennebaud, and M. Gross, *Feature preserving point set surfaces based on non-linear kernel regression*, in CGF, vol. 28, 2009, pp. 493–501. (Cited on page 24.)

[178] J. Pan, X. Han, W. Chen, J. Tang, and K. Jia, *Deep mesh reconstruction from single rgb images via topology modification networks*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9964–9973. (Cited on page 61.)

[179] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, *Deepsdf: Learning continuous signed distance functions for shape representation*,

in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 165–174. (Cited on pages 60, 61 and 98.)

[180] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep learning library*, arXiv preprint arXiv:1912.01703, (2019). (Cited on pages 76 and 90.)

[181] R. Pincus, *Barnett, v., and lewis t.: Outliers in statistical data*, Biometrical Journal, 37 (1995), pp. 256–256. (Cited on page 23.)

[182] A. Poulenard, M.-J. Rakotosaona, Y. Ponty, and M. Ovsjanikov, *Effective rotation-invariant point cnn with spherical harmonics kernels*, in 2019 International Conference on 3D Vision (3DV), IEEE, 2019, pp. 47–56. (Cited on pages 12 and 19.)

[183] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, in Proc. CVPR, 2017, pp. 652–660. (Cited on pages 22, 24, 26, 27, 28, 47, 49, 50, 51, 54, 95 and 116.)

[184] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, arXiv preprint arXiv:1706.02413, (2017). (Cited on pages 8, 9, 15, 24, 26, 47, 49 and 54.)

[185] M.-J. Rakotosaona, N. Aigerman, N. J. Mitra, M. Ovsjanikov, and P. Guerrero, *Differentiable surface triangulation*, ACM Transactions on Graphics (TOG), (2021). (Cited on pages 12 and 18.)

[186] M.-J. Rakotosaona, P. Guerrero, N. Aigerman, N. J. Mitra, and M. Ovsjanikov, *Learning delaunay surface elements for mesh reconstruction*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 22–31. (Cited on pages 12, 18 and 76.)

[187] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, *PointCleanNet: Learning to denoise and remove outliers from dense point clouds*, in Computer Graphics Forum, vol. 39, Wiley Online Library, 2020, pp. 185–203. (Cited on pages 8, 11, 14 and 18.)

[188] M.-J. Rakotosaona and M. Ovsjanikov, *Intrinsic point cloud interpolation via dual latent space navigation*, in European Conference on Computer Vision, Springer, 2020, pp. 655–672. (Cited on pages 9, 11, 15 and 18.)

[189] P. J. Rousseeuw and M. Hubert, *Robust statistics for outlier detection*, WIREs: Data Mining and Knowledge Discovery, 1 (2011), pp. 73–79. (Cited on page 23.)

[190] R. Roveri, A. C. Öztireli, I. Pandele, and M. Gross, *Pointpronets: Consolidation of point clouds with convolutional neural networks*, CGF, 37 (2018), pp. 87–99. (Cited on pages 25, 35, 36 and 37.)

[191] J. Sassen, B. Heeren, K. Hildebrandt, and M. Rumpf, *Solving Variational Problems Using Nonlinear Rotation-invariant Coordinates*, in Symposium on Geometry Processing 2019- Posters, D. Bommes and H. Huang, eds., The Eurographics Association, 2019. (Cited on page 49.)

[192] H. Shao, A. Kumar, and P. Thomas Fletcher, *The riemannian geometry of deep generative models*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2018, pp. 315–323. (Cited on pages 49 and 54.)

[193] N. Sharp and K. Crane, *You can find geodesic paths in triangle meshes by just flipping edges*, ACM Transactions on Graphics (TOG), 39 (2020), pp. 1–15. (Cited on page 85.)

[194] N. Sharp and M. Ovsjanikov, *Pointtrinet: Learned triangulation of 3d point sets*, arXiv preprint arXiv:2005.02138, (2020). (Cited on pages 60, 62, 67 and 76.)

[195] N. Sharp, Y. Soliman, and K. Crane, *The vector heat method*, ACM Trans. Graph., 38 (2019). (Cited on page 67.)

[196] Y. Shen, C. Feng, Y. Yang, and D. Tian, *Mining point cloud local structures by kernel correlation and graph pooling*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4548–4557. (Cited on page 47.)

[197] J. Shewchuk, T. K. Dey, and S.-W. Cheng, *Delaunay mesh generation*, Chapman and Hall/CRC, 2016. (Cited on page 61.)

[198] J. Solomon, F. De Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas, *Convolutional wasserstein distances: Efficient optimal transportation on geometric domains*, ACM Transactions on Graphics (TOG), 34 (2015), p. 66. (Cited on page 49.)

[199] Y. Sun, S. Schaefer, and W. Wang, *Denoising point sets via l0 minimization*, CAGD, 35 (2015), pp. 2–15. (Cited on page 24.)

[200] V. Surazhsky and C. Gotsman, *Explicit surface remeshing*, in Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2003, pp. 20–30. (Cited on page 78.)

[201] G. Taubin, *Curve and surface smoothing without shrinkage*, in Computer Vision, IEEE, 1995, pp. 852–857. (Cited on page 30.)

[202] ——, *A signal processing approach to fair surface design*, in Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 351–358. (Cited on page 79.)

[203] C. D. Toth, J. O'Rourke, and J. E. Goodman, *Handbook of discrete and computational geometry*, CRC press, 2017. (Cited on pages 75 and 80.)

[204] J. Tournois, P. Alliez, and O. Devillers, *Interleaving delaunay refinement and optimization for 2d triangle mesh generation*, in Proceedings of the 16th international meshing roundtable, Springer, 2008, pp. 83–101. (Cited on page 75.)

[205] S. Valette, J. M. Chassery, and R. Prost, *Generic remeshing of 3d triangular meshes with metric-dependent discrete voronoi diagrams*, IEEE Transactions on Visualization and Computer Graphics, 14 (2008), pp. 369–381. (Cited on page 79.)

[206] B. Vallet and B. Lévy, *Spectral geometry processing with manifold harmonics*, in Computer Graphics Forum, vol. 27, Wiley Online Library, 2008, pp. 251–260. (Cited on pages 7 and 14.)

[207] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, *Learning from synthetic humans*, in CVPR, 2017. (Cited on page 54.)

[208] A. Vaxman, C. Müller, and O. Weber, *Conformal mesh deformations with möbius transformations*, ACM Transactions on Graphics (TOG), 34 (2015), p. 55. (Cited on page 48.)

[209] W. Von Funck, H. Theisel, and H.-P. Seidel, *Vector field based shape deformations*, ACM Transactions on Graphics (TOG), 25 (2006), pp. 1118–1125. (Cited on page 48.)

[210] P. von Radziewsky, E. Eisemann, H.-P. Seidel, and K. Hildebrandt, *Optimized subspaces for deformation-based modeling and shape interpolation*, Computers & Graphics, 58 (2016), pp. 128–138. (Cited on page 49.)

[211] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, *Pixel2mesh: Generating 3d mesh models from single rgb images*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 52–67. (Cited on page 61.)

[212] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, *O-cnn: Octree-based convolutional neural networks for 3d shape analysis*, ACM Transactions On Graphics (TOG), 36 (2017), pp. 1–11. (Cited on pages 9 and 15.)

[213] P.-S. Wang, Y. Liu, and X. Tong, *Mesh denoising via cascaded normal regression*, ACM TOG, 35 (2016), p. 232. (Cited on pages 24, 41 and 42.)

[214] X. Wang, X. Ying, Y.-J. Liu, S.-Q. Xin, W. Wang, X. Gu, W. Mueller-Wittig, and Y. He, *Intrinsic computation of centroidal voronoi tessellation (cvt) on meshes*, Computer-Aided Design, 58 (2015), pp. 51–61. (Cited on page 79.)

[215] Y. Wang, B. Liu, and Y. Tong, *Linear surface reconstruction from discrete fundamental forms on triangle meshes*, in Computer Graphics Forum, vol. 31, Wiley Online Library, 2012, pp. 2277–2287. (Cited on page 53.)

[216] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, *Dynamic graph cnn for learning on point clouds*, arXiv preprint arXiv:1801.07829, (2018). (Cited on pages 22, 24, 26, 35, 36 and 37.)

[217] J. Wei and Y. Lou, *Feature preserving mesh simplification using feature sensitive metric*, Journal of Computer Science and Technology, 25 (2010), pp. 595–605. (Cited on page 77.)

[218] L. Wei, Q. Huang, D. Ceylan, E. Vouga, and H. Li, *Dense human body correspondences using convolutional networks*, in CVPR, 2016, pp. 1544–1553. (Cited on page 22.)

[219] F. Williams, T. Schneider, C. Silva, D. Zorin, J. Bruna, and D. Panozzo, *Deep geometric prior for surface reconstruction*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10130–10139. (Cited on page 61.)

[220] B. Wirth, L. Bar, M. Rumpf, and G. Sapiro, *A continuum mechanical approach to geodesics in shape space*, International Journal of Computer Vision, 93 (2011), pp. 293–318. (Cited on page 49.)

[221] K. Wolff, C. Kim, H. Zimmer, C. Schroers, M. Botsch, O. Sorkine-Hornung, and A. Sorkine-Hornung, *Point cloud noise and outlier removal for image-based 3d reconstruction*, in 3D Vision, IEEE, 2016, pp. 118–127. (Cited on pages 23 and 43.)

[222] J. Xie, Y. Xu, Z. Zheng, S.-C. Zhu, and Y. N. Wu, *Generative pointnet: Deep energy-based learning on unordered point sets for 3d generation, reconstruction and classification*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 14976–14985. (Cited on pages 8 and 15.)

[223] D. Xu, H. Zhang, Q. Wang, and H. Bao, *Poisson shape interpolation*, Graphical models, 68 (2006), pp. 268–281. (Cited on page 48.)

[224] D.-M. Yan, G. Bao, X. Zhang, and P. Wonka, *Low-resolution remeshing using the localized restricted voronoi diagram*, IEEE transactions on visualization and computer graphics, 20 (2014), pp. 1418–1427. (Cited on page 79.)

[225] D.-M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang, *Isotropic remeshing with fast and exact computation of restricted voronoi diagram*, in Computer graphics forum, vol. 28, Wiley Online Library, 2009, pp. 1445–1454. (Cited on page 79.)

[226] D.-M. Yan and P. Wonka, *Non-obtuse remeshing with centroidal voronoi tessellation*, IEEE transactions on visualization and computer graphics, 22 (2015), pp. 2136–2144. (Cited on page 79.)

[227] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, *Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling*, in Proceedings of

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 5589–5598. (Cited on page 97.)

[228] Y. Yang, C. Feng, Y. Shen, and D. Tian, *Foldingnet: Point cloud auto-encoder via deep grid deformation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 206–215. (Cited on pages 64 and 116.)

[229] W. Yifan, N. Aigerman, V. G. Kim, S. Chaudhuri, and O. Sorkine-Hornung, *Neural cages for detail-preserving 3d deformations*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 75–83. (Cited on page 97.)

[230] K. Yin, H. Huang, D. Cohen-Or, and H. Zhang, *P2p-net: Bidirectional point displacement net for shape transform*, ACM TOG, 37 (2018), pp. 152:1–152:13. (Cited on pages 24 and 45.)

[231] Y.-J. Yoon, A. Lelidis, A. C. Öztireli, J.-M. Hwang, M. Gross, and S.-M. Choi, *Geometry representations with unsupervised feature learning*, in Big Data and Smart Computing, IEEE, 2016, pp. 137–142. (Cited on page 24.)

[232] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, *Pu-net: Point cloud upsampling network*, in CVPR, 2018. (Cited on pages 24 and 45.)

[233] W. Yue, Q. Guo, J. Zhang, and G. Wang, *3d triangular mesh optimization in geometry processing for cad*, in Proceedings of the 2007 ACM symposium on Solid and physical modeling, 2007, pp. 23–33. (Cited on page 78.)

[234] J. Zeng, G. Cheung, M. Ng, J. Pang, and C. Yang, *3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model*, arXiv preprint arXiv:1803.07252, (2018). (Cited on page 24.)

[235] B. Zhang and P. Wonka, *Point cloud instance segmentation using probabilistic embeddings*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 8883–8892. (Cited on pages 8, 9 and 15.)

[236] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, *Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising*, IEEE TIP, 26 (2017), pp. 3142–3155. (Cited on page 24.)

[237] K. Zhang, W. Zuo, S. Gu, and L. Zhang, *Learning deep cnn denoiser prior for image restoration*, in CVPR, vol. 2, 2017. (Cited on page 24.)

[238] Z. Zhang, G. Li, H. Lu, Y. Ouyang, M. Yin, and C. Xian, *Fast as-isometric-as-possible shape interpolation*, Computers & Graphics, 46 (2015), pp. 244–256. (Cited on page 49.)

[239] Q. Zheng, A. Sharf, G. Wan, Y. Li, N. J. Mitra, D. Cohen-Or, and B. Chen, *Non-local scan consolidation for 3d urban scenes*, ACM TOG, 29 (2010), pp. 94–1. (Cited on page 24.)

[240] Q. Zhou and A. Jacobson, *Thingi10k: A dataset of 10,000 3d-printing models*, arXiv preprint arXiv:1605.04797, (2016). (Cited on pages 67, 87 and 90.)

[241] S. Zuffi, A. Kanazawa, D. Jacobs, and M. J. Black, *3D menagerie: Modeling the 3D shape and pose of animals*, in IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), July 2017. (Cited on page 54.)

ECOLE
DOCTORALE

Titre : Représentations et méthodes basées sur l'apprentissage pour l'analyse, la manipulation et la reconstruction de formes en 3D

Mots clés : maillage, géométrie, représentation de forme, réseaux de neurones, interpolation, débruitage

Résumé : Traiter et analyser efficacement les données 3D est un défi crucial dans les applications modernes, car les formes 3D sont de plus en plus répandues avec la prolifération des dispositifs d'acquisition et des outils de modélisation. Alors que les succès de l'apprentissage profond en 2D sont devenus monnaie courante et entourent notre vie quotidienne, les applications qui impliquent des données 3D sont à la traîne. En raison de la structure non uniforme plus complexe des formes 3D, les méthodes d'apprentissage profond en 2D ne peuvent pas être facilement étendues et il existe une forte demande pour de nouvelles approches qui peuvent à la fois exploiter et permettre l'apprentissage en utilisant la structure géométrique. De plus, être capable de gérer les différentes représentations existantes des formes 3D telles que les nuages de points et les maillages, ainsi que les artefacts produits par les dispositifs d'acquisition 3D augmente la difficulté de la tâche. Dans cette thèse, nous proposons des approches systématiques qui exploitent pleinement les informations géométriques des données 3D dans des architectures d'apprentissage profond. Nous contribuons aux méthodes de débruitage de nuages de points, d'interpolation de formes et de reconstruction de formes. Nous observons que les architectures d'apprentissage profond facilitent l'apprentissage de la structure de surface sous-jacente des nuages de points, qui peut ensuite être utilisée pour le débruitage et l'interpolation de formes. L'encodage de prieurs appris basés sur des patchs locaux, ainsi que d'informations géométriques complémentaires telles que la longueur des arrêtes, permet de créer des pipelines puissants qui génèrent des formes réalistes. Le principal fil conducteur de nos contributions est de faciliter la conversion entre différentes représentations de formes. En particulier, alors que l'utilisation de l'apprentissage profond sur des mailles triangulaires est complexe en raison de leur nature combinatoire, nous introduisons des méthodes inspirées du traitement de la géométrie qui permettent la création et la manipulation de faces de triangles. Nos méthodes sont robustes et se généralisent bien aux données inconnues malgré des jeux d'entraînement limités. Notre travail ouvre donc la voie à une manipulation plus générale, robuste et universellement utile des données 3D.

Title : Learning-based representations and methods for 3D shape analysis, manipulation and reconstruction

Keywords : meshing, geometry processing, surface representation, neural networks, interpolation, denoising

Abstract : Efficiently processing and analysing 3D data is a crucial challenge in modern applications as 3D shapes are becoming more and more widespread with the proliferation of acquisition devices and modeling tools. While successes of 2D deep learning have become commonplace and surround our daily life, applications that involve 3D data are lagging behind. Due to the more complex non-uniform structure of 3D shapes, successful methods from 2D deep learning cannot be easily extended and there is a strong demand for novel approaches that can both exploit and enable learning using geometric structure. Moreover, being able to handle the various existing representations of 3D shapes such as point clouds and meshes, as well as the artefacts produced from 3D acquisition devices increases the difficulty of the task. In this thesis, we propose systematic approaches that fully exploit geometric information of 3D data in deep learning architectures. We contribute to point cloud denoising, shape interpolation and shape reconstruction methods. We observe that deep learning architectures facilitate learning the underlying surface structure on point clouds that can then be used for denoising as well as shape interpolation. Encoding local patch-based learned priors, as well as complementary geometric information such as edge lengths, leads to powerful pipelines that generate realistic shapes. The key common thread throughout our contributions is facilitating seamless conversion between different representations of shapes. In particular, while using deep learning on triangle meshes is highly challenging due to their combinatorial nature we introduce methods inspired from geometry processing that enable the creation and manipulation of triangle faces. Our methods are robust and generalize well to unseen data despite limited training sets. Our work, therefore, paves the way towards more general, robust and universally useful manipulation of 3D data.

Institut Polytechnique de Paris
91120 Palaiseau, France