Shape reconstruction from 3D point clouds

Rao Fu

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Reconstruction de formes à partir de nuages de points 3D

Rao FU

Centre Inria d'Université Côte d'Azur
Equipe, TITANE

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur
Dirigée par : Pierre Alliez
Soutenue le : 21 février 2024

Devant le jury, composé de :
Jean-Luc Mari, Professeur, Univ. Aix-Marseille
Guillaume Lavoué, Professeur, ENISE Saint-Etienne
Nicolas Mellado, CRCN CNRS, IRIT Toulouse
Noura Faraj, Maitre de conférence, Univ. Montpellier

Inria

HORIZON 2020

Geometry Factory

Reconstruction de formes à partir de nuages de points 3D

Shape reconstruction from 3D point clouds

Jury:

Rapporteurs
Guillaume Lavoué, Professeur, ENISE Saint-Etienne
Jean-Luc Mari, Professeur, Université Aix-Marseille

Examinateurs
Nicolas Mellado, Chargé de recherche CNRS, Laboratoire IRIT, Toulouse
Noura Faraj, Maitre de conférence, Université de Montpellier

Directeur de thèse
Pierre Alliez, Directeur de recherche, Centre Inria d'Université Côte d'Azur

Invitée
Jane Tournois, R&D Engineer, GeometryFactory

Acknowledgements

I would like to extend my gratitude to those who have supported me during my PhD thesis.

Résumé

La reconstruction de formes 3D à partir de nuages de points bruts est un défi scientifique central dans le domaine de la modélisation géométrique. Cette thèse explore ce défi notoirement multifacette, en se concentrant particulièrement sur deux aspects : la segmentation de primitives et la reconstruction de surfaces basée sur des maillages.

En ce qui concerne la segmentation de primitives, nous introduisons BPNet, une architecture d'apprentissage profond conçue pour segmenter des nuages de points 3D avec des primitives de Bézier. Contrairement aux méthodes conventionnelles qui traitent différents types de primitives indépendamment, BPNet est générale et plus adaptable. Inspirés par les techniques de décomposition de Bézier couramment utilisées pour les modèles de surfaces NURBS, nous utilisons la décomposition de Bézier pour guider la segmentation des nuages de points 3D, éliminant ainsi les contraintes imposées par des types spécifiques de primitives. Notre contribution est une méthode d'optimisation conjointe comprenant un terme de régularisation qui améliore la segmentation des primitives, un module qui favorise le regroupement de points partageant les mêmes caractéristiques et un module de reconstruction qui affine les primitives segmentées. L'approche proposée est validée sur les données ABC et AIM@Shape. Nos expériences montrent des performances de segmentation supérieures et un temps d'inférence plus court que l'état de l'art.

En ce qui concerne la reconstruction de surfaces, nous contrinuons une méthode conçue pour générer directement des maillages surfaciques triangulaires isotropes à partir de nuages de points bruts. Les avantages de cette approche résident dans sa capacité à s'adapter à une taille locale caractéristique (LFS) estimée sur le nuage de points. Cette méthode se compose de trois étapes : l'estimation de la LFS, la reconstruction de la fonction implicite et le dimensionnement du maillage adapté à la LFS estimée. Le processus d'estimation de la LFS calcule le minimum de deux propriétés géométriques : le rayon de courbure local et le diamètre de la forme, déterminés par des techniques d'ajustement de surfaces paramétriques locales (jets) et une recherche dichotomique contrainte par la propriété 1-Lipschitz de la LFS. L'étape de reconstruction de la fonction implicite se déroule en trois sous-étapes : la construction d'un multi-domaine 3D décomposé en tétraèdres à partir d'une fonction de distance non signée, la signature du multi-domaine par ajustement de données, et la génération d'une fonction de distance robuste signée. Enfin, la

fonction de dimensionnement du maillage, dérivée de la LFS estimée localement, contrôle le processus de raffinement de Delaunay utilisé pour mailler l'ensemble de niveau zéro de la fonction implicite. Cette approche permet d'obtenir, directement à partir de nuages de points 3D, des maillages isotropes tout en offrant une densité de maillage adaptée à la LFS. Nos expériences démontrent la robustesse de cette approche, en montrant sa capacité à gérer le bruit, les valeurs aberrantes ou les données manquantes.

Mots clés: Nuages de points, Apprentissage en profondeur, Segmentation de primitives, Décomposition de Bézier, Reconstruction de surfaces, Taille de caractéristiques locales, Fonction implicite

Abstract

Shape reconstruction from raw 3D point clouds is a core challenge in the field of computer graphics and computer vision. This thesis explores new approaches to tackle this multifaceted problem, with a specific focus on two aspects: primitive segmentation and mesh-based surface reconstruction.

For primitive segmentation, we introduce BPNet, a deep-learning framework designed to segment 3D point clouds according to Bézier primitives. Unlike conventional methods that address different primitive types in isolation, BPNet offers a more general and adaptive approach. Inspired by Bézier decomposition techniques commonly employed for Non-Uniform Rational B-Spline (NURBS) models, we employ Bézier decomposition to guide the segmentation of point clouds, removing the constraints posed by specific categories of primitives. More specifically, we contribute a joint optimization framework via a soft voting regularizer that enhances primitive segmentation, an auto-weight embedding module that streamlines the clustering of point features, and a reconstruction module that refines the segmented primitives. We validate the proposed approach on the ABC and AIM@Shape datasets. The experiments show superior segmentation performance and shorter inference time compared to baseline methods.

On surface reconstruction, we contribute a method designed to generate isotropic surface triangle meshes directly from unoriented 3D point clouds. The benefits of this approach lie in its adaptability to local feature size (LFS). Our method consists of three steps: LFS estimation, implicit function reconstruction and LFS-aware mesh sizing. The LFS estimation process computes the minimum of two geometric properties: local curvature radius and shape diameter, determined via jet-fitting and a Lipschitz-guided dichotomic search. The implicit function reconstruction step proceeds in three sub-steps: constructing a tetrahedron multi-domain from an unsigned distance function, signing the multi-domain via data fitting, and generating a signed robust distance function. Finally, the mesh sizing function, derived from the locally estimated LFS, controls the Delaunay refinement process used to mesh the zero-level set of the implicit function. This approach yields isotropic meshes directly from 3D point clouds while offering an LFS-aware mesh density. Our experiments demonstrate the robustness of this approach, showcasing its ability to handle noise, outliers or missing data.

Keywords: Point clouds, Deep learning, Primitive segmentation, Bézier decomposition, Surface reconstruction, Local feature size, Implicit function

Contents

CHAPTER 1

Introduction

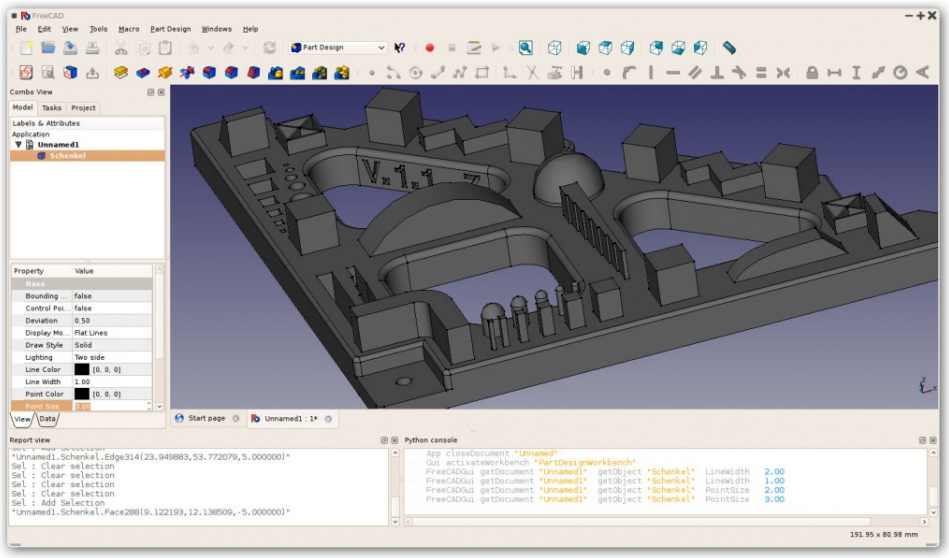
1.1 Context

In recent years, there has been a growing demand from practitioners to reconstruct shapes from 3D point clouds. This demand spans various fields, including manufacturing, video games and specialized applications like autonomous driving and digital twinning. This increasing demand signifies a substantial commercial market with promising investment prospects, capturing the attention of both industry and academia.

In manufacturing, shape reconstruction is a crucial functionality for computer-aided design (CAD) software, which is widely employed in mechanical prototyping, aerospace analysis and urban planning. In this context, shape reconstruction techniques transform raw 3D point cloud data, often acquired through 3D scanning or other measurement methods, into CAD models. This process facilitates shape reconstruction, converting real-world objects into digital copies. Figure 1.1 presents an example of a CAD software that utilizes shape reconstruction to design mechanical components.

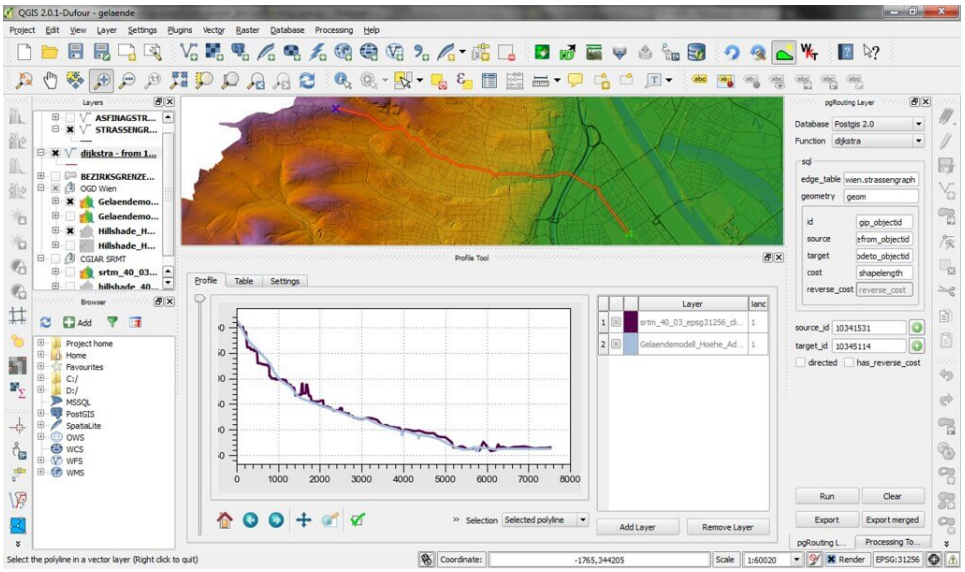
In geography, shape reconstruction plays a pivotal role in geographic information systems (GIS) software for analyzing and visualizing geographical data. GIS applications depend on precise shape representations for terrain modeling and environmental simulations. In this scenario, raw point cloud data, often obtained through LiDAR, serves as the source for shape reconstruction. These reconstructed shapes assist in downstream GIS-based flood modeling, land-use planning and disaster management analysis. Figure 1.2 shows an example of applying a GIS software for geological map analysis.

The task of shape reconstruction from 3D point clouds is a challenging problem and remains a rapidly evolving area of research. It is an inherently ill-posed problem by nature. The illness arises because a point cloud represents a discrete sampling of an original surface, which means some information about the original surface is inevitably lost. In contrast, shape reconstruction seeks to infer the original



(a) Computer-aided design (CAD)

Figure 1.1: Shape reconstruction in CAD. Applying FreeCAD to design mechanical parts. Image taken from FreeCAD documentation [RMvH16].



(a) Geographic information systems (GIS)

Figure 1.2: Shape reconstruction in GIS. Applying QGIS to generate a geological map. Image taken from the QGIS documentation [MP18].

continuous surface from this limited and discrete point cloud data. This mismatch between the discrete nature of the point cloud data and the objective of reconstruction of a continuous surface makes the shape reconstruction a complex scientific problem.

Existing solutions for shape reconstruction often follow two primary directions. The first direction involves fitting multiple Non-Uniform Rational B-Spline (NURBS) surface patches to approximate the original shape using point cloud data. NURBS surfaces are defined by control points and blending functions, offering a smooth and precise representation of shapes. However, this direction can be challenging when dealing with complex shapes that contain rich details, requiring the segmentation of point clouds before fitting. Nevertheless, combining segmentation with NURBS fitting remains a complex problem, and no mature packages or algorithms are currently readily available for industrial applications.

The second direction in shape reconstruction involves constructing a surface triangle mesh to approximate the original shape, a widely employed approach supported by mature packages that offer algorithms for mesh generation. For instance, the CGAL library [CGA09] provides various surface reconstruction algorithms built upon the half-edge data structure [BKP+10] for mesh representation. However, challenges persist when dealing with real-world 3D point cloud data, which may be imperfect due to limitations in scanning equipment. Additionally, achieving high-fidelity mesh reconstructions while maintaining simplicity and well-shaped triangles remains an open and active area of research.

Before delving into the detailed scientific challenges associated with shape reconstruction, we first discuss how to obtain 3D point clouds, which serve as the direct input source for our shape reconstruction task. These 3D point clouds are assemblies of three-dimensional coordinates, often complemented by additional attributes like normals or RGB colors. To provide context, we explore popular techniques for acquiring 3D point clouds. Examining the principles and technologies behind these data-acquisition techniques offers valuable insights into the scientific challenges that arise for the shape reconstruction task.

Laser Scanning. Laser scanning (Figure 1.3) can capture precise 3D point clouds of the original shape. This process involves emitting laser beams toward the target and measuring the time it takes for the laser pulses to bounce back to a sensor. Laser scanners generate dense point cloud data representing the original shape by analyzing these measurements. However, certain challenges arise with laser scanning. High-reflection materials like glass can pose difficulties, leading to missing

data in the point clouds. Additionally, occlusions caused by objects obstructing the scanner's line of sight can result in gaps in the acquired data, requiring additional processing to address these issues. The sampling density, often measured in points per square meter, and accuracy, usually in millimeters, are critical factors in the quality of data acquired through laser scanning. They influence the success of subsequent shape reconstruction efforts.

Figure 1.3: Laser scanning. A laser scanner is used to scan point clouds from a construction field. Image taken from [WYLU22].

Structured Light Scanning. Structured light scanning (Figure 1.4) utilizes a projector to cast a known pattern onto the surface of the target and a camera to capture how that pattern deforms, resulting in a structured point cloud representing the original shape. Unlike laser scanning, which is employed to capture large-scale scenes, structured light scanning is suitable for small to medium-sized shapes. However, structured light scanning still struggles with highly reflective or transparent materials. The accuracy and density of the generated point cloud depend on factors such as the projector, camera quality and the complexity of the projected pattern.

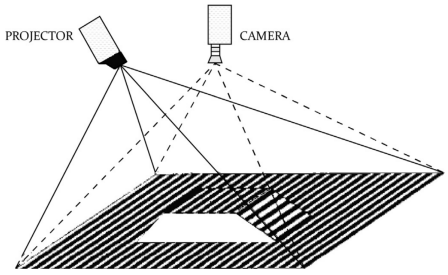


Figure 1.4: Structured Light Scanning. Standard projector–camera configuration used in structured light profilometry techniques. Image taken from [VdJD16].

Photogrammetry. Photogrammetry (Figure 1.5) relies on capturing and analyzing multiple 2D images of a shape taken from different viewpoints. It leverages image feature matching and stereo principles to reconstruct the 3D point coordinates from multiple images. However, 3D point clouds obtained by photogrammetry are usually noisy, as the lighting conditions and reflections can affect the quality of the images and, consequently, the accuracy of the reconstruction. Calibration and synchronization of multiple cameras are also critical for precise results. In addition, the 3D point clouds suffer from extensive missing data because parts of the shape are occluded or not captured clearly in some images. This results in gaps or discontinuities in the point cloud data, challenging the shape reconstruction task. Despite these defects, photogrammetry remains a valuable tool, particularly when dealing with scenarios that are more amenable to image-based data capture.

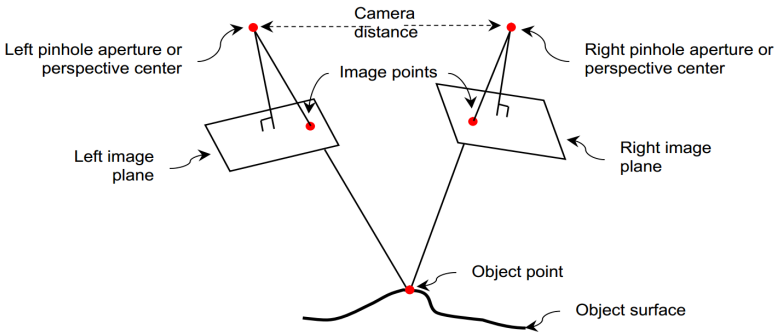


Figure 1.5: Photogrammetry. A schematic of the stereophotogrammetry technique showing the two cameras and how the location of a point is identified. Image taken from [HNAS11].

Depth Sensors. Depth sensors (Figure 1.6), like Laser scanners, also operate by emitting infrared light pulses and measuring their time-of-flight (ToF) to calculate distances. It then generates depth maps containing the shape structure based on the ToF, where we can obtain the 3D point cloud from the depth image. However, noise is a notable concern in depth sensors. Noise levels typically vary with distance, with greater distances resulting in less accurate depth measurements. Environmental factors like ambient light can also introduce noise, particularly in outdoor settings. Despite these limitations, depth sensors are favored for their cost-effectiveness and have found widespread usage in various applications, especially for scenarios where real-time depth perception is indispensable.

Figure 1.6: Depth Sensor: The Azure Kinect DK depth camera. Image taken from Microsoft [Mic23].

Simulation. Simulation generates raw point clouds digitally, emulating real-world scanning scenarios. It allows for controlled experiments and preserves the raw essence of intended environments or objects. However, simulations rely upon accurate modeling and assumptions, which can introduce errors. They may also lack the richness and defects of real-world data, limiting their real-world applicability.

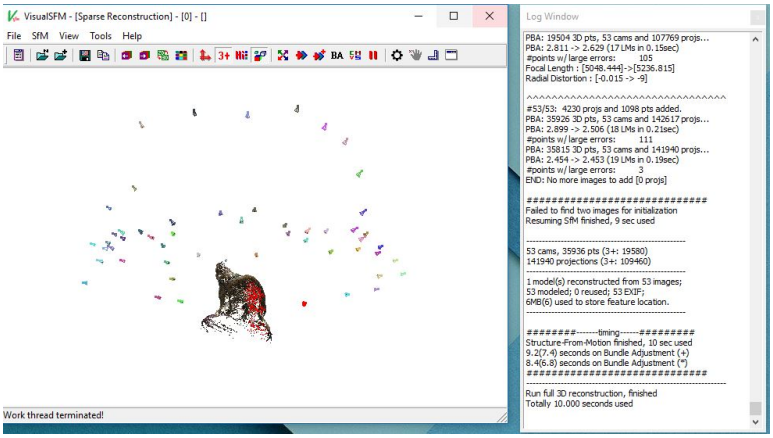


Figure 1.7: Simulation: point clouds with synthesized cameras. Image taken from VisualSFM [Wu11].

Different acquisition methods yield raw 3D point clouds, which are crucial in various applications. Laser scanning excels in high-precision data capture, making it suitable for metrology and large-scale scanning. Structured light offers real-time capabilities, making it valuable in interactive scenarios like gaming and virtual

reality. Photogrammetry, while sometimes noisy, can handle city-scale outdoor environments, making it important for geographic information systems. Depth sensors are cost-effective and find applications in consumer devices like gaming consoles. However, the raw point clouds obtained from the aforementioned scanning technologies often come with imperfections due to inherent limitations. These imperfections motivate the design of robust shape reconstruction algorithms.

1.2 Scientific Challenges

Shape reconstruction refers to reconstruct a surface S from a given point cloud $P = \{p \in \mathbb{R}^3\}$. The problem lies in developing robust and efficient algorithms capable of transforming these unprocessed point clouds into accurate and complete 3D representations of objects. Addressing this problem involves tackling issues such as noise reduction, surface reconstruction, hole filling and ensuring geometric fidelity. The problem is multifaceted, marked by intricate challenges in those pivotal aspects: sampling conditions, reconstruction priors, geometric fidelity, topology, memory usage and computation, automation, generalization and evaluation criteria. Within the scope of this thesis, the focus is exclusively on unstructured 3D point clouds.

Sampling conditions. In the context of sampling conditions, the complexities arise from the diverse nature of data acquisition methods, each imparting specific characteristics to the raw point clouds. Challenges include variations in point density, presence of non-uniform sampling, the inherent noise in the acquired datasets, outliers from data inaccuracies or sensor errors, misalignment from multiple scans and missing data from occlusions. Figure 1.8 illustrates those challenges in 2D.

Reconstruction priors. These varied reconstruction priors correspond to different methodologies into the shape reconstruction problem, reflecting the assumption of priors about the nature of the unknown surface. Some methods assume global smoothness, expecting the surface to have a continuous and smooth variation. Others consider piecewise smoothness, where the surface can exhibit distinct regions of smoothness separated by discontinuities. There are also approaches assuming piecewise linearity, where a set of triangles directly interpolates the unknown surface from the input points. However, in the presence of noise in the point data, global smooth priors may lead to over-smoothing by approximation, piecewise

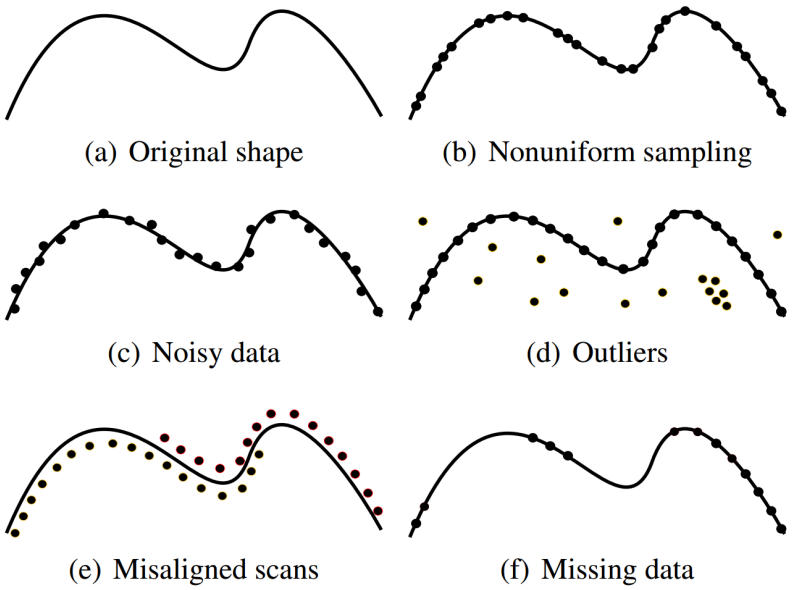


Figure 1.8: Different sampling conditions. The point cloud artifacts include non-uniform sampling, noisy data, outliers, misaligned scans and missing data. Image taken from [BTS+17].

smooth priors may struggle to identify the distinct regions, while piecewise linear priors often fail to smooth the noise. These trade-offs and considerations in choosing priors highlight the challenges of shape reconstruction, especially when dealing with real-world data characterized by imperfections and complexities. Please refer to Figure 1.9.

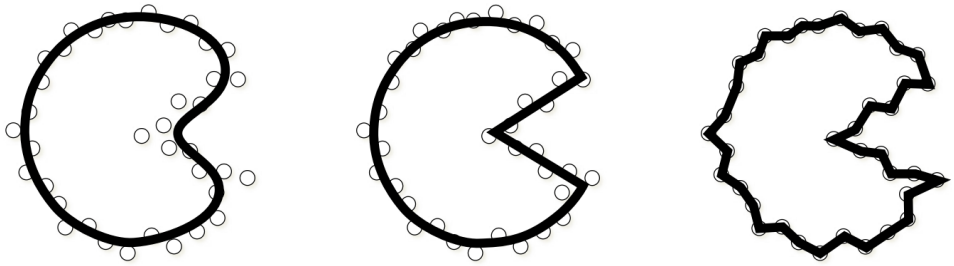


Figure 1.9: Priors for reconstruction methodologies. We identify three main priors: global smoothness, piecewise smoothness and linear interpolation.

Geometric fidelity. Geometric fidelity reflects how the reconstructed shape is close to the original shape or the input point cloud data. It pertains to the accuracy with which the reconstructed shape approximates the true geometry of the original shape. Achieving high geometric fidelity is crucial, especially in applications where precision is paramount, such as aerospace, medical imaging, and manufacturing. The challenge in assessing geometric fidelity arises from the fact that the true original shape is often unknown, and the input point cloud itself may contain imperfections and uncertainties. Therefore, measuring the geometric fidelity of the reconstruction becomes challenging, as there may be no reference shape for direct comparison.

Topology. Topology defines the connectivity and relationships within a shape, reflecting the number of connected components, boundaries, handles, and voids. In 3D shape reconstruction, maintaining the accurate preservation of these topological features is challenging, especially when the input point cloud is defect-laden. Some methods prioritize preserving the watertightness of the reconstructed shape, ensuring that there are no gaps or holes. Others focus on preserving non-manifold properties, which describe singular structures such as objects with multiple connected components or complex boundaries. Figure 1.10 depicts an example of a reconstructed hand with wrong topologies.

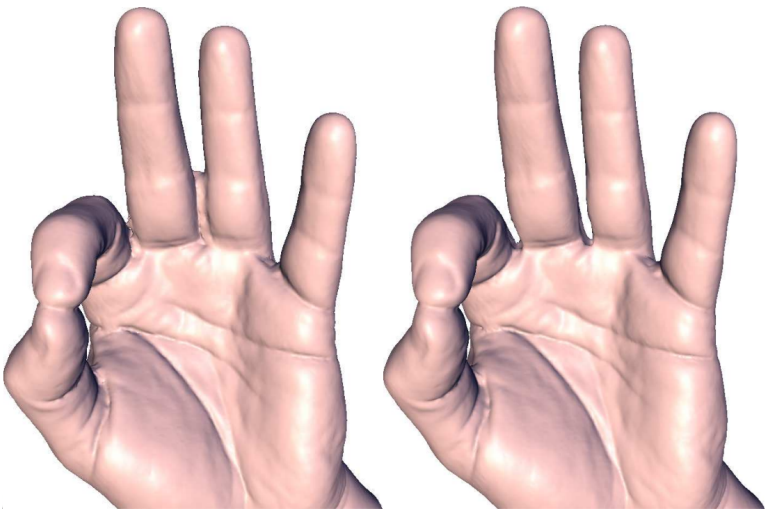


Figure 1.10: Topology issue for shape reconstruction reconstruction. Left: two fingers are connected with wrong topology. Right: two are fingers are separated with correct topology. Image taken from [SLS+07].

Memory usage and computation. Memory usage and computational demands present significant challenges in shape reconstruction. Handling extensive 3D point clouds can quickly deplete available memory resources, potentially leading to inefficiencies. Moreover, the computational requirements for processing large point clouds can result in extended processing times, reducing overall efficiency. Achieving a balance between efficient memory management and computational speed is crucial. While machine learning-based methods have gained popularity, they often struggle with memory and computation issues, making them less practical for industrial applications. Figure 1.11 provides a visual example of memory-intensive shape reconstruction, with the primary source of high memory usage being the 3D convolution operations on voxels.

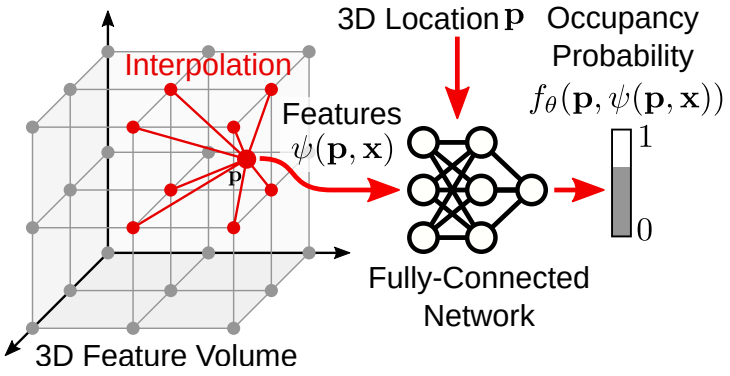


Figure 1.11: Memory-consuming reconstruction. Convolution on a voxel grid consumes large amounts of memory, thus lacking scalability for large-scale point clouds. Image taken from [PNM+20].

Automation. Making shape reconstruction automatic is a fundamental goal. While some methods can handle challenging topology cases with human intervention, such an interactive approach is labor-intensive and can produce results that vary based on different individuals' input, making it less reliable for large-scale or standardized applications. Figure 1.12 presents an example of interactive surface reconstruction, which requires users to provide hints to improve the reconstruction quality.

Generalization. Generalization in shape reconstruction refers to an algorithm's capacity to extend its reconstructive abilities beyond specific, limited examples and datasets. This is particularly significant in the context of learning-based methods.

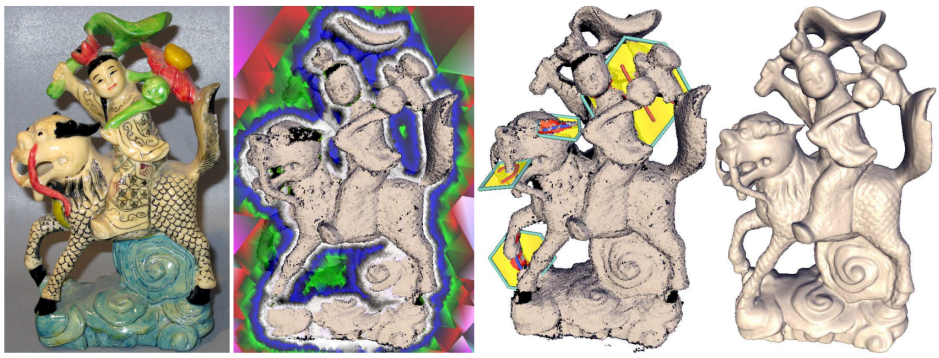


Figure 1.12: Interactive reconstruction. This approach requires users to provide scribbles for ambiguous areas. Image taken from [SLS+07].

In essence, a well-generalized shape reconstruction algorithm should exhibit proficiency in accurately reconstructing surfaces that extend beyond the scope of its training data. However, many learning-based approaches are trained on specific object categories, such as chairs or couches. While they may perform well within these predefined categories, they can fail when faced with shapes significantly differing from their training data. Figure 1.13 highlights an example of the limitation of generalization using DeepSDF [PFS+19]. This specific method is trained and tested on particular models, which restricts its applicability in industrial scenarios where shapes can vary significantly.

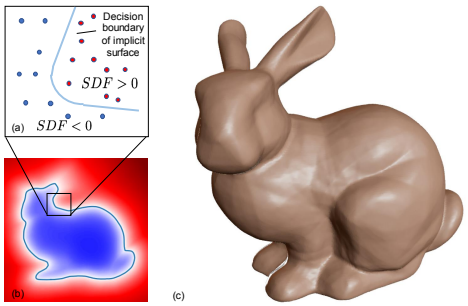


Figure 1.13: Non-general reconstruction. The learned SDF is constrained to specific shapes. Image taken from [PFS+19].

Other evaluation criteria. In addition to geometric fidelity and topological correctness, various other evaluation criteria are also crucial for shape reconstruction. These criteria encompass aspects such as visual fidelity, which assesses how closely

the reconstructed shape resembles the original from a visual standpoint. Additionally, some evaluations consider view-dependent variants, accounting for how the reconstructed shape appears from different viewpoints. These diverse evaluation criteria provide a comprehensive assessment of the reconstructed shapes, ensuring their suitability for various applications.

1.3 Contributions

We first provide a summary of our contributions to shape reconstruction. Specifically, we explored both explicit and implicit shape reconstruction. In terms of explicit reconstruction, we have concentrated on primitive segmentation. Our proposed method involves segmenting point clouds with guidance from Bézier decomposition, enhancing the genericity of shape recognition. In terms of implicit reconstruction, we have introduced a new mesh reconstruction technique designed to extract LFS-aware (local-feature-size-aware) triangle surface meshes directly from 3D point clouds.

1.3.1 Primitive Segmentation

Existing primitive segmentation methods commonly approach the task by identifying individual primitives from point clouds in a separate manner. These approaches imply that a distinct optimization process is required for each primitive type. Consequently, these methods lack the generality required to handle a wide range of primitive shapes efficiently. Recognizing this limitation, our work aims to overcome the constraints of common approaches by introducing a more general and adaptive solution. We provide a detailed discussion of those existing methods in Section 2.1.

Our first contribution is to define a general primitive type for the primitive segmentation task. We introduce BPNet, a new end-to-end deep learning framework designed to facilitate Bézier primitive segmentation on 3D point clouds. Unlike existing approaches constrained to finite shape categories because of distinct optimization, BPNet is inspired by the operation of Bézier decomposition techniques applied to NURBS models, which decompose different primitives into rational Bézier patches. We adapt and extend these principles to guide point cloud segmentation, making the Bézier decomposition learnable for point clouds and casting off the constraints of primitive types.

Specifically, our approach employs a joint optimization framework, enabling the concurrent learning of Bézier primitive segmentation and geometric fitting within a

cascaded deep learning architecture. Notably, we introduce a soft voting regularizer to enhance primitive segmentation and propose an auto-weight embedding module to cluster point features, augmenting the network's robustness and generality. Additionally, we introduce a reconstruction module that effectively processes multiple CAD models, each featuring different primitives, simultaneously.

To evaluate the efficacy of BPNet, we conducted extensive experiments on both synthetic ABC datasets and real-scan datasets. The results demonstrate superior segmentation performance compared to previous methods, accompanied by an improvement in inference speed. Furthermore, we demonstrate the generalizability of our method by training the model on the synthetic ABC datasets but testing with free-form points. Our model also yields reasonable segmentations on free-form 3D point sets with smoother boundaries compared to other baseline methods.

1.3.2 Surface Reconstruction

The primary goal in surface reconstruction is to ensure fidelity in reconstructing shapes. Many modern surface reconstruction algorithms place a strong emphasis on achieving high fidelity by closely fitting the reconstructed mesh to the input data. However, this main focus on fidelity can sometimes result in poor-quality meshes, particularly when contouring implicit surfaces using contouring methods like marching cubes or marching tetrahedrons, leading to non-isotropic and dense triangular meshes. Consequently, post-processing steps, such as remeshing, become necessary to convert these non-isotropic and dense meshes into isotropic ones with simplified complexity. Additionally, it is important for the mesh itself to be adaptive, meaning that it should use small triangles in regions with rich details. While post-processing can undoubtedly enhance mesh quality, it also introduces the potential risk of compromising the overall reconstruction quality. This gives rise to another objective in surface reconstruction: Is it possible to obtain an isotropic and adaptive mesh directly from surface reconstruction without the need for post-processing?

Our second contribution addresses the two above objectives altogether by contributing a method for reconstructing an isotropic surface triangle mesh directly from an unoriented 3D point cloud, without remeshing. The key novelty of our approach lies into its adaptability to the local feature size (LFS), resulting in a density that aligns with the local details of the underlying data. Instead of following the conventional path of dense reconstruction followed by remeshing, our method simultaneously reconstructs both an implicit function and an LFS-aware mesh sizing function. These two components cooperate to produce the final LFS-aware

reconstructed mesh, which is isotropic with adaptive sizes.

Determining the LFS is a critical step for our surface reconstruction method. We achieve LFS estimation by considering the minimum of two geometric properties: the local curvature radius and half of the shape diameter. We derive the local curvature from a polynomial surface adjusted via least squares fitting (also referred to as jet fitting), while we estimate the shape diameter by our proposed Lipschitz-guided dichotomic search. We then construct the implicit function in three main steps: creating a tetrahedron multi-domain mesh from an unsigned distance function, refining the multi-domain with data fitting, and generating a robust signed distance function.

Finally, we utilize the Delaunay refinement meshing approach to obtain the final LFS-aware mesh. Specifically, we derive a mesh sizing function, which controls the density of the final mesh, from the estimated LFS. We then employ the sizing function with the implicit function to mesh the zero-level set of the implicit function using Delaunay refinement. The distinctive feature of our approach is its ability to generate isotropic meshes from 3D point clouds while maintaining an LFS-aware density. Our experiments demonstrate the method's robustness in handling defects such as noise, outliers, or missing data. Furthermore, we provide experiments to show that our method can reconstruct shapes with complicated topologies, such as high genus.

1.4 Publications

This thesis is supported by a publication and a submission:

- Rao Fu, Cheng Wen, Qian Li, Xiao Xiao, Pierre Alliez. BPNet: Bézier Primitive Segmentation on 3D Point Clouds. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023.

1.5 Outlines

This thesis contributes to the shape reconstruction problem by approaching it from the perspectives of primitive segmentation and surface mesh reconstruction. The thesis is structured as follows:

1. Chapter 1 provides an overview of shape reconstruction with its applications and challenges. It also discusses 3D point cloud acquisition that will challenge the shape reconstruction task. Finally, this chapter outlines our contributions.

2. Chapter 2 presents a thorough survey of the existing literature and related work in the field of shape reconstruction, with a primary focus on primitive segmentation and surface reconstruction.

3. Chapter 3 delves into our contributions and methodologies concerning primitive segmentation, accompanied by extensive experiments and comparisons in this domain.

4. Chapter 4 investigates our contributions and methodologies related to surface reconstruction, providing an in-depth analysis of the experiments conducted in this area.

5. Chapter 5 summarizes the key findings and contributions of the thesis and outlines potential directions for future research.

CHAPTER 2

Literature Reviews

This chapter reviews the previous work related to the two main contributions of the thesis: primitive segmentation and surface reconstruction.

2.1 Primitive Segmentation

We now review the topic of primitive segmentation for 3D point clouds. We first review canonical primitive detection and recent advances on this topic using deep learning. Besides, we also review deep learning approaches devised to address other tasks that are made learnable from 3D point clouds. Figure 2.1 illustrates the primitive detection process from different sources of input.

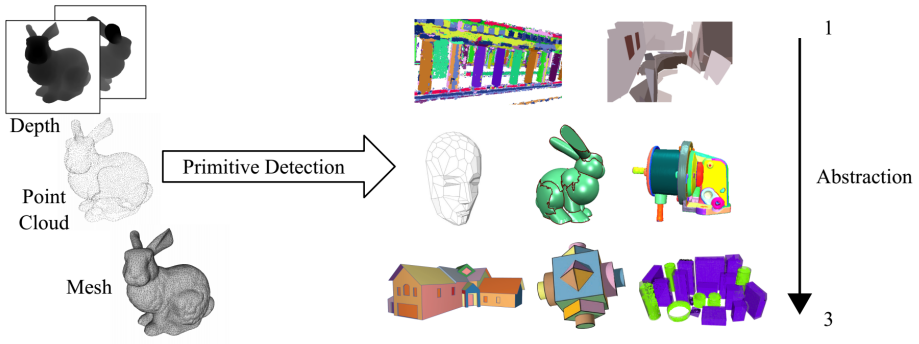


Figure 2.1: Primitive detection. The detection of primitives can be applied to different sources of input sources such as depth images, 3D point clouds or surface meshes. Image taken from [KYZB19].

2.1.1 Primitive Detection

In our context, primitive segmentation refers to the search and approximation of geometric primitives from unorganized 3D point clouds. The said primitives can be canonical geometric primitives, such as planes or spheres, or parametric surface

patches, such as Bézier patches, B-Splines, or NURBS (Non-Uniform Rational B-Splines). We can classify primitive segmentation methods into two lines of approaches: geometric optimization and machine learning.

Geometric Optimization-based. Popular geometric optimization-based methods include RANSAC [FB81, SWK07], region growing [MLM01], and Hough transforms [RDvdHV07]. We refer to [KYZB19] for a comprehensive survey, and summarize the different properties in Figure 2.2. One limitation of geometric optimization-based methods is that they require strong prior knowledge and are hence sensitive to parameters. In addition, most geometric optimization-based methods only support simple canonical geometric primitives and are thus unable to deal with complex free-form shapes.

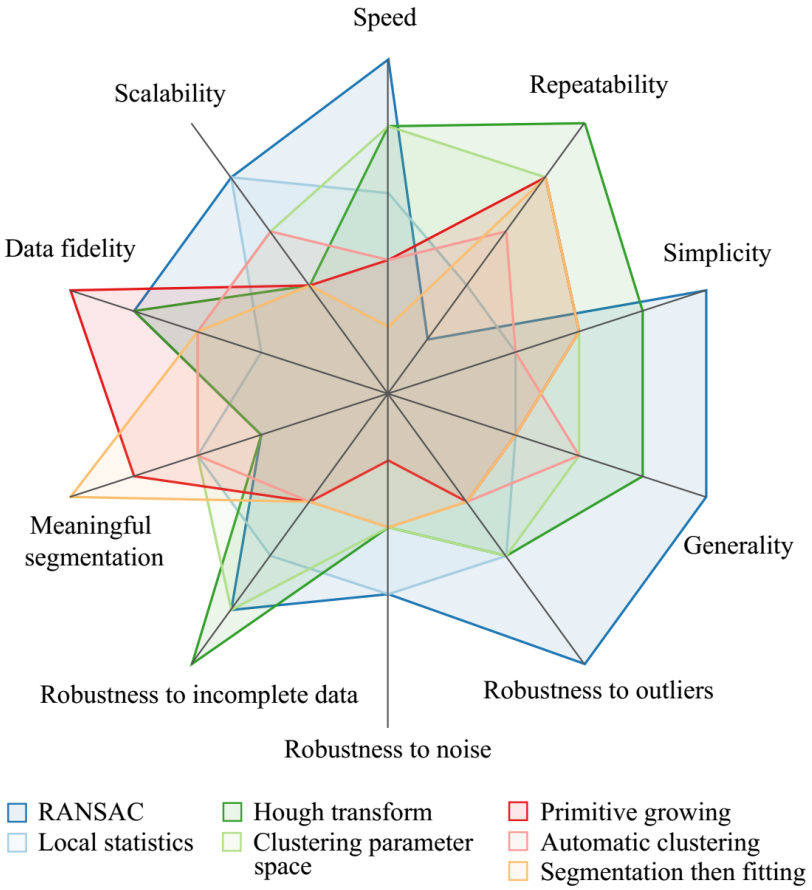


Figure 2.2: Qualitative comparison of different geometric optimization-based methods. Image taken from [KYZB19].

Learning-based. Recent approaches utilize deep neural networks to alleviate the limitations of geometric optimization-based primitive detection. For example, some approaches apply deep learning to identify cuboids [ZYY+17, TSG+17] or ellipsoids [SDR+22] from the input 3D point cloud. However, this type of method only supports a single type of primitive. One noticeable algorithm is the supervised learning approach referred to as the Supervised Primitive Fitting Network (SPFN) [LSD+19]. It detects a wider repertoire of primitives, including planes, spheres, cylinders and cones. Figure 2.3 presents the pipeline of the SPFN network. Specifically, SPFN relies on the PointNet++ learning framework [QYSG17] and takes a 3D point cloud as input. It outputs three per-point properties: point-to-primitive membership, normals, and associated primitive type. As the index of the ground truth primitives and the prediction is mismatched, a reordering step is applied to align the index. Then, the output primitive parameters are estimated from the point properties in the model estimations step.

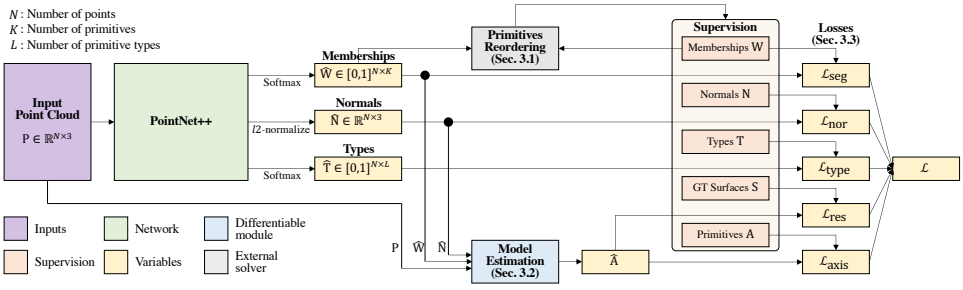


Figure 2.3: Overview of the SPFN pipeline. The loss is defined as the sum of five different loss terms. Image taken from [LSD+19].

As SPFN only supports four basic canonical primitive types, a later work denoted ParSeNet [SLM+20] takes into consideration open and closed B-Spline surface, thus supporting a richer class compared to SPFN. Figure 2.4 depicts the pipeline of the ParSeNet network. ParSeNet consists of two modules: (1) The decomposition module takes a 3D point cloud (with optional normals) and decomposes it into segments labeled by primitive type. (2) The fitting module predicts parameters of a primitive that best approximates each segment. It includes a SplineNet to fit B-Spline patches. The two modules are jointly trained end-to-end. An optional postprocess module refines the output.

Based on the primitive type supported by ParSeNet, the HPNet [YY21] leverages hybrid representations that combine one learned semantic descriptor, two spectral descriptors derived from predicted geometric parameters, as well as an adjacency

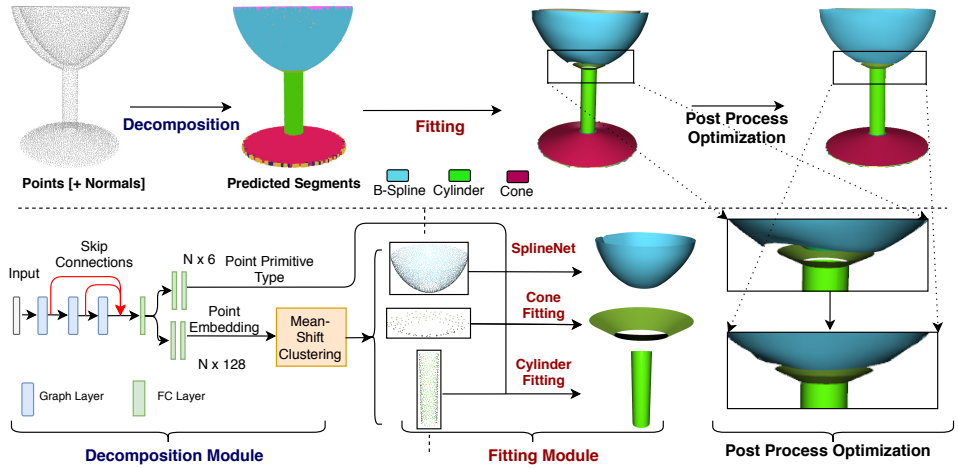


Figure 2.4: Overview of the ParSeNet pipeline. It has two modules: decomposition and fitting. Optionally, a post-processing optimization is applied to refine the output. Image taken from [SLM+20].

matrix that encodes sharp edges to improve the primitive segmentation results. Figure 2.5 depicts the pipeline of the ParSeNet network. Specifically, it has three modules: (1) The dense descriptor module takes a point cloud and optional normal as input and outputs a semantic feature descriptor, a type indicator vector, and a shape parameter vector. (2) The spectral embedding module takes dense descriptors as input and builds geometric consistency matrix A_c and smoothness matrix A_s. Then it outputs consistency feature U_c and smoothness feature U_s. (3) The clustering module combines three features with adaptive weights and use mean-shift clustering to output the segmentation result.

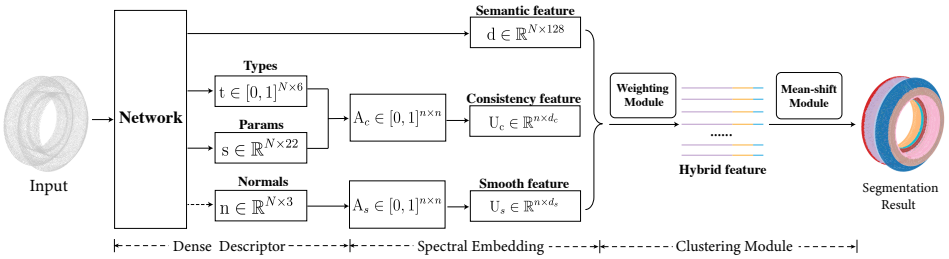


Figure 2.5: Overview of the HPNet pipeline. It consists of three modules: dense descriptor, spectral embedding, and clustering. Image taken from [YY21].

Nevertheless, all above methods treat different primitive types separately and are thus insufficiently generic. Such an insufficient genericity for learning-based

primitive segmentation methods makes them unsuitable for batch operations, thus suffering long inference times. Deep learning-based methods are less sensitive to parameters but often support a limited repertoire of primitives.

2.1.2 Instance Segmentation

Instance segmentation is related to primitive segmentation, where all instances are identified separately. Some techniques of instance segmentation can be directly applied to primitive segmentation. Furthermore, instance segmentation is more challenging than semantic segmentation as the number of instances is not known a priori. Points assigned to the same instance should fall into the same semantic class. We distinguish between two types of methods: proposal-based [YZW+19, YWC+19, EBF+20] and proposal-free methods [WYHN18, JZS+20, HZS21]. On the one hand, proposal-based methods utilize an object-detection module and usually learn an instance mask for prediction. On the other hand, proposal-free methods tackle the problem as a clustering step after semantic segmentation. We refer to a recent comprehensive survey [GWH+20] and show a chronological overview of the most relevant deep learning-based 3D instance methods in Figure 2.6. The significant difference between instance segmentation and primitive segmentation is that instance segmentation only focuses on partitioning individual objects where primitive fitting is absent.

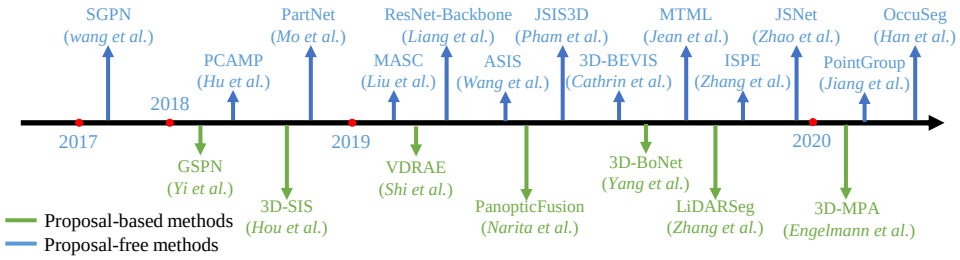


Figure 2.6: Chronological overview of relevant deep learning-based 3D instance segmentation methods. Image taken from [GWH+20].

2.1.3 Patch-based Representation

We also briefly review the patch-based representations, which are also related to some learning techniques for the task of patch-based primitive. Patch-based representations refer to finding a mapping from a 2D patch to a 3D surface. AtlasNet [GFK+18] represents a 3D shape as a collection of learnable parameterizations,

which transform a set of 2D squares to the surface, covering it in a way similar to placing strips of paper on a shape to form a papier-mâché. FoldingNet [YFST18] pioneered the idea of learning a parametric mapping from a 2D patch to a 3D surface. Deng et al. [DBSF20] propose an approach that explicitly encourages global consistency of the local mappings by introducing new loss terms that favor surface normal consistency and minimize stitching error. Bednarik et al. [BPG+20] reconstruct shapes by learning differentiable surface representations. These methods learn a parametric 2D mapping by minimizing the Chamfer distance [FSG17]. One limitation of using the Chamfer distance is that it is not differentiable when using the nearest neighbor to find matched pairs. Unlike previous works, we learn a *uv* mapping with the supervision of *uv* labels directly obtained from Bézier decomposition instead of using the Chamfer distance. Please refer to chapter 3 for details. Direct learning *uv* parameters enables us to re-evaluate points from the proposed generalized Bézier primitives, making the training process differentiable and supporting batch operations.

2.2 Surface Reconstruction

We now provide a detailed review of surface reconstruction from 3D point clouds. As our surface reconstruction approach involves three main components: estimation of the local feature size, mesh-based surface reconstruction, iso-surfacing, and remeshing, we review each component individually.

2.2.1 Estimation of Local Feature Size

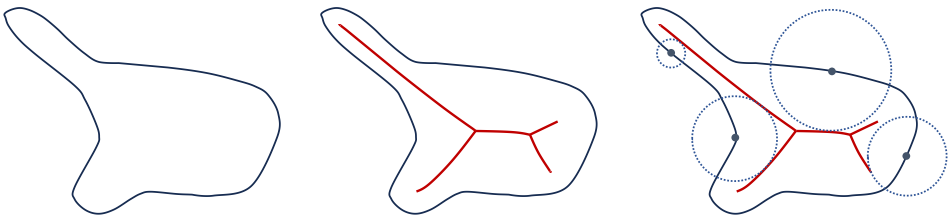


Figure 2.7: Local feature size for a smooth manifold with its medial axis. Left: the original shape (black). Middle: the shape with its medial axis (red). Right: The local feature size is the distance from the query point on the shape to its closest point on the medial axis.

Surface reconstruction may be seen as the analogous of signal reconstruction.

The goal is to infer a continuous surface or shape from discrete 3D point cloud data, mirroring the task of recovering continuous signals from discrete data points. Aiming at defining the analogous of signal frequency but for 3D shapes, Amenta et al. [AB98] defined the concept of local feature size, which is used to analyze the sampling conditions. Mathematically, the local feature size (LFS) on a 3D shape is the distance from a query point to its closest point on the shape's medial axis. The reach of a shape refers to the minimum of the LFS [AKC+19]. Figure 2.7 shows a smooth shape with the medial axis and the local feature size. There are two main lines of approaches to estimating the LFS: indirect and direct. Indirect methods commonly involve computing the medial axis [TDS+16] defined as the locus of centers of spheres that touch the shape's boundary in two or more faces. Amenta proved that the Voronoi poles provide a good estimation of the medial axis if the point set used to generate the corresponding Voronoi diagram is noise-free and satisfies the ε-sampling conditions [AB98]. Several approaches have been proposed based on this idea [AB98, ACK01, BC01, DS06], which estimate the medial axis and then estimate the LFS by finding the nearest valid Voronoi pole. Figure 2.8 depicts an example of medial axis estimated from the Voronoi poles.

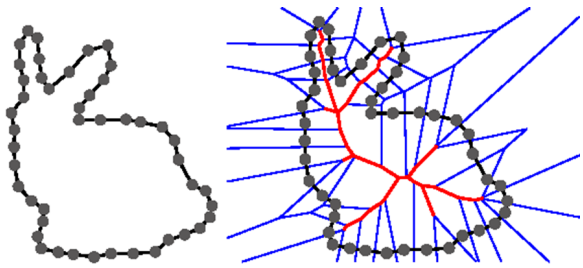


Figure 2.8: Point samples and the medial axis in 2D. Left: the point cloud is sampled from a shape. Right: the Voronoi diagram (blue) estimates the medial axis (red). Image taken from [ZSC+14].

The main limitation of these methods is sensitivity to noise and non-uniform sampling, which significantly affect the accuracy of the estimated medial axis. Recent approaches [RAV+19, LBKP21] attempt to gain robustness by computing a direct medial axis in an optimization scheme, free of the construction of the Voronoi diagram. However, oriented normals are required as a prior. Direct methods [SSCO08, RNDA13] avoid estimating the medial axis explicitly. Instead, they estimate the shape's diameter function from the input shape.

While direct methods are less sensitive to noise, estimating the shape diameter

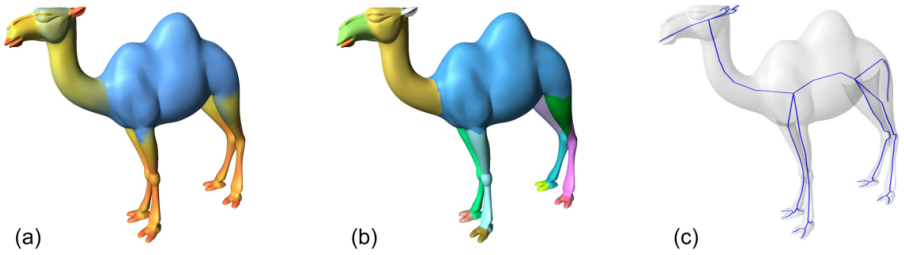


Figure 2.9: Illustration of the shape diameter function. The local shape diameter function provides a link from the surface of the mesh to its volume. (a) A color map of the function values on the mesh from red (narrow diameter) to blue (wide diameter). The SDF is related to the medial axis transform and can be used to find a consistent hierarchical partitioning of objects (b) and to create simple robust skeletons (c). Image taken from [SSCO08].

function on a 3D point cloud is challenging because finding the antipodal point given a ray cast from an input point is ill-posed. Furthermore, the shape diameter function only measures the thickness of the shape, while LFS also captures separation and curvature. Departing from previous approaches, our LFS estimation approach relies on both the shape diameter function and curvature estimation [CP05]. The shape diameter function is estimated through analyzing the unsigned distance function to the input 3D point cloud.

2.2.2 Mesh-based Surface Reconstruction

Many mesh-based surface reconstruction approaches have been proposed over the years, broadly categorized into five main categories: Delaunay-based, implicit-based, primitive-based, learning-based and hybrid methods. We refer to comprehensive surveys for a detailed overview [BTS+17, HWW+22b]. Figure 2.10 provides an overview of different reconstruction algorithms.

Delaunay-based. Delaunay-based methods utilize the Delaunay triangulation or Voronoi diagram to generate a mesh that interpolates the input points. The popular crust and power crust approaches leverage the Voronoi diagram to compute the medial axis of a shape and then construct a mesh from the Delaunay triangulation of the crust or power diagram [AB98, ACK01]. Figure 2.11 illustrates the construction of power crust. The power crust first constructs the Voronoi diagram from the input point clouds. Then, the Voronoi poles will be selected. In two dimensions,

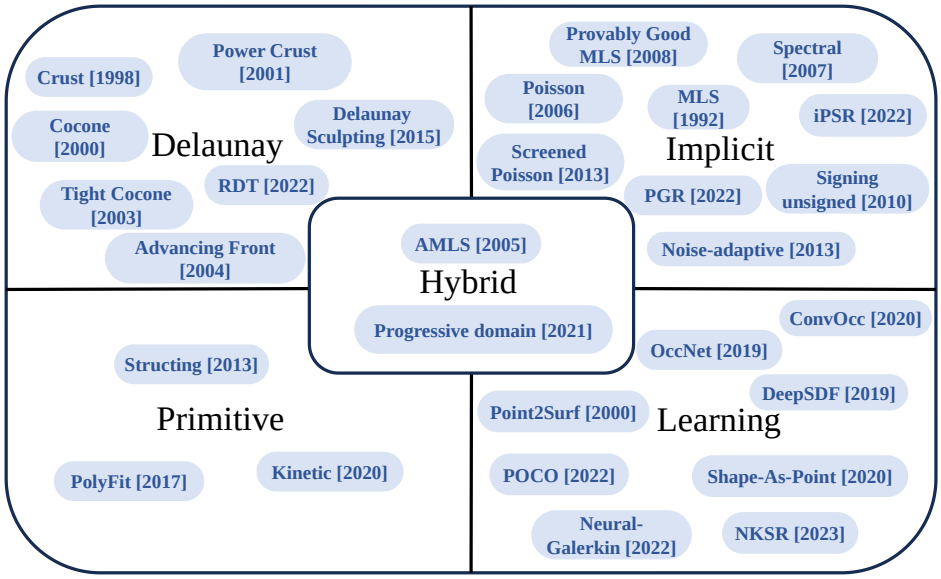


Figure 2.10: Overview of different reconstruction algorithms with their publication date. We classify reconstruction as Delaunay-based, primitive-based, implicit-based, learning-based, and hybrid.

the power crust selects all Voronoi vertices as poles; in three dimensions, the power crust selects only certain ones near the medial axis. Finally, the surface is the boundary of the inner and outer power diagram cells. The cocone algorithm

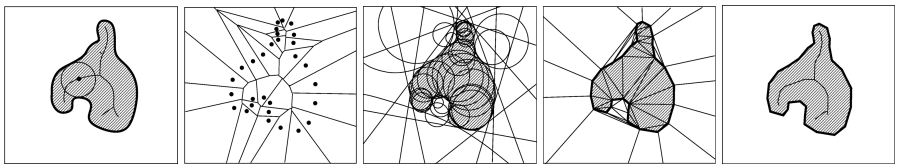


Figure 2.11: Two-dimensional example of power crust construction. (a) An object (shaded) with its medial axis. (b) The Voronoi diagram of a sample of points from the object boundary. (c) The sets of inner (shaded) and outer polar balls. Outer polar balls with centers at infinity degenerate to halfspaces on the convex hull. (d) The power diagram cells of the poles. In two dimensions this is similar to the Delaunay triangulation of the samples, but not in three dimensions. (e) The power crust and the inner portion of the power shape. Image taken from [ACK01].

[ACDL00] constructs a surface triangle mesh from the so-called cocones at each

sample point. Later, the tight cocone algorithm (Figure 2.12) [DG03] extends the

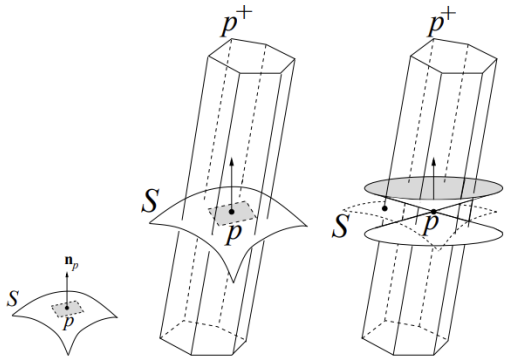


Figure 2.12: Illustration of the tight cocone: A Voronoi cell V_p is elongated along the normal n_p. The pole vector $p^+ - p$ approximates n_p. The cocone C_p is the region in V_p between the two cones at p. Image taken from [DG03].

cocone algorithm by introducing a tightness criterion that ensures that the mesh is close to the input 3D point cloud. The advancing front algorithm [CSD04] (Figure 2.13) constructs a mesh by greedily advancing a front of Delaunay triangles. The

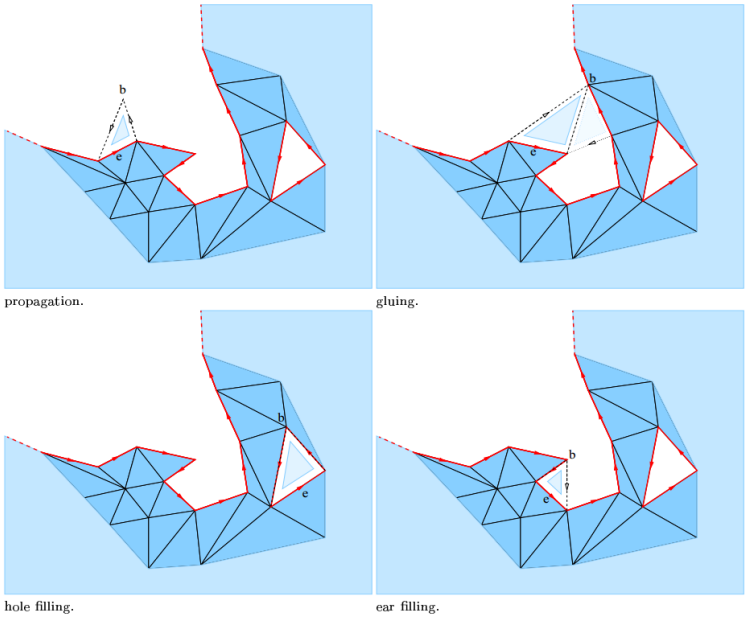


Figure 2.13: Valid candidates of the advancing front algorithm. Image taken from [CSD04].

Delaunay sculpting algorithm [PM15] builds a 3D triangulation (tetrahedralization) in the convex hull and then distinguishes interior from exterior tetrahedral elements. The restricted Delaunay triangulation explicit reconstruction algorithm [WWX+22] starts from an initial simple surface mesh and alternately performs Filmsticking and Sculpting with the initial mesh to obtain the final mesh. While Delaunay-based methods provide provable guarantees under certain conditions, they often require a dense sampling and are sensitive to noise and outliers.

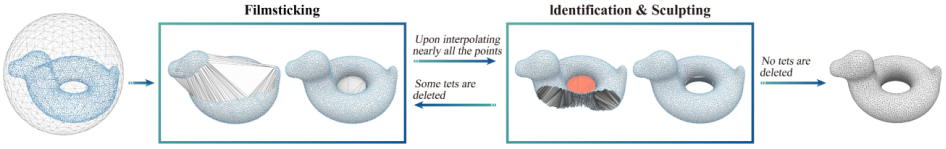


Figure 2.14: Reconstruction pipeline from [WWX+22]. From left to right: A 3D point cloud and the bounding sphere as an initial (guiding) surface, a Filmsticking step to make the guiding surface interpolate nearly all of the points, a Sculpting step to chip off tetrahedral elements that are identified as unnecessary, and return the guiding surface as the result if it remains unchanged in the sculpting step. Image taken from [WWX+22].

Implicit-based. Implicit-based methods are devised to represent the reconstructed surface as an isolevel of an implicit function. The said function is often defined on a discretized 3D domain such as an octree, or defined using smoothness priors like radial basis functions (RBF), moving least squares (MLS) or kernel regression. Some implicit-based approaches require oriented normals as input, while others do not. More specifically, MLS-based methods [HDD+92, Kol08] (Figure 2.15) rely on oriented normals. Poisson surface reconstruction approaches [KBH06, KH13] (Figure 2.16) diffuse the input oriented normals on an octree before solving for a Poisson equation. Some approaches deduce a signed implicit function from a unoriented point set [MDGD+10, GCSA13], and a spectral approach solves a generalized eigenvalue problem to obtain a signed function from unoriented normals and covariance matrices [ACSTD07] (Figure 2.17). Some modified Poisson surface reconstruction approaches can run without normal orientation. For example, the PGR algorithm [LXSW22] considers normals and surface elements in the Gauss formula as unknowns and optimizes the parametric function space. The recent iPSR algorithm [HWW+22a] starts from random normals and iterates the screened Poisson reconstruction to update signed normals derived from the current mesh.

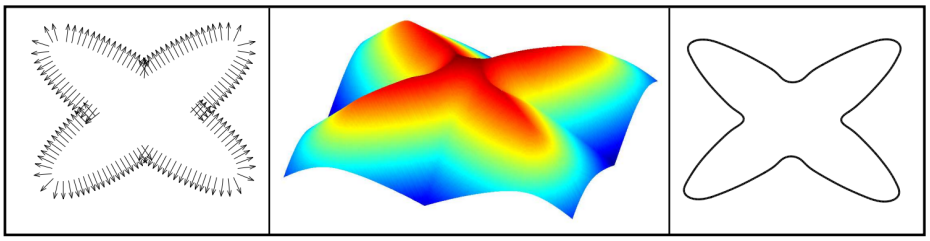


Figure 2.15: Illustration of MLS surface. Left, a set of points with outside normals. Center, the implicit function built by the MLS algorithm [Kol08] from the oriented points. Right, the reconstructed curve which is the zero set of the implicit function. Image taken from [Kol08].

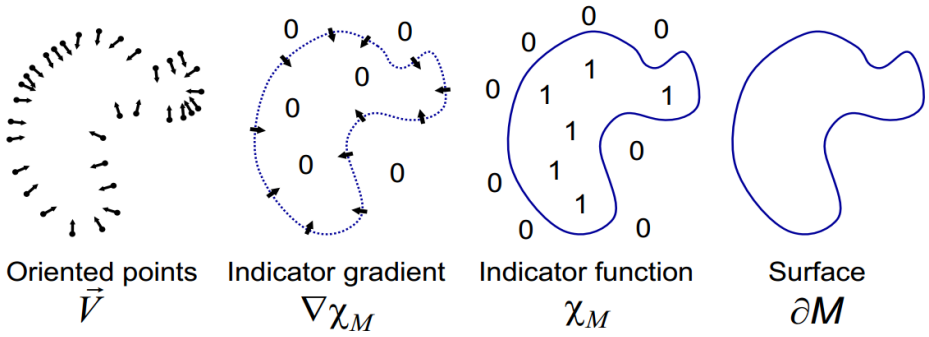


Figure 2.16: Intuitive illustration of Poisson reconstruction in 2D. Image taken from [KBH06].

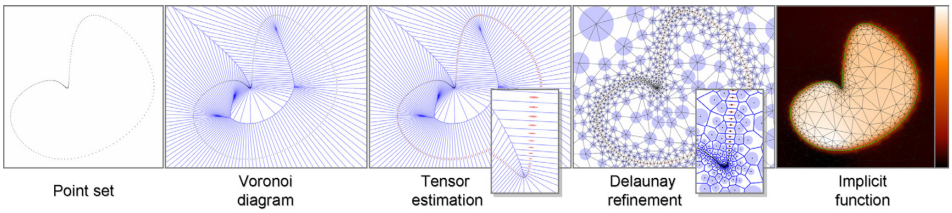


Figure 2.17: Reconstruction process from [ACSTD07]. From left to right: input point set; its Voronoi diagram; covariance matrices of the cells shown as (rescaled) ellipses; Steiner points added through Delaunay refinement (isotropic tensors are assigned to Steiner points); piecewise linear function f (solution of a generalized eigenvalue problem) that best fits the input data, with the reconstructed curve (iso-surfacing of f). Image taken from [ACSTD07].

Figure 2.18 illustrates the pipeline of the iPSR algorithm. While these methods can remove the need for oriented normals, they do not consider the local topology. Implicit-based methods are often more robust to noise than Delaunay-based methods

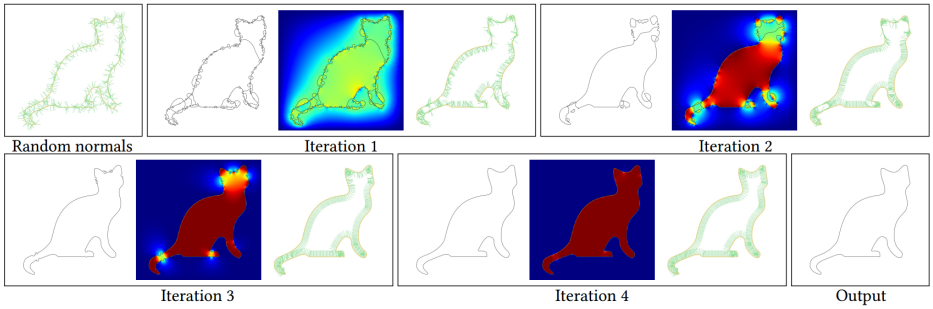


Figure 2.18: iPSR pipeline on a 2D point set. Image taken from [HWW+22a].

but may produce a dense mesh. They usually require a fine discretization of the 3D domain to capture the surface details, resulting in a complex uniform output mesh, even on flat areas.

Primitive-based. Primitive-based reconstruction is related to primitive extraction and primitive assembling. The structuring algorithm [LA13] first consolidates the point clouds and then extracts the surface by solving a graph-cut problem formulated on the 3D Delaunay triangulation. The PolyFit algorithm [NW17] proceeds by detecting the intersection of multiple planes and seeks for an appropriate combination of different planes to obtain a manifold polygonal surface. The Kinetic algorithm [BL20] (Figure 2.19) leverages a plane-based kinetic data structure for partitioning the space into convex polyhedra and extracting a watertight mesh from the partition via solving for min-cut optimization problem. Specifically, the Kinetic

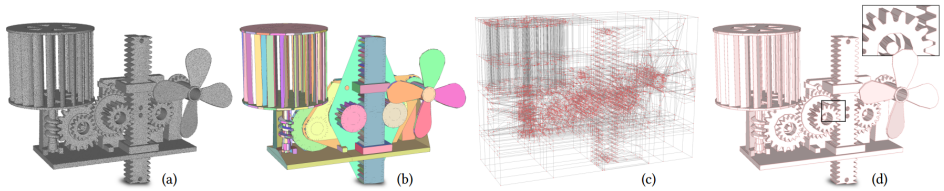


Figure 2.19: Overview of the Kinetic shape reconstruction. Image taken from [BL20].

algorithm takes as input a 3D point cloud with oriented normals and a configuration of convex polygons that approximate the object. Convex polygons can be obtained

through shape detection as the convex hulls of inlier points are projected onto the shape. Then, the algorithm runs a kinetic partitioning to decompose the 3D bounding box of the object from the convex polygons. Finally, a concise polygonal mesh is extracted by labeling each polyhedron as inside or outside the object.

Learning-based. The learning-based reconstruction methods draw lots of attention in recent years. The occupancy network and its convolutional variant [MON+19, PNM+20] predict an occupancy probability for the grid and then extract the mesh based on the said prediction. Point2Surf [EGO+20] learns an implicit signed distance function from a local patch of the point cloud. The Shape-As-Points approach [PJL+21] pioneered a differentiable formulation of the Poisson surface reconstruction. The POCO approach [BM22] uses convolutions and computes latent vectors at each point to deduce an occupancy for large-scale point clouds. The neural kernel reconstruction methods [HCH22, HGA+23] (Figure 2.20) solve an implicit function via optimizing with data-driven kernel functions. Like Poisson surface reconstruction, they solved an implicit function based on the octree with the data-driven neural kernels. They thus gain scalability and generalizability because the matrix for solving the implicit function is sparse, and only the kernel function is data-driven. As illustrated in Figure 2.20, NKSR [HGA+23] takes an oriented point cloud as input and predicts a sparse hierarchy of voxel grids containing features as well as normals in each voxel. It then constructs a sparse linear system and solves for a set of coefficients $\boldsymbol{\alpha}$ per voxel. The linear system corresponds to the gram matrix arising from a kernel, which depends on the predicted features, denoted by $\mathbf{L}$ and $\boldsymbol{v}$ in this figure. The final reconstructed surface is the zero level of the linear combination of the learned kernel basis functions.

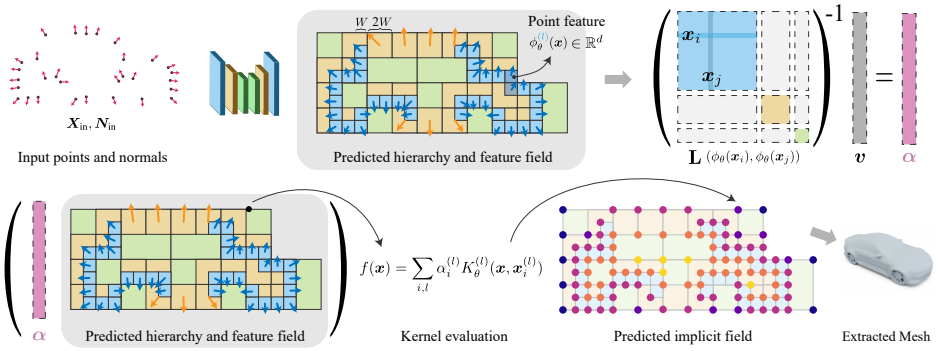


Figure 2.20: Overview of the NKSR approach [HGA+23]. Image taken from [HGA+23].

Hybrid-based. Some hybrid methods combine Delaunay 3D triangulations and implicit functions. The AMLS approach [DS05] projects input points onto an LFS-adaptive implicit surface and then leverages the tight cocone to interpolate the projected points. A recent progressive approach interleaves refinement of a triangulated 3D domain with implicit mesh-based solvers [ZAB+21] (Figure 2.21). Thus, the discretized domain is adapted near the isosurface and optimized to improve both the solver conditioning and the quality of the output surface mesh contoured via marching tetrahedra.

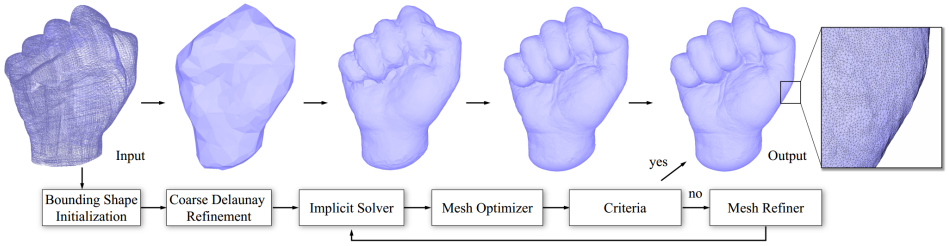


Figure 2.21: Overview of the recent progressive implicit approach [ZAB+21]. The domain boundary is initialized by a loose sphere bounding the input 3D point sample, refined by coarse Delaunay refinement. The algorithm then iterates through the solver, optimizer and refiner. The isosurface, contoured by marching tetrahedra, is generated as output, once user-specified criteria are met.

2.2.3 Iso-surfacing

Iso-surfacing, also referred to as iso-surface extraction, consists in extracting a sublevel from a given scalar field. Marching cubes [LC87] with its variants [Nie04, SW04] can extract surface triangle meshes on a regular voxel grid or an octree. The marching tetrahedra approach [DK91] simplifies the marching cubes algorithm by replacing the voxels with tetrahedron cells. In order to preserve the sharp features, the dual contouring approach [JLSW02] is proposed to optimize vertex locations inside the individual cells by minimizing a quadratic error function, i.e., the sum of squared distances to the tangent planes. In order to generate isotropic triangle surface meshes as output, GradNorm and its variant [HT20, ZZW+23] first tile the 3D space with fixed or adaptive tetrahedra before extracting an isotropic mesh. Besides, neural versions of marching cubes and dual contouring [LDG18, CZ21, CTFZ22] have been proposed as differentiable mesher. Specifically, deep marching cubes [LDG18] make the meshing procedure differential by introducing point-to-mesh distance

and curvature losses. Neural marching cubes [CZ21] re-design the tessellation templates to preserve sharp features and develop a new neural network to learn the vertex positions and mesh topologies. Neural dual contouring [CTFZ22] extends the neural marching cubes method to preserve better sharp features by incorporating the traditional dual contouring method. Figure 2.22 illustrates the difference between dual contouring and marching cubes.

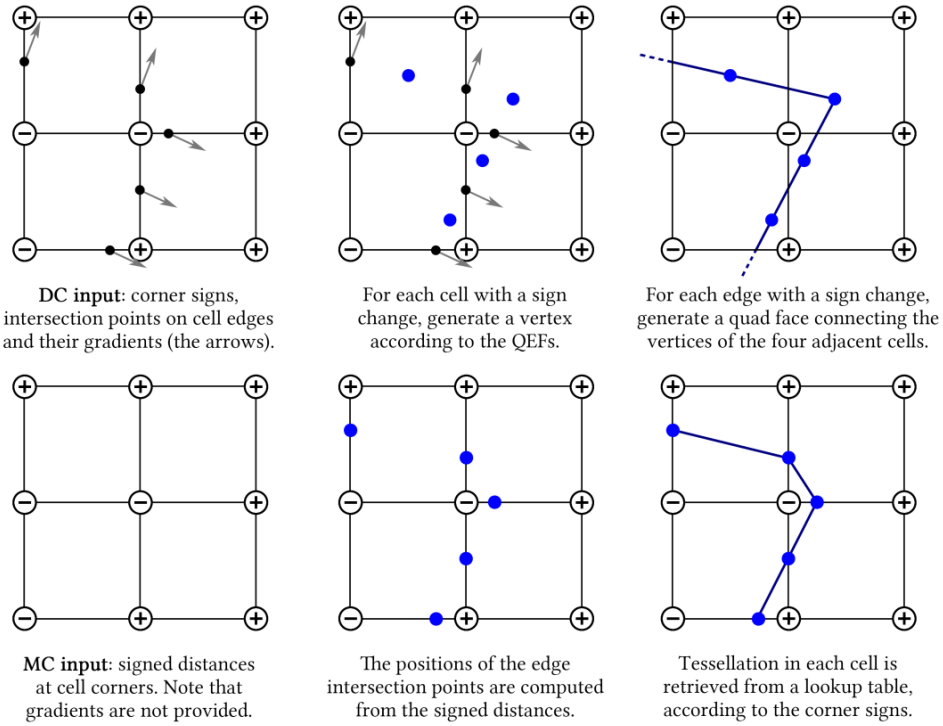


Figure 2.22: Dual Contouring (DC) vs. Marching Cubes (MC) – depicted in 2D on different input points sampled from the same underlying shape, DC (top) reconstructs sharp features (as intersections between faces, see the top-right cell), while MC (bottom) does not. Image taken from [CTFZ22].

2.2.4 Remeshing

Remeshing is used to improve mesh quality in terms of vertex sampling, regularity, and shape of its elements [AUGA08]. An example of uniform remeshing is provided in Figure 2.23.

Remeshing involves computing a new mesh approximating the original mesh while meeting specific quality requirements. It proceeds either in a parameter

Figure 2.23: Uniform remeshing of the Digital Michelangelo David model. Left: the mesh extracted by marching cubes contains many badly-shaped triangles. Right: the mesh becomes isotropic (well-shaped) after remeshing. Image taken from [AUGA08].

space or directly on the input mesh. Fewer remeshing approaches[SG03, SLC+19, NLG15] generate output meshes that are adapted to the local curvature or estimated local feature size with an estimation process applied to the input mesh. Departing from these methods, our method estimates LFS directly on the input 3D point cloud. LFS is then used for iso-surfacing the implicit function via Delaunay refinement, resulting in a lower mesh complexity while preserving more details. Chapter 4 provides a detailed explanation with rich experiments.

CHAPTER 3

Bézier Primitive Segmentation

3.1 Introduction

Structuring and abstracting 3D point clouds via segmentation is a prerequisite for various computer vision and 3D modeling applications. Many approaches have been proposed for semantic segmentation, but the finite set of semantic classes limits their applicability. 3D instance-level segmentation and shape detection are much more demanding, while this literature lags far behind its semantic segmentation counterpart. Finding a generalized way to decompose point clouds is essential. For example, man-made objects can be decomposed into canonical primitives such as planes, spheres, and cylinders, which are helpful for visualization and editing. However, the limited types of canonical primitives are insufficient to describe objects' geometry in real-world tasks. We are looking for a generalized way of decomposing point clouds. The task of decomposing point clouds into different geometric primitives with corresponding parameters is referred to as *parametric primitive segmentation*. Parametric primitive segmentation is more reasonable than semantic instance segmentation for individual 3D objects, which unifies the 3D objects in the parametric space instead of forming artificially defined parts. However, the task is quite challenging as 1) there is no exhaustive repertoire of canonical geometric primitives, 2) the number of primitives and points belonging to that primitive may significantly vary, and 3) points assigned to the same primitive should belong to the same type of primitive.

Inspired by the fact that Bézier decomposition, where NURBS models can be divided into canonical geometric primitives (plane, sphere, cone, cylinder, etc.) and parametric surfaces into rational Bézier patches, we propose to learn Bézier decomposition on 3D point clouds. We focus on segmenting point clouds sampled from individual objects, such as CAD models. Departing from previous primitive segmentation, we generalize different primitive types to Bézier primitives, making them suitable for end-to-end and batch training. To the best of our knowledge, our method is the only work to learn Bézier decomposition on point clouds. Specifically,

we introduce a novel soft voting regularizer for the relaxed intersection over union (IOU) loss, improving our primitive segmentation results. We also design a new auto-weight embedding module to cluster point features which is free of iterations, making the network robust to real scan data and work for axis-symmetric free-form point clouds. Besides, we propose a new reconstruction module where we succeed in using a generalized formula to evaluate points on different primitive types, enabling our training process to be fully differential and compatible with batch operations. Last but not least, we conduct extensive experiments to demonstrate that our method can work on the free-form point clouds and real scan data even if we only train our model on the ABC dataset. Furthermore, we present one application of Bézier primitive segmentation to reconstruct the full Bézier model while preserving the sharp features.

3.2 Method

3.2.1 Preliminaries

We start with a brief introduction of the Bézier decomposition. A rational Bézier surface patch is a parametric surface defined by a set of weighted control points. Given a pair of uv parameters (u,v), the corresponding 3D point coordinates (x',y',z') for a degree m-by-n rational Bézier patch are defined as following:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \frac{\sum_{m_i=0}^{m}\sum_{n_i=0}^{n} B_m^{m_i}(u)B_n^{n_i}(v)\mathbf{c}_{m_in_i}(c_w)_{m_in_i}}{\sum_{m_i=0}^{m}\sum_{n_i=0}^{n} B_m^{m_i}(u)B_n^{n_i}(v)(c_w)_{m_in_i}}, \tag{3.1}$$

where $B_d^l(t) = \binom{d}{l}t^l(1-t)^{d-l}$ $(0 \le l \le d)$ denotes the Bernstein basis function for degree d, t is u or v parameter in UV space, $\mathbf{c}_{m_in_i}$ denotes the control point coordinates and $(c_w)_{m_in_i}$ denotes its weight. For simplicity, we use $\mathbf{R}_{kmn}(u,v)$ to denote this function for patch k. Note that the number of control points in U or V direction must equal $d+1$.

Bézier decomposition [PT96, BSEH11] refers to splitting NURBS models into piecewise rational Bézier surface patches. More specifically, decomposition is accomplished by repeating all interior knots of a knot vector until their multiplicity equates d+1, where d denotes the degree. The advantage of Bézier decomposition is that the algorithm can express a wide range of shape types into rational Bézier surface patches, i.e., for planes, cones, cylinders, spheres, as well as closed and open BSplines.

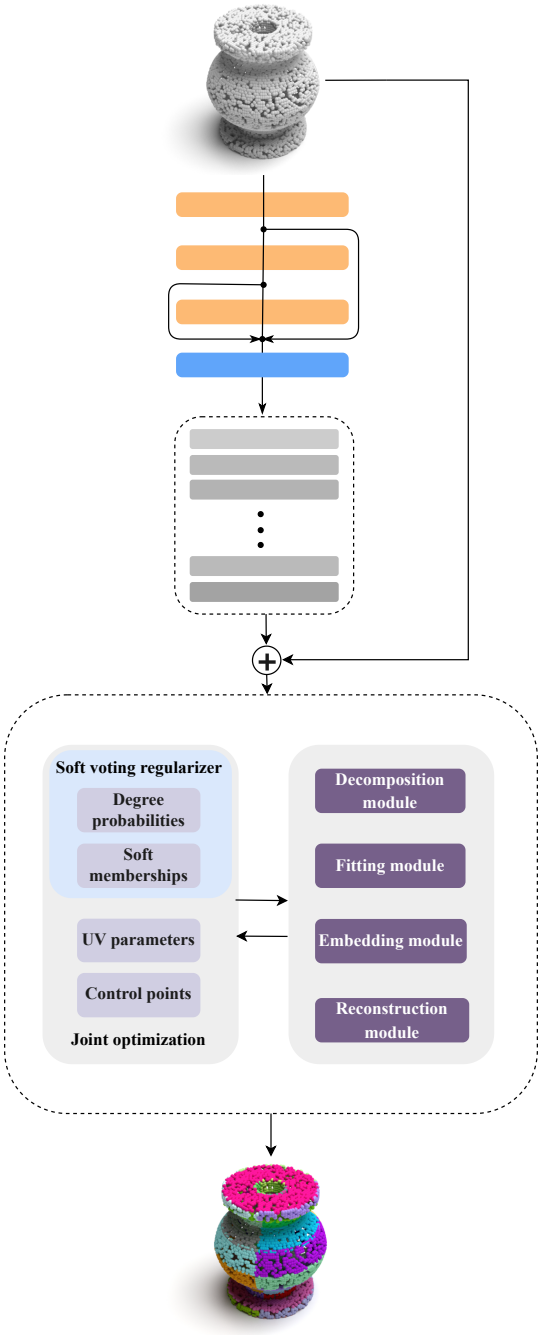


Figure 3.1: Overview of the proposed pipeline. The network takes a point cloud as input. It outputs pointwise features followed by four modules to predict Bézier geometry and topology: (1) The decomposition module decomposes point clouds into multiple patches. (2) The fitting module regresses *uv* parameters for each point and control points for each Bézier patch. (3) The embedding module clusters pointwise features assigned to the same patch. (4) The reconstruction module re-evaluates the input point clouds from the above predictions.

3.2.2 Overview

Figure 3.1 shows an overview of the proposed neural network. The input to our method is a 3D point cloud $P = \{p_i | 0 \leq i \leq N-1\}$, where p_i denotes the point coordinates (with or without normals). The output is the per-point patch labels $\{P_k | \cup_{k=0} P_k = P\}$, where each patch corresponds to a Bézier primitive. The network will also output patch degree (d_u-by-d_v) and weighted control points $C = \{\mathbf{c}_{kmn} = (x,y,z,w) | 0 \leq m \leq d_u, 0 \leq n \leq d_v, 0 \leq k \leq K-1\}$, where K denotes the number of patches. We constrain the maximum degree to be $M_d * N_d$. We let our network output a maximum number of K Bézier patches for all CAD models, and we use $\hat{K}$ to denote the ground truth number of patches which is smaller than K and varies for each CAD model.

3.2.3 Architecture

Our architecture consists of two components: a backbone for extracting features and a cascaded structure for joint optimization. The backbone is based on three stacked *EdgeConv* [WSL+19] layers and extracts a 256D pointwise feature for each input point. Let $\mathbf{P} \in \mathbb{R}^{N \times D_{in}}$ denote the input matrix, where each row is the point coordinates (D_{in} is three) with optional normals (D_{in} is six). Let $\mathbf{X} \in \mathbb{R}^{N \times 256}$ denote the 256D pointwise feature matrix extracted from the backbone. We use a cascaded structure to optimize the per-point degree probability matrix $\mathbf{D} \in \mathbb{R}^{N \times (M_d * N_d)}$, the soft membership matrix $\mathbf{W} \in \mathbb{R}^{N \times K}$, the UV parameter matrix $\mathbf{T} \in \mathbb{R}^{N \times 2}$, and the weighted control points tensor $\mathbf{C} \in \mathbb{R}^{K \times (M_d+1) \times (N_d+1) \times 4}$ jointly. Because $\mathbf{D}$, $\mathbf{W}$, $\mathbf{T}$, and $\mathbf{C}$ are coupled, it is natural to use a cascaded structure to jointly optimize them. Here, the cascaded structure is similar to [DHS16], where the features are concatenated and transformed for different MLP branches.

3.2.4 Joint Optimization

We have four modules: decomposition, fitting, embedding, and reconstruction. They are coupled to optimize $\mathbf{D}$, $\mathbf{W}$, $\mathbf{T}$ and $\mathbf{C}$ jointly by using our proposed four modules.

3.2.4.1 Decomposition

Degree classification. We use Bézier primitive with different degrees to replace classical primitives, including plane, sphere, plane, BSpline, etc. For the sake of the classification of degrees, the straightforward idea would be to use a cross-entropy loss: $\text{CE} = -\log(p_t)$, where p_t denotes the possibility of the true degree labels.

However, the degree type is highly imbalanced. For example, surfaces of degree type 1-by-1 represent more than 50%, while 3-by-2 surfaces are rare. To deal with the imbalance, we utilize the multi-class focal-loss [LGG+17]: $\mathrm{FL} = -(1-p_t)^{\gamma}\log(p_t)$, where γ denotes the focusing parameter. Then the degree type classification loss is defined as:

$$L_{\text{deg}} = \frac{1}{N}\sum_{i=0}^{N-1} \mathrm{FL}(\mathbf{D}_{i,:}) \tag{3.2}$$

Primitive segmentation. The output of primitive segmentation is a soft membership indicating per-point primitive instance probabilities. Each element w_{ik} is the probability for a point p_i to be a member of primitive k. Since we can acquire pointwise patch labels from our data pre-processing, we use a relaxed IOU loss [KK13, YHL+18, LSD+19] to regress the $\mathbf{W}$:

$$L_{\text{seg}} = \frac{1}{\hat{K}}\sum_{k=0}^{\hat{K}-1}\left[1 - \frac{\mathbf{W}_{:,k}^T\hat{\mathbf{W}}_{:,\hat{k}}}{\|\mathbf{W}_{:,k}\|_1 + \|\hat{\mathbf{W}}_{:,\hat{k}}\|_1 - \mathbf{W}_{:,k}^T\hat{\mathbf{W}}_{:,\hat{k}}}\right], \tag{3.3}$$

where $\mathbf{W}$ denotes the output of the neural network and $\hat{\mathbf{W}}$ is the one-hot encoding of the ground truth primitive instance labels. The best matching pairs $(k,\hat{k})$ between prediction and ground truth are found via the Hungarian matching [Kuh55]. Please refer to [LSD+19] for more details.

Soft voting regularizer. Since we learn $\mathbf{D}$ and $\mathbf{W}$ separately, points belonging to the same primitive instance may have different degrees, which is undesirable. To favor degree consistency between points assigned to the same primitive, we propose a soft voting regularizer that penalizes pointwise degree possibilities. We first compute a score for each degree case for all primitive instances by $\mathbf{S} = \mathbf{W}^T\mathbf{D}$, where each element s_{kd} denotes the soft number of points for degree d in primitive instance k. We then perform L_1-normalization to convert $\mathbf{S}$ into primitive degree distributions $\hat{\mathbf{S}}$:

$$\hat{\mathbf{S}} = \left[\frac{1}{\sum_{d=0} S_{kd}}\right] \odot \mathbf{S}, \tag{3.4}$$

where the first term denotes the sum of each column and $\odot$ denotes the element-wise product. Finally, we utilize a focal loss to compute the primitive degree voting loss:

$$L_{\text{voting}} = \frac{1}{\hat{K}}\sum_{k=0}^{\hat{K}-1} \mathrm{FL}(\hat{\mathbf{S}}_{k,:}), \tag{3.5}$$

where FL denotes the focal loss. The global loss for the decomposition module is defined as: $L_{\text{dec}} = L_{\text{deg}} + L_{\text{seg}} + L_{\text{voting}}$.

3.2.4.2 Fitting

Parameter regression. Through Bézier decomposition we obtain the ground truth labels for the (u, v) parameters and record all parameters into matrix $\hat{\mathbf{T}}$. We regress the uv parameters using a mean squared error (MSE) loss:

$$L_{\text{para}} = \frac{1}{N} \sum_{i=0}^{N-1} \|\mathbf{T}_{i,:} - \hat{\mathbf{T}}_{i,:}\|_2^2 \tag{3.6}$$

Control point regression. We select a maximum number of primitive instances K for all models. As the ground truth primitive instance $\hat{K}$ varies for each model, we reuse the matching pairs directly from the Hungarian matching already computed in the primitive segmentation step. Note that as the predicted degree (d_u, d_v) may differ from the ground truth $(\hat{d}_u, \hat{d}_v)$, we align the degree to compute the loss via a maximum operation as $(max(d_u, \hat{d}_u), max(d_v, \hat{d}_v))$. The network always outputs $(M_d+1) \times (N_d+1)$ control points for each primitive corresponding to the predefined maximum degree in U and V direction, and these control points will be truncated by the aligned degree. Furthermore, if the ground truth degree is smaller than the prediction, we can pad "fake" control points that are zero for the ground truth patch; otherwise, we just use the aligned degree, which is the maximum of the predicted and the ground truth. Finally, the control point loss is defined as:

$$L_{\text{ctrl}} = \frac{1}{N_{\mathbf{c}}} \sum_{t=0}^{N_{\mathbf{c}}-1} \|\mathbf{c}_t - \hat{\mathbf{c}}_t\|_2^2, \tag{3.7}$$

where $\mathbf{c}_t$ and $\hat{\mathbf{c}}_t$ denote the matched control points, and $N_{\mathbf{c}}$ is the number of matched control point pairs. Finally, we define the L_{fit} loss as: $L_{\text{fit}} = L_{\text{para}} + L_{\text{ctrl}}$.

3.2.4.3 Embedding

We use the embedding module to eliminate over-segmentation by pulling point-wise features toward their center and pushing apart different centers. Unlike ParSeNet and HPNet, 1) we do not need a mean-shift clustering step which is time-consuming; 2) we calculate the feature center in a weighted manner rather than simply averaging. The weights are chosen as $\mathbf{W}$ and will be automatically updated in the decomposition module; 3) $\mathbf{W}$ will be further optimized to improve the segmentation. Moreover, our embedding module is suitable for batch operations even though the number of primitive instances for each CAD model and the number of points for each primitive varies. Otherwise, one has to apply mean-shift for each primitive, which deteriorates timing further.

To be specific, we use $\mathbf{W}$ to weight $\mathbf{X}$ to obtain primitive features for all candidate primitive instances. Then, we reuse $\mathbf{W}$ to weigh all the primitive instance features to calculate a "soft" center feature for each point. We favor that each point feature embedding should be close to its "soft" center feature, and each primitive instance feature embedding should be far from each other. The primitive instance-wise feature matrix $\mathbf{X}_{\text{ins}}$ is defined as:

$$\mathbf{X}_{\text{ins}} = [\frac{1}{\sum_{i=0}^{N-1} w_{ik}}] \odot (\mathbf{W}^T \mathbf{X}), \tag{3.8}$$

where each row of $\mathbf{X}_{\text{ins}}$ denotes the instance-wise features for each patch. We then compute the "soft" center feature matrix $\mathbf{X}_{\text{center}}$ as: $\mathbf{X}_{\text{center}} = \mathbf{W}\mathbf{X}_{\text{ins}}$, where each row denotes the "soft" center for each point.

Then we define L_{pull} as:

$$L_{\text{pull}} = \frac{1}{N} \sum_{i=0}^{N-1} \text{Relu}(\|\mathbf{X}_{i,:} - (\mathbf{X}_{\text{center}})_{i,:}\|_2^2 - \delta_{\text{pull}}), \tag{3.9}$$

and we define L_{push} as:

$$\begin{aligned} L_{\text{push}} = \frac{1}{2K(K-1)} \sum_{k_1<k_2} \text{Relu}(\delta_{\text{push}} - \\ \|(\mathbf{X}_{\text{ins}})_{k_1,:} - (\mathbf{X}_{\text{ins}})_{k_2,:}\|_2^2). \end{aligned} \tag{3.10}$$

Finally, the total embedding loss L_{emb} is defined as: $L_{\text{emb}} = L_{\text{pull}} + L_{\text{push}}$.

3.2.4.4 Reconstruction

The reconstruction module is designed to reconstruct points from the predicted multiple Bézier primitives, i.e., rational Bézier patches, and further jointly optimize $\mathbf{W}$. One difficulty is that each CAD model has various numbers of primitives, and the degree of each primitive is also different. Therefore, we seek a generalized formula to support tensor operations on re-evaluating points for a batch of CAD models. The straightforward approach would be to compute a synthesizing score for all degree types. Assume the maximum number of primitive instances is K, and we have $M_d * N_d$ types of different degrees. The total number of combinations is $K * M_d * N_d$. We define a synthesizing score for each case in Einstein summation form: $(s_w)_{kci} = w_{ik} * s_{kc}$, where w_{ik} denotes the probability of point p_i to belong to primitive instance k and s_{kc} denotes the degree score for degree type m-by-n indexed with $c = M * (m-1) + (n-1)$ for primitive instance k coming from $\mathbf{S}$. Then, we

need to normalize $(s_w)_{kdi}$ such that $\sum_{k,d,i}(s_w)_{kdi} = 1$. Finally, the reconstructed point coordinates p_i are defined as:

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \sum_{k,m,n} (s_w)_{kci} \mathbf{R}_{kmn}(u_i, v_i), \tag{3.11}$$

where parameter (u_i, v_i) for point p_i is shared for all combinations. Such a formulation makes extending the formula in matrix form easy and avoids resorting to loop operations. However, such an approach is too memory-intensive. We thus truncate the degree from the degree probability matrix by re-defining the Bernstein basis function for degree d as:

$$(B_M)_d^l(t) = \begin{cases} \binom{d}{l} t^l (1-t)^{d-l}, & l \leq d \\ 0, & l > d \end{cases}, \tag{3.12}$$

where $0 \leq l \leq M$, and M is the maximum degree. Then, the reconstructed point coordinates for p_i for a degree m-by-n patch k is:

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \frac{\sum_{m_i}^{M_d} \sum_{n_i}^{N_d} (B_{M_d})_m^{m_i}(u)(B_{N_d})_n^{n_i}(v) \mathbf{c}_{m_i n_i} (c_w)_{m_i n_i} w_{ik}}{\sum_{m_i, n_i} (B_{M_d})_m^{m_i}(u)(B_{N_d})_n^{n_i}(v)(c_w)_{m_i n_i} w_{ik}}, \tag{3.13}$$

where $\mathbf{c}_{m_i n_i}$ denotes the control point coordinates and $(c_w)_{m_i n_i}$ denotes its weight, and w_{ik} is the element of $\mathbf{W}$.

If we also input the normal $(n_{x_i}, n_{y_i}, n_{z_i})$ for point p_i, we can also reconstruct the normal $(n'_{x_i}, n'_{y_i}, n'_{z_i})$ by:

$$\begin{pmatrix} n'_{x_i} \\ n'_{y_i} \\ n'_{z_i} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_i'}{\partial u} \\ \frac{\partial y_i'}{\partial u} \\ \frac{\partial z_i'}{\partial u} \end{pmatrix} \times \begin{pmatrix} \frac{\partial x_i'}{\partial v} \\ \frac{\partial y_i'}{\partial v} \\ \frac{\partial z_i'}{\partial v} \end{pmatrix}, \tag{3.14}$$

where $\times$ denotes the cross product.

$\mathbf{p}_i$ denotes the input point coordinates. $\mathbf{p}_i^*$ denotes the reconstructed point coordinates. $\mathbf{n}_{p_i}$ denotes the input point normals. $\mathbf{n}_{p_i}^*$ denotes the reconstructed normals. The coordinate loss is defined as:

$$L_{\text{coord}} = \frac{1}{N} \sum_{i=0}^{N-1} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2. \tag{3.15}$$

If we also input the normals, the normal loss is defined as:

$$L_{\text{norm}} = \frac{1}{N} \sum_{i=0}^{N-1} (1 - |\mathbf{n}_{p_i}^T \mathbf{n}_{p_i}^*|). \tag{3.16}$$

The loss for the reconstruction module is defined as:

$$L_{\text{recon}} = \begin{cases} L_{\text{coord}}, & \text{without normals,} \\ L_{\text{coord}} + L_{\text{norm}}, & \text{with normals.} \end{cases} \tag{3.17}$$

3.2.4.5 Total Loss

The total loss is defined as the sum of decomposition, fitting, embedding, and reconstruction losses: $L = L_{\text{dec}} + L_{\text{fit}} + L_{\text{emb}} + L_{\text{recon}}$. We do not use different weights for each loss item because all point clouds are normalized into a unit sphere. Moreover, the *uv* parameters are outputted directly from a *sigmoid* layer, and the control points are outputted directly by a *tanh* layer. Thus, each loss item is almost at the same scale, so we do not need different weights for each loss item. Furthermore, we use different learning rates for different modules to balance the training. Specific training details are listed in section 3.3.2.

3.3 Experiments

3.3.1 Dataset Pre-Processing

We evaluate our approach on the ABC dataset [KMJ+19]. The ABC dataset provides three formats for each 3D CAD model, STEP, YAML, and OBJ, to describe the geometry and topology. Among those formats, STEP is the raw file format with all information from the CAD models, YAML is used for recording the feature descriptors, and OBJ is used to record the related meshes. We only use the STEP format, as the YAML and OBJ are absent for some CAD models. The ABC dataset does not

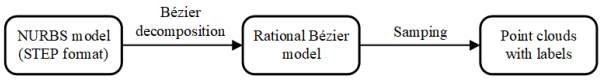
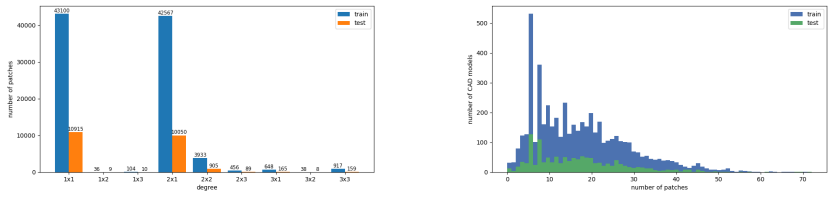


Figure 3.2: The pipeline of pre-processing ABC dataset.

have the annotations to learn Bézier decomposition on point clouds. Therefore, we utilize the CGAL library [CGA09] and OpenCascade library [Ope18] to perform Bézier decomposition on STEP files directly and perform random sampling on the surface to obtain the following labels: point coordinates, point normals, point *uv* parameters, surface patch indices of the corresponding points, surface patch degrees, and surface patch control points. NURBS surfaces with significant trimming defects are not sampled during the initial dense sampling step. Figure 3.2 shows the pipeline

of pre-processing ABC dataset. After the initial dense sampling, we find that the degrees for most patches are below 3-by-3, and the number of patches for most CAD models is lower than 75. As a result, we do not add into the dataset the CAD models with degrees higher than 3-by-3 and patches with more than 75. We then proceed to random sampling to generate $8,192$ points from the initial dense samples. Figure 3.3 plots the distributions of the dataset, i.e., the number of points against degree types and the number of CAD models against patch numbers. The resulting training dataset contains 6,500 models, 5,200 for training and 1,300 for testing.



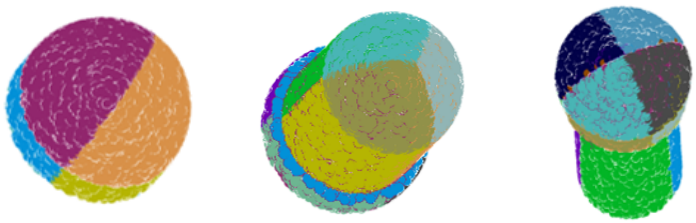
(a) (b)

Figure 3.3: Distributions of surface patches. (a) We plot the number of patches against patch degrees. (b) We plot the number of CAD models against the number of patches. We find that the degrees for most patches are below 3-by-3, and the number of patches for most CAD models is lower than 75.

One issue of Bézier decomposition is that the decomposition patterns are not unique for the ground truth data. Figure 3.4 shows the result of decomposing sphere-like objects. There are three types of patterns: bisection, trisection, and quad-section for spheres. The reason is that some CAD models designed by designers may contain duplicate control points, thus resulting in the different patterns of Bézier decomposition. The embedding module of our network could solve this issue deriving from the dataset. Our network tends to quarter the smooth axis-symmetric objects while keeping the boundaries of each patch smooth, no matter what the patterns are. These properties make our method more capable of adapting to real-world scenarios such as laser-scanned point clouds. The following sections will show the results of free-form point clouds and real scan data.

3.3.2 Training Details

We train a multi-task learning model. The learning rates differ depending on the MLP branch. The learning rate for the backbone, soft membership, and *uv* parameters is set to 10^{-3}, while the learning rate for the degree probabilities and control



(a) Bisection (b) Trisection (c) Quad-section

Figure 3.4: Patterns of Bézier decomposition of sphere-like objects in the dataset. Here we perform Bézier decomposition directly on the ABC STEP files and then sample points for each rational Bézier patch. Left: the sphere is bisected. Middle: the sphere is trisected. Right: the sphere is quadrisected.

points is set to 10^{-4}. As we have several learning tasks that are not independent, we set a lower learning rate for loss items, such as degree probabilities which converges faster. We set γ as 3.0 for the focal loss, and δ_{pull} as 0 and δ_{push} as 2.0 for the embedding losses. We employ ADAM to train our network. The model is then trained using 150 epochs.

3.3.3 Comparisons

We compare our algorithm with SPFN, ParSeNet, and HPNet [LSD+19, SLM+20, YY21]. We use both points and normals for training all the algorithms. Since SPFN only supports four types of canonical primitives (plane, sphere, cone, and cylinder), we consider points belonging to other primitives falling out of the supported canonical primitive types as the "unknown" type. To make fair comparisons, we modify SPFN to let the network take point coordinates and normals as input for training. For ParSeNet, we only train the segmentation module on the ABC dataset. We use their pre-trained fitting model (SplineNet) directly. For HPNet, we also use the pre-trained fitting model directly, which is the same as ParSeNet. We observed that the output of HPNet is very sensitive to the number of points. In order to use HPNet at its best, we down-sample the point clouds to 7k points for training and testing. We choose the following evaluation metrics:

1. **Primitive Type Accuracy** ("Acc"):
   $\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{I}(t_k == \hat{t}_k)$, where t_k and $\hat{t}_k$ are predicted primitive type and ground truth type, respectively. This is used to measure the type accuracy. Note that our primitive types differ from other baselines.

2. **Rand Index** ("RI"):
   $\frac{a+b}{c}$, where c is $\binom{N}{2}$ denoting the total possible pairs for all points, and a denotes the number of pairs of points that are both in the same primitive of prediction and ground truth, while b denotes the number of pairs of points that are in a different primitive of prediction and ground truth. Rand index is a similarity measurement between two instances of data clustering, and a higher value means better performance [CGF09, YHL+18].

3. **Normal Error** ("Err"):
   $\frac{1}{N}\sum_{i=0}^{N-1}\arccos(|\mathbf{n}_{p_i}^T\mathbf{n}_{p_i}^*|)$, where $\mathbf{n}_{p_i}$ and $\mathbf{n}_{p_i}^*$ are ground truth and predicted unit normal, respectively.

4. **Inference Time** ("Time"):
   The inference time on the whole test dataset.

5. **Average Primitive Number** ("Num"):
   The predicted average number of primitives on the whole test data set.

We record these evaluation metrics in Table 3.1. Figure 3.5, Figure 3.6 and Figure 3.7 show visual depictions of the results. Our results show the best performance regarding primitive type accuracy, normal fitting error, and inference time. Our

Method	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Time(min)(↓)	Num
HPNet	94.09	97.76	0.1429	1120.31	13.75
ParSeNet	94.98	97.18	-	252.01	14.06
SPFN	83.20	93.03	0.1452	11.52	21.02
Ours	**96.83**	95.68	**0.0522**	**4.25**	19.17

Table 3.1: Evaluation for primitive instance segmentation on the ABC-decomposition dataset. "rad" denotes the radian. Here we input both point coordinates and normals.

method is much faster for inference because it uses a general formula for different primitive types, and the embedding module is free of iterations. Other methods treat primitives with different equations, and ParSeNet and HPNet need a mean-shift step. What's more, HPNet tends to mix thin layers of point clouds for complex shapes, seeing the third row of Figure 3.5 and the first row of Figure 3.7. ParSeNet tends to under-segment point clouds, also seeing the first row of Figure 3.7: two separate cylinders are recognized as one cylinder (marking orange) even though it

looks clean. SPFN is the first method of deep learning to detect primitives; however, it only supports four basic primitives and treats all parametric surfaces as unknown types, thus leading to bad performance for patch type accuracy.

We also show the performance of all the methods without normals as input. For our method and SPFN, we only input point coordinates into the neural networks but use normals as supervision. Since ParSeNet does not regress normals, we cannot use normals as supervision. We train ParSeNet without normals as input to test its performance. HPNet uses the network to regress the normals from the input and also utilizes the ground truth normals to construct an affinity matrix as a post-processing step for clustering. We modify HPNet to let the affinity matrix be constructed from the regressed normals instead of the ground truth normals. Table 3.2 records the evaluation metrics of each method without using normals. From the experiments, we deduce that normals are important for the task of parametric primitive segmentation.

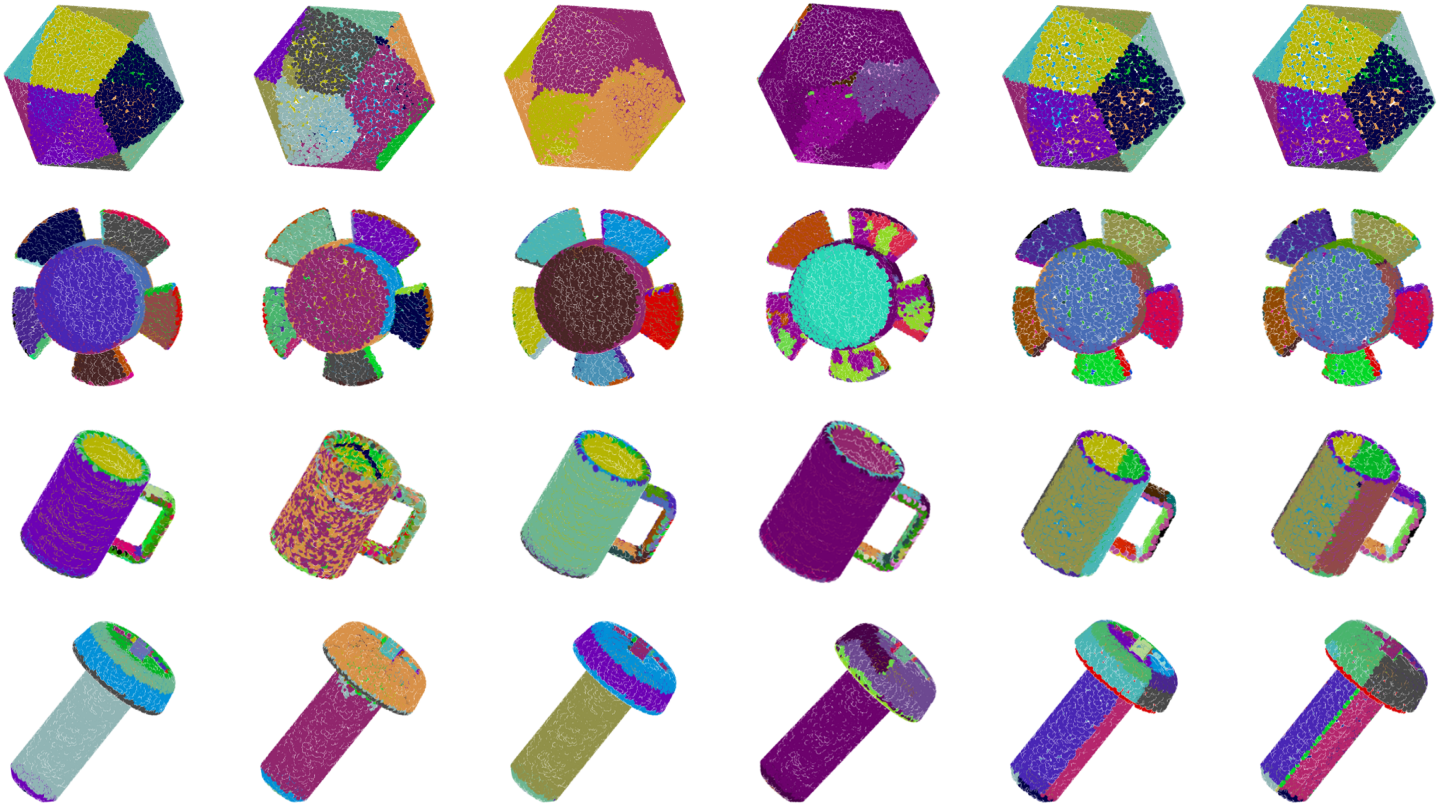
Method	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Time(min)(↓)	Num
HPNet	90.45	96.04	0.2256	1165.95	22.08
ParSeNet	90.31	94.55	-	206.74	15.39
SPFN	72.90	80.76	0.3309	10.46	27.61
Ours	**92.34**	90.54	**0.2003**	**4.05**	33.67

Table 3.2: Here we only input point coordinates into the neural network.

3.3.4 Ablation Studies

We first conduct experiments to verify the usefulness of the soft voting regularizer. The soft voting regularizer favors point primitive type consistency for each primitive instance, i.e., points assigned to the same primitive instance should have the same primitive type. From our experiment, we find that the soft voting regularizer not only improves the primitive type accuracy but also accelerates training relaxed IOU. Please refer to Figure 3.8 and the last two rows of Table 3.3.

We then verify the functionalities of each module. If we only use the decomposition module, the result is not good even though the "Acc" and "RI" are slightly higher because the decomposition module ignores the fitting, limiting the segmentation applicable to specific datasets. The reconstruction module reduces the "Err" significantly compared to the fitting module because the reconstruction module controls how "well-fitted" a predicted Bézier primitive is to the input point clouds. In contrast, the fitting module only regresses the control points and *uv* parameters.



(a) GT-Basic (b) HPNet (c) ParSeNet (d) SPFN (e) GT-Ours (f) Ours

Figure 3.5: Comparisons on the ABC dataset, where GT-Ours denotes the ground truth for Bézier decomposition, and GT-Basic denotes the ground truth for HPNet, ParSeNet, and SPFN.

(a) GT-Basic (b) HPNet (c) ParseNet (d) SPFN (e) GT-Ours (f) Ours

Figure 3.6: Comparisons on the ABC dataset, where GT-Ours denotes the ground truth for Bézier decomposition, and GT-Basic denotes the ground truth for HPNet, ParSeNet, and SPFN.

(a) GT-Basic (b) HPNet (c) ParseNet (d) SPFN (e) GT-Ours (f) Ours

Figure 3.7: Comparisons on the ABC dataset, where GT-Ours denotes the ground truth for Bézier decomposition, and GT-Basic denotes the ground truth for HPNet, ParSeNet, and SPFN.

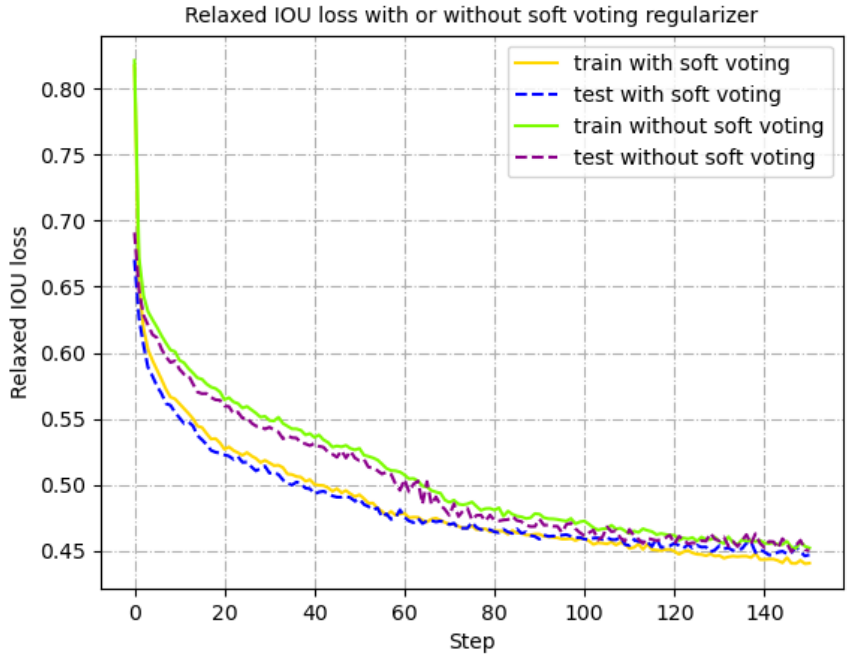


Figure 3.8: Relaxed IOU loss with or without soft voting loss.

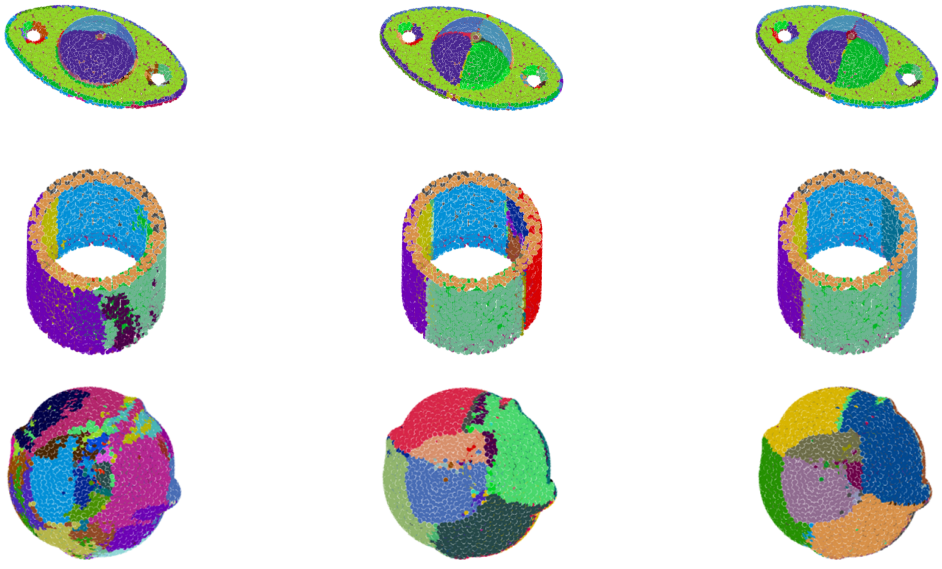
The embedding module is designed to eliminate small patches that contain few points, seeing the "Num" column. Therefore, experimenting with the embedding module results in fewer patch numbers than its counterpart. To conclude, training with all the modules yields the best results.

Moreover, we also provide a visual ablation study of different modules, seeing Figure 3.9. The decomposition module is the basic module. Training with the decomposition module will already provide a coarse segmentation. The embedding module will eliminate small patches, making the segmentation more compact. Furthermore, the fitting module directly regresses the control points and *uv* parameters. The reconstruction module also implicitly improves the regressing of the control points and *uv* parameters by reconstructing the input point clouds. The fitting and reconstruction modules work together to improve the final segmentation results.

Finally, we also conduct experiments to justify using focal loss instead of cross-entropy. Focal loss is chosen for two reasons: 1) cross-entropy yields faster training convergence while training relaxed IOU is slow, and 2) the degree type is imbalanced, seeing distributions of degree types. Please refer to Table 3.4 for the performance of the loss.

Module	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
D	97+0.10	96+0.83	1.0982	24.46
D+E	97+0.26	96−0.36	0.9834	22.54
D+F	97+0.19	96+0.52	0.5424	23.45
D+E+F	97−0.02	96−0.39	0.4884	20.85
D+F+R	97+0.19	96+0.48	0.0819	23.08
No-voting	97−1.32	96−0.44	0.0547	19.45
Full-module	97−0.17	96−0.32	0.0522	19.17

Table 3.3: Ablation study. D denotes the decomposition module. E denotes the embedding module. F denotes the fitting module. R denotes the reconstruction module. No-voting denotes using all modules without the soft voting regularizer. Full-module denotes using all the modules plus the soft voting regularizer.



(a) (b) (c)

Figure 3.9: The visual results of ablation study. (a): D. (b): D+E. (c): D+E+F+R. The above two rows are the testing data, while the last row is the real scan data.

Loss	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
CE	**96.95**	95.25	0.0684	28.99
FL	96.83	**95.68**	**0.0522**	19.17

Table 3.4: Ablation study. CE: using cross-entropy with all the modules. FL: using focal loss with all the modules.

3.3.5 Stress Tests

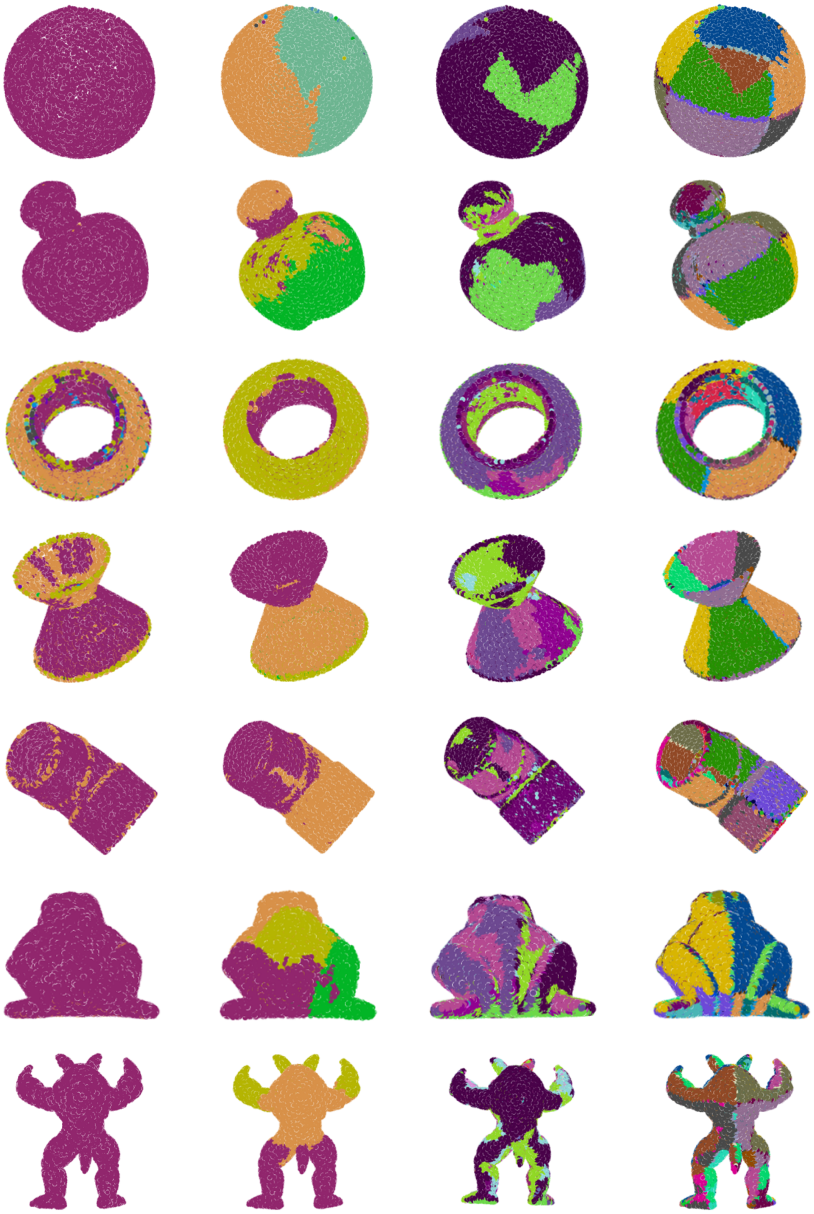
We first test the performance of our network by adding Gaussian white noise. We apply different scales of Gaussian white noise to the point coordinates after normalizing them into a unit sphere. The noise scale denotes the standard deviation of the Gaussian white noise. It ranges from 0.01 to 0.05. Importantly, our network was trained exclusively on clean, noise-free data. However, we designed the testing phase to be more challenging by subjecting the trained network to the influence of Gaussian white noise. Please refer to Table 3.5.

Noise scale	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
No-noise	**96.83**	**95.68**	**0.0522**	19.17
0.01	96.75	94.27	0.0525	20.38
0.02	96.63	93.48	0.0529	21.68
0.03	96.34	92.73	0.0538	22.76
0.04	96.15	92.04	0.0552	23.68
0.05	96.07	91.34	0.0559	24.38

Table 3.5: Evaluation of our algorithm at different noise scales.

To ensure the practical viability of our algorithm and to gauge its efficacy in real-world settings, we extended our evaluation to incorporate results from the Aim@Shape dataset [Fal04] which comes from the real scans. The Aim@Shape dataset offers a distinct challenge from the previously discussed ABC dataset, as it presents non-uniform sampling, missing data, and inherent measurement noise—conditions. It is important to note that, due to the absence of ground truth labels within the Aim@Shape dataset, direct training of our network was unfeasible. Consequently, we still use the models trained on the ABC dataset and test the performance directly on the Aim@Shape dataset. Our algorithm still works, while other methods are sensitive. Another positive aspect is that our algorithm

could decompose the axis-symmetric free-form point clouds with much smoother boundaries of different patches. Please refer to Figure 3.10.



(a) HPNet (b) ParSeNet (c) SPFN (d) Ours

Figure 3.10: Stress tests on real scan data. The above rows are the CAD point cloud, while the last two rows are the free-form point clouds.

3.3.6 Applications

We can apply our method to fit the full Bézier model by giving just point clouds. Although the network outputs rational Bézier patches, we do not utilize it directly because fitting rational Bézier patches is substantially more challenging than canonical Bézier patches. However, the degrees of the rational Bézier patch and canonical Bézier patch can be shared. As a result, we fetch the results of segmentation and predicted degrees of each patch from the output of the neural network as input for the refitting procedure. Algorithm 1 summarizes how we refit the untrimmed full Bézier model.

Algorithm 1: Refit Bézier model

Input: Segmented point cloud P with predicted degree d_u-by-d_v for each patch P_k, evaluation uv domain $[u_s, u_t] \times [v_s, v_t]$

Output: untrimmed Bézier model

foreach $P_k \in P$ **do**
- mesh P_k using advancing front [CSD04];
- parameterize the mesh of P_k using [LPRM02] to compute uv parameters for each input point;
- fit control points using least squares given the degree d_u-by-d_v and uv parameters;
- apply De Casteljau's algorithm [PT96] to evaluate the Bézier patch from a uniform uv domain grid $[u_s, u_t] \times [v_s, v_t]$;
- tessellate the evaluation points.

end

To evaluate the Bézier patch, we do not constrain the u and v parameters on the domain $[0,1] \times [0,1]$. Instead, we enlarge the domain to $[u_s, u_t] \times [v_s, v_t]$ where u_s and v_s are smaller than 0, and u_t and v_t are larger than 1. Thus, we can ensure the intersections of the adjacent untrimmed Bézier patch, which will be served as the trimming curves. Please note that we do not compute the closed form of trimming curves explicitly. Instead, we estimate the trimming curve by computing the intersections of the meshes coming from the tessellated Bézier patch mesh. Algorithm 2 summarizes how we trim the untrimmed Bézier model.

Differrent ParSeNet, we do not need to pre-train a model that outputs a fixed control point size. Instead, we refit a canonicial Bézier and then trim the Bézier model. Comparing to ParSeNet, our reconstructed full Bézier model can preserve

Algorithm 2: Trim Bézier model

Input: Segmented point cloud P, untrimmed Bézier model, area threshold a_{th}, distance threshold d_{th}

Output: trimmed Bézie model

```
find intersection pairs Pr of meshes from untrimmed Bézier model ;
foreach pr ∈ Pr do
    do co-refinement [CGA09] for pr to calculate and refine the
     intersection edges from meshes ;
end
foreach P_k ∈ P do
    break the co-refined mesh of P_k by removing the intersection edges;
    find connected components comp of the broken mesh;
    foreach comp do
        compute the area a of triangles which we could project sample
         points of P_k into;
        compute the average distance d from the incenter of triangles to its
         closest sample points of P_k;
        if a < a_th or d > d_th then
            remove comp;
        end
    end
end
remove small isolated fragments;
```

the sharp features, while the boundaries of ParSeNet for different primitives are jaggy and thus fail to preserve the sharp features. Please refer to Figure 3.11.

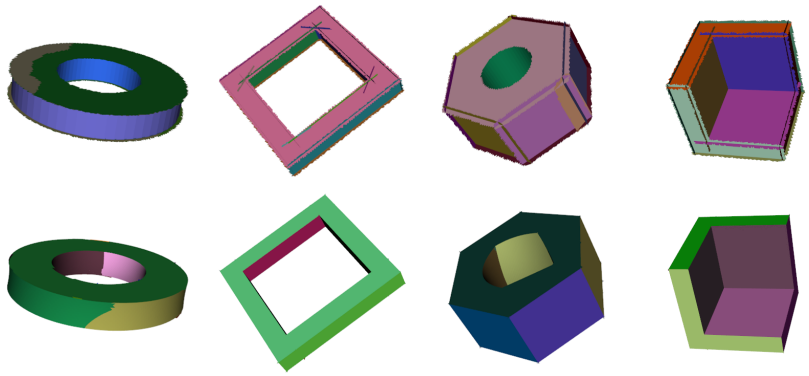


Figure 3.11: Reconstruction of the full Bézier model and visual comparison with ParSeNet. The above is the results of ParseNet and the bottom is our results.

3.3.7 Network Details and Verification

We present more detailed experiments on the architecture of the proposed BPNet. The BPNet has two parts: a backbone for extracting features and a cascaded structure for joint optimization.

The backbone is based on *EdgeConv* layers [WSL+19], which extract global and local features from the KNN graph. If we input point normals, the input dimension is six; otherwise, it is three when inputting only point coordinates. Table 3.6 records the details of our network. We first extract local pointwise 256D features from the *EdgeConv* layers. Then, we use a multilayer perceptron (MLP) to extract a global 1024D feature from the local pointwise 256D features. We then concatenate the 1024D global feature and the 256D local pointwise features and feed the tied features (1280D) into a 2-layer MLP to yield the final 256D pointwise features.

The cascaded structure is inspired from [DHS16], and shown in Figure 3.12. Specifically, the cascaded structure transforms the pointwise features for different learning tasks. We first concatenate $\mathbf{X}$ with $\mathbf{P}$ and feed them into the degree probabilities branch to learn the per-point degree probability matrix $\mathbf{D}$, where each row denotes the degree probability for each point. We then concatenate $\mathbf{X}$, $\mathbf{P}$ and $\mathbf{D}$ as input of the instance segmentation branch to regress the soft membership matrix $\mathbf{W}$, where each element w_{ik} denotes the probability for the point p_i to be a member of instance k. We then concatenate $\mathbf{X}$, $\mathbf{P}$, $\mathbf{D}$, $\mathbf{W}$ and use them as input to regress the

Index	Layer type	Input dimension	Output dimension
1	EdgeConv	$N \times D_{in}$	$N \times 64$
2	EdgeConv	$N \times 64$	$N \times 64$
3	EdgeConv	$N \times 64$	$N \times 128$
4	Cat	$N \times (64, 64, 128)$	$N \times 256$
5	Conv-Bn-Relu	$N \times 256$	$N \times 1024$
6	Pool	$N \times 1024$	1024
7	Cat	$(N \times 256, 1024)$	$N \times 1280$
8	Conv-Bn-Relu	$N \times 1280$	$N \times 512$
9	Conv-Bn-Relu	$N \times 512$	$N \times 256$
10	Cat	$N \times (256, D_{in})$	$N \times (256 + D_{in})$
11	Conv-Bn-Relu	$N \times (256 + D_{in})$	$N \times 256$
12	Conv	$N \times 256$	$N \times 9$
13	Softmax	$N \times 9$	$N \times 9$
14	Cat	$N \times (256, D_{in}, 9)$	$N \times (265 + D_{in})$
15	Conv-Bn-Relu	$N \times (265 + D_{in})$	$N \times 256$
16	Conv	$N \times 256$	$N \times 75$
17	Softmax	$N \times 75$	$N \times 75$
18	Mul	$N \times (75, 9)$	75×9
19	Cat	$N \times (256, D_{in}, 9, 75)$	$N \times (340 + D_{in})$
20	Conv-Bn-Relu	$N \times (340 + D_{in})$	$N \times 256$
21	Conv	$N \times 256$	$N \times 2$
22	Sigmoid	$N \times 2$	$N \times 2$
23	Cat	$N \times (256, D_{in}, 9, 2)$	$N \times (267 + D_{in})$
24	Mul	$N \times (75, 267 + D_{in})$	$75 \times (267 + D_{in})$
25	Conv-Bn-Relu	$75 \times (267 + D_{in})$	75×256
26	Conv-Bn-Relu	75×256	75×64
27	Reshape	75×64	$75 \times 16 \times 4$
28	Tanh	$75 \times 16 \times 3$	$75 \times 16 \times 3$
29	Softmax	75×16	75×16

Table 3.6: Details of our network. D_{in} is three for just inputting point coordinates and six when inputting both coordinates and normals.

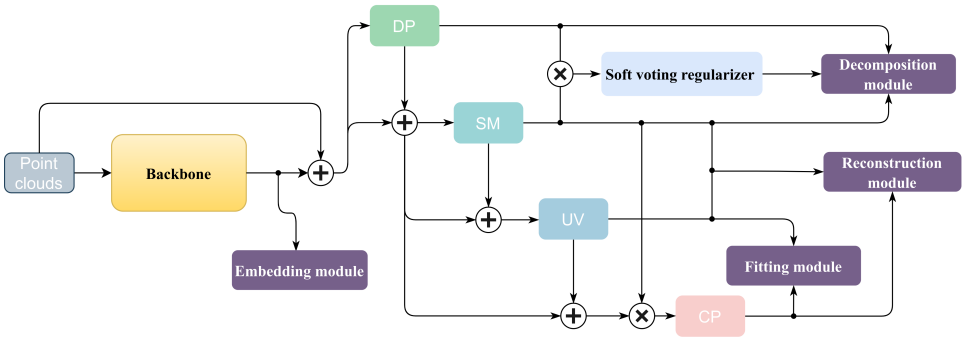


Figure 3.12: Details of the cascaded structure."DP" denotes the degree possibilities, "SP" denotes the soft memberships, "UV" denotes the UV parameters, and "CP" denotes the control points.

parameter matrix $\mathbf{T}$, where each row denotes the (u,v) parameters for a point p_i. Finally, we regress the control points matrix by first concatenating $\mathbf{X}$, $\mathbf{P}$, $\mathbf{D}$, $\mathbf{T}$ to generate the per-point feature matrix $\mathbf{X}_{ctrl0}$ where each row records the pointwise feature. We then use $\mathbf{W}$ as a weight to sum the $\mathbf{X}_{ctrl0}$ to yield the per-instance features $\mathbf{X}_{ctrl}$: $\mathbf{X}_{ctrl} = \mathbf{W}^T * \mathbf{X}_{ctrl0}$, where the size of $\mathbf{X}_{ctrl0}$ is $N * L$ denoting L-D per-point feature, and the size of $\mathbf{X}_{ctrl}$ is $K * L$ denoting the instance-wise feature. Finally, we feed $\mathbf{X}_{ctrl}$ into a 2-layer MLP to regress the control points matrix $\mathbf{C}$ for all instances. Note that the network outputs $(M_d + 1) \times (N_d + 1)$ control points for each patch. However, we truncate the control points matrix according to the predicted degree for fitting.

We then present a quantitative analysis of the complexity of our models using number of parameters and FLOPs in Table 3.7. The "Parameters" column indicates the number of learnable parameters within the model. This metric is indicative of the model's capacity to capture underlying patterns and nuances in the data. A higher number of parameters might suggest a model's ability to handle complex relationships, but it could also lead to overfitting if not carefully managed. The "FLOPs (GFLOPs)" column represents the rate at which floating-point operations are performed per second. FLOPs provide a measure of the computational workload required during both the training and inference phases.

We also verify the reasonable choice of cascaded structure for our joint optimization. An alternative structure might be a parallel structure for each module, seeing Figure 3.13. However, our learning objectives $\mathbf{D}$, $\mathbf{W}$, $\mathbf{T}$ and $\mathbf{C}$ are coupled. Optimizing one module will influence others. If we use the parallel structure, the

Layer index	Parameters		FLOPs	
	p	p+n	p	p+n
1-9	1.08M	1.08M	34.65G	35.1G
10-13	69.38K	70.15K	0.57G	0.58G
14-18	88.64K	89.41K	0.73G	0.74G
19-22	89.08K	89.85K	0.73G	0.74G
23-29	86.34K	87.1K	0.006G	0.1G

Table 3.7: Network complexity. "p" denotes using point coordinates as inputs, and "p+n" denotes using both point coordinates and normals.

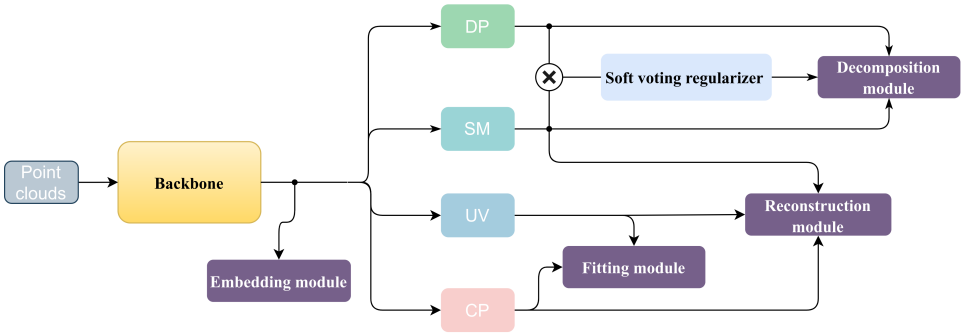


Figure 3.13: Details of the parallel structure.

gradient flow of each module has little impact on other modules. Thus, training on the parallel structure is more challenging. It needs a lower learning rate and more training epochs. From our experiment, we set the learning rate for the backbone, soft membership, and *uv* parameters to be $1e^{-4}$, while the learning rate for the degree probabilities and control points to be $1e^{-5}$. The training epoch is 300. We keep other configurations be same as the training of the cascaded structure. Please refer to the main paper for training details. Table 3.8 records the evaluations of both structures.

Structure	Acc(%)(↑)	RI(%)(↑)	Err(rad)(↓)	Num
Parallel	94.86	86.05	0.1081	15.47
Cascaded	**96.83**	**95.68**	**0.0522**	19.17

Table 3.8: Evaluation of the cascaded structure and parallel structure.

3.4 Conclusion

This chapter presents an end-to-end method to group points by learning Bézier decomposition. In contrast to approaches treating different geometric primitives separately, our method uses a general formulation for different primitive types. Regarding limitations, Bézier decomposition may naturally generate overly complex segmentations. In addition, we choose the rational Bézier patch as the primitive type. As the formulation is not linear, fitting the parametric patch is not direct. In future work, we wish to use the neural network to directly regress the canonical Bézier patch.

CHAPTER 4

LFS-Aware Surface Reconstruction

4.1 Introduction

Surface reconstruction refers to recovering a continuous surface (here a triangle surface mesh) from a 3D point cloud. The task was explored through different approaches, including explicit interpolation with Delaunay triangulations and implicit functions combined with iso-surfacing algorithms like marching cubes or marching tetrahedra [LC87, DK91]. However, reconstructing a surface mesh is often not sufficient, as additional properties such as mesh quality are often sought after for the said mesh, which are important for downstream applications such as simulation or visualization. Nevertheless, much attention has been paid to the fidelity of the reconstructed meshes, and less emphasis has been put on mesh quality or adaptive mesh sizing. When isotropic meshes are desired as output, remeshing algorithms have been shown relevant as a post-processing step for the raw reconstructed meshes. Isotropic remeshing algorithms aim to achieve one of the two main objectives: 1) ensuring that the triangle elements are as isotropic as possible, and 2) adapting the size of the triangle elements to the features of the surface, such as curvature or local feature size [AB98]. Large triangles shall be used for flat areas, while smaller triangles are required to approximate detailed areas. Generating such adaptive isotropic meshes is difficult as one must find the right balance between two contradictory objectives: complexity and reconstruction error.

In this work, we focus on generating a mesh with well-shaped (isotropic) elements and variable mesh sizing according to the local feature size (LFS). Such meshes exhibit a satisfactory balance between the quality of the elements, low complexity, and geometric fidelity. We tackle reconstruction and remeshing altogether by reconstructing an adaptive isotropic mesh directly from an unoriented 3D point cloud. Our method is motivated by the desire to operate LFS-aware meshing via Delaunay refinement [JAYB15] on an implicit function whose zero level-set

delineates the reconstructed surface. LFS captures important local or topological information: curvature, thickness, and separation. LFS is relevant to adapt the size of triangle elements for preserving fine details while obtaining lower reconstruction errors than proceeding to remeshing as a post-processing step.

4.2 Method

The input of our algorithm is a 3D point cloud $P = \{p \in \mathbb{R}^3\}$ with or without normals $N = \{n \in \mathbb{R}^3\}$. The output is an isotropic triangle surface mesh where the size of the triangle elements is controlled by the estimated LFS. Figure 4.1 depicts an overview of our algorithm. But we will first discuss our proposed Lipschitz-guided recursive dichotomic search before giving a step-by-step explanation, which is utilized several times in our reconstruction pipeline.

4.2.1 Lipschitz-Guided Recursive Dichotomic Search

We start with our proposed Lipschitz-guided recursive dichotomic search algorithm to detect the sublevels of a distance function. More specifically, we utilize the said dichotomic search to find the antipodal points when estimating the local shape diameter (Section 4.2.2.2), which is fundamental for estimating LFS. We also use it to determine a sign guess for an edge inside the envelope enclosed in the multi-domain discretization for solving the implicit function (Section 4.2.3.2). Unlike an exhaustive search that requires dense point sampling on a ray to ensure that all intersections are detected, such a dichotomic search accelerates computations by avoiding unnecessary point sampling.

Let $\Omega \subset \mathbb{R}^3$ denote a 3D domain and $\partial\Omega$ denote the boundary of Ω. The distance function $f(x) \in \mathbb{R}$ is defined as $f(x) = \inf_{y \in \partial\Omega} |x - y|$, where $x \in \mathbb{R}^3$ denotes a query point. From the definition, $f(x)$ is 1-Lipschitz continuous. Since we search along a ray, the query point x can be parameterized by a parameter $t \in \mathbb{R}$. For simplicity, we use this parameter t to refer to a query point x on a given ray. For a parameterized query point $t_1, t_2 \in \mathbb{R}$ on a given ray, we have $|f(t_1) - f(t_2)| \leq |t_1 - t_2|$, i.e. the distance function satisfies the 1-Lipschitz continuity. Therefore, given a ray trimmed by a search interval $[a, b]$, we leverage the Lipschitz continuity to shrink the search interval. More specifically, we obtain the following inequalities:

$$\begin{cases} |f(t) - f(a)| \leq |t - a| \\ |f(t) - f(b)| \leq |t - b| \end{cases} \tag{4.1}$$

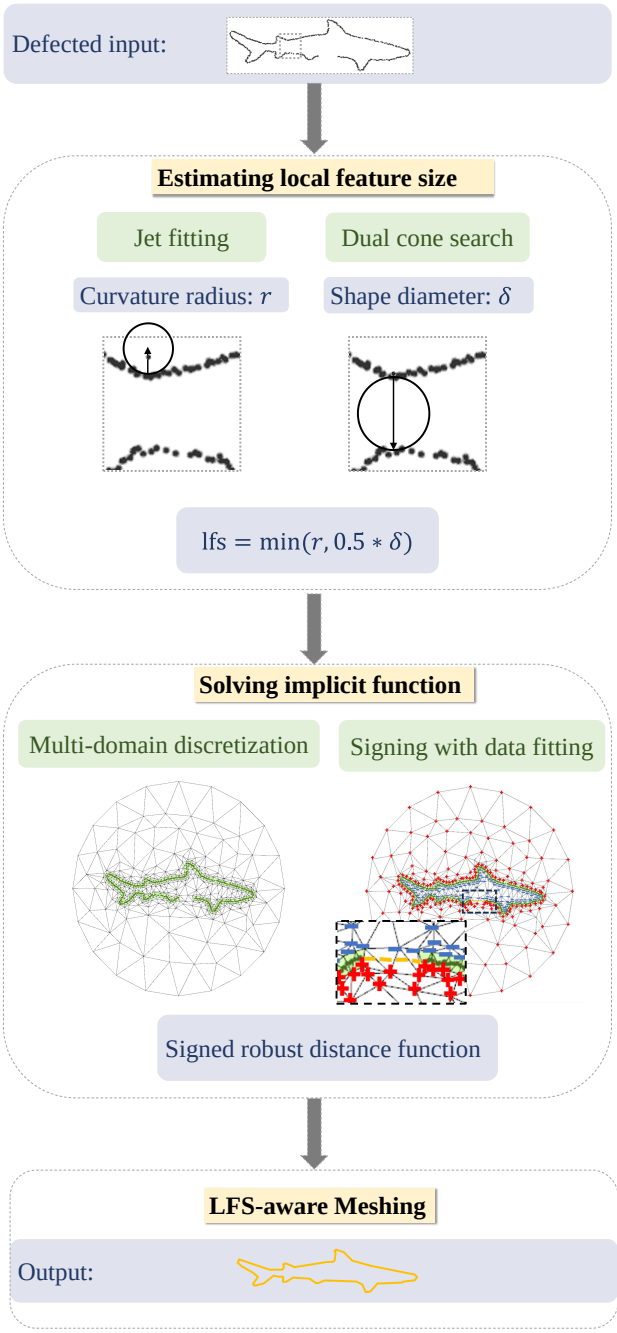


Figure 4.1: Overview. The input is a defective point cloud, with or without normals. The algorithm first estimates LFS as the minimum of the local curvature radius and half of the shape diameter. An implicit function is solved on a multi-domain discretization obtained by Delaunay refinement so as to fill large holes. Yellow dashed lines delineate the filled holes, "+" denotes the positive vertices, and "-" denotes the negative vertices. Finally, the output LFS-aware mesh is generated by Delaunay refinement.

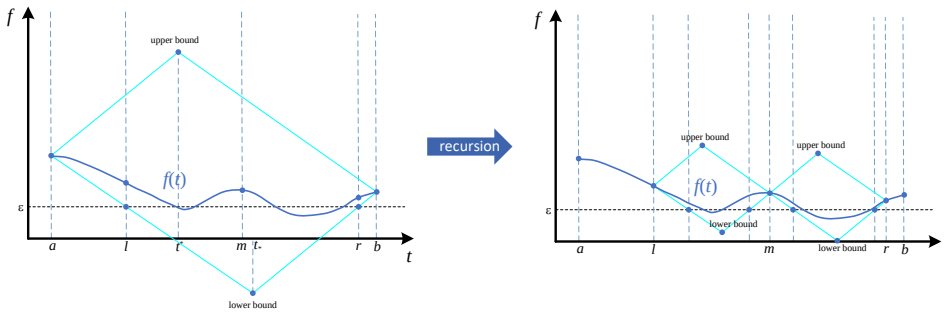


Figure 4.2: Lipschitz-guided recursive dichotomic search. The algorithm first searches in $[a,b]$, then the following recursion searches in $[l,m]$ and $[m,r]$ after bisection.

Equation (4.1) reflects that function $f(t)$ is bounded by a parallelogram region. By choosing a small value ε, we find two points l and r satisfying $|l-a| = \varepsilon$ and $|r-b| = \varepsilon$, as depicted by Fig. 4.2. This means that the points on the ray satisfying of $f(x) = \varepsilon$ lie in the interval $[l,r]$. Algorithm 3 details the steps of the searching algorithm that recursively bisects the search interval.

In Algorithm 3, m will be the approximated point if $f(m)$ approaches ε and the length $|r-l|$ is smaller than ε. Note that the container R_t stores the pair values approximating ε. When Algorithm 3 finds an approximated point m, if both $f(a)$ and $f(b)$ are greater or smaller than ε, we push m twice into the container R_t; otherwise, m is pushed once (see Fig. 4.3). The container stores the points such that $R_t = \{x_r : f(x_r) \approx \varepsilon\}$. Even though ε is a small value, the approximated points do not directly estimate the zero sublevel. To make a better approximation, we take the average values of m_2 and m_3 (Fig. 4.3) as the final points if $f(a)$ and $f(b)$ on the opposite side of ε. That explains why we push m_1 twice into the container and push m_2 and m_3 into the container only once. Finally, we regard the final point as the average of the adjacent two values stored in R_t.

4.2.2 LFS Estimation

Mathematically, LFS is defined as the distance from a query point on the surface to its closest point on the medial axis. Estimating the medial axis, e.g., from the Voronoi diagram [DS06] is difficult as it is sensitive to noise and sampling density. Instead, we estimate LFS directly from the input point set and bypass the construction of the medial axis. LFS captures a shape's local curvature, thickness, and separation - see Figure 4.4. Given a query point x on a shape embedded in 3D

Algorithm 3: Recursive_dichotomic_search(a, b)

Input: the search interval $[a,b]$, a 1-Lipschitz continuous function $f(t)$, a small value ε, a global empty container R_t

Output: approximated points x_r will be stored in R_t such that $R_t = \{x_r : |f(x_r)| \approx \varepsilon\}$

/* calculate the lower bound */

$t_* \leftarrow \frac{1}{2}[a+b-f(b)+f(a)]$, $y_* \leftarrow f(a)-(t_*-a)$

/* calculate the upper bound */

$t^* \leftarrow \frac{1}{2}[a+b+f(b)-f(a)]$, $y^* \leftarrow f(a)+(t^*-a)$

if $y_* \geq \varepsilon$ ***or*** $y^* \leq \varepsilon$ **then**

  return

end

if $f(a) > \varepsilon$ **then** $k_a \leftarrow -1$ **else** $k_a \leftarrow 1$;

if $f(b) > \varepsilon$ **then** $k_b \leftarrow 1$ **else** $k_b \leftarrow -1$;

$l \leftarrow a + \frac{\varepsilon - f(a)}{k_a}$, $r \leftarrow b + \frac{\varepsilon - y_b}{k_b}$, $m \leftarrow \frac{l+r}{2}$;

if $|f(m)-\varepsilon| \leq 10^{-7}$ **and** $|r-l| \leq \varepsilon$ **then**

  /* compare the binary signs */

  if $\text{sign}(f(a)-\varepsilon) == \text{sign}(f(b)-\varepsilon)$ **then**

    R_t.push(m) ;

    R_t.push(m) ;

  else

    R_t.*push*(m);

  end

  return;

end

Recursive_dichotomic_search(l, m);

Recursive_dichotomic_search(m, r);

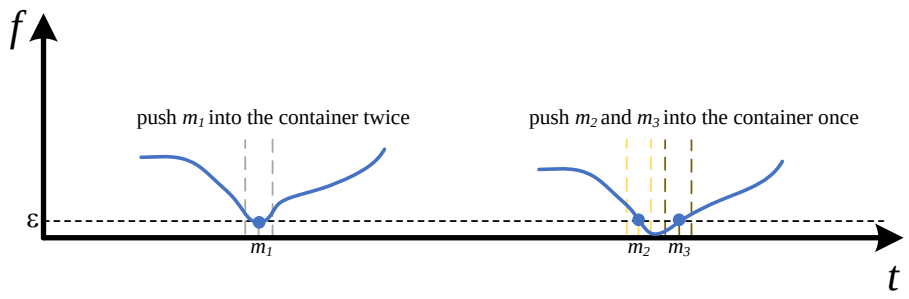


Figure 4.3: Points stored in the container R_t. Algorithm 3 pushes m_1 twice into container R_t, and pushes m_2 and m_3 once.

space, we define LFS as

$$\text{lfs}(x) = \min(r(x), 0.5 * \delta(x)), \tag{4.2}$$

where $r(x)$ denotes the estimated local curvature radius, and $\delta(x)$ denotes the estimated shape diameter function [SSCO08, RNDA13]. The curvature radius captures the local curvature, while the shape diameter function captures the local thickness and separation of a shape. In addition, our surface reconstruction approach relies on the reach - the minimum LFS - to construct a reach-aware multi-domain discretization.

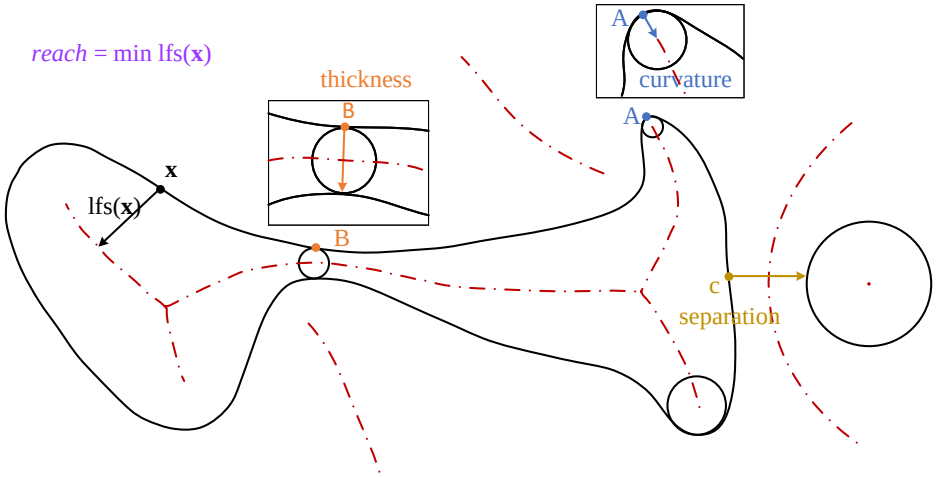


Figure 4.4: LFS captures the local curvature, thickness, and separation. The reach is the minimum of the LFS for the whole shape. The red dashed line depicts the medial axis.

4.2.2.1 Local Curvature Radius

The local curvature radius is estimated by fitting a differential jet locally[CP05]. Jet fitting constructs a local Monge coordinate system, whose basis (d_1, d_2, n) is defined by three orthogonal directions: maximum principal curvature, minimum principal curvature, and normal. The jet-fitting approach fits a Monge form as

$$z_p = \frac{1}{2}(k_1 x_p^2 + k_2 y_p^2) + hot, \tag{4.3}$$

where k_1 and k_2 denote the local principal curvatures, (x_p, y_p, z_p) denotes the coordinates of p, and *hot* denotes a higher-order term of polynomial. The curvature radius for a query point x is then defined as

$$r(x) = 1/|h(x)|, \tag{4.4}$$

where $h(x)$ is the curvature estimated via jet fitting with the maximum absolute value.

When the input 3D point cloud has no normal attributes, we estimate the normals via jet fitting, which are unoriented. Our reconstruction approach does not require normal orientation - see Section 4.2.3.

4.2.2.2 Shape Diameter

The shape diameter function maps a query point x on the surface of a 3D solid shape to a scalar defined as the distance from x to its antipodal surface point with respect to the local normal direction. Previous approaches [SSCO08, RNDA13] only consider the local thickness of the shape as they only search along the inward normal direction. Thus, the local separation is ignored. Besides, the original shape diameter function can only be estimated on a surface mesh where the antipodal points are well-defined given rays cast from a query point. Finding the antipodal points for an input 3D point cloud is hampered by the fact that the intersection point of a ray with a point cloud is ill-defined.

We address the two aforementioned issues by using a dual cone search algorithm that estimates an extended version of the shape diameter function $\delta(x)$, defined as

$$\delta(x) = \min(\tau(x), \sigma(x)), \tag{4.5}$$

where $\tau(x)$ denotes the thickness defined as the distance from x to its antipodal point along the inward normal direction, and $\sigma(x)$ denotes the separation defined as the distance from x to its antipodal along the outward normal direction. It's

worth noting that in our implementations, we conduct the search along the normal direction and its inverse direction, subsequently selecting the minimum of these two values. The final shape diameter value does not need normal orientation.

We proceed by analyzing the unsigned distance function to the input point cloud P along a ray r. The unsigned distance function is defined as

$$d_u(x) = \min_{p \in P}\{|x - p|\}, \tag{4.6}$$

where $x \in \mathbb{R}^3$ denotes a query point. As $d_u(x)$ is 1-Lipschitz continuous, we utilize a Lipschitz-guided recursive dichotomic search to improve efficiency - see Section 4.2.1. If $d_u(x)$ is smaller than a small distance value referred to as ε, we treat x as the antipodal point. We estimate ε from the unsigned robust distance function $\hat{d}_u(x)$ mapping a query point $x \in \mathbb{R}^3$ to a scalar value. $\hat{d}_u(x)$ is defined as

$$\hat{d}_u(x) = \sqrt{\frac{1}{k} \sum_{p \in N_k(x)} \|x - p\|^2}, \tag{4.7}$$

where $N_k(x)$ denotes the k nearest neighbors to a query point $x \in \mathbb{R}^3$. k is used to trade accuracy for robustness to noise and outliers [MDGD+10]. Then ε is estimated as the minimum $\hat{d}_u(x)$ computed at all input points, i.e., $\varepsilon = \min\{\hat{d}_u(p) | p \in P\}$. In addition, to capture both thickness and separation, we cast random searching rays inside two opposite cones with x as the apex and n and $-n$ as axes because it is challenging for a single ray to find an antipodal point directly. More specifically, we cast N random rays within each cone for a query point x to collect k antipodal points and calculate the robust distance function from x to the collected k antipodal points. Such a dual cone search may not find any antipodal points when the local point sampling is too sparse. In this case, we set as the local shape diameter the diameter of the loose bounding sphere of the input point cloud. Figure 4.5 illustrates the dual cone search for estimating the shape diameter.

4.2.2.3 Smoothing

In theory, LFS is a 1-Lipschitz continuous function. However, our LFS estimation procedure may lead to a noisy function when applied to point clouds with variable sampling density and noise. For example, two nearby query points may have a very large curvature estimated by jet fitting due to noise next to a large shape diameter equating to the radius of the loose bounding sphere due to sparse sampling. This results in salt-and-pepper noise in the estimated LFS function. We thus apply a two-step filtering process to denoise and smooth out the estimated LFS function.

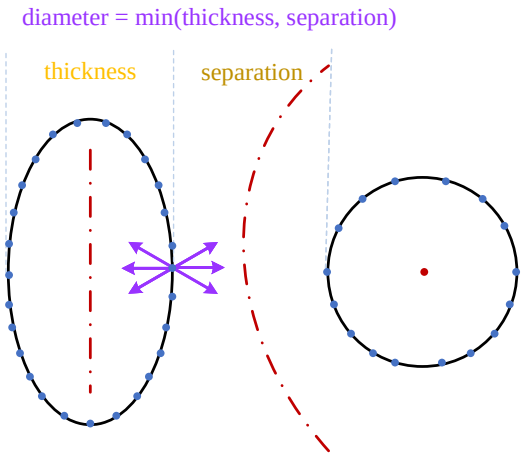


Figure 4.5: Dual cone search. The red dashed line depicts the medial axis. The shape diameter is the minimum of the thickness and separation.

We first apply a median filter to remove the salt-and-petter noise and then apply Laplacian smoothing to smooth out the denoised LFS function further.

4.2.3 Implicit Function

Our objective is to solve for a signed implicit function, piecewise-linear and defined on a 3D triangulation, whose zero level-set corresponds to the reconstructed surface. Instead of solving the implicit function using a single global solve step, our rationale is to compute its components sequentially. More specifically, we first construct a 3D multi-domain that discretizes an unsigned distance function to the input points. We then solve a least squares problem to obtain a signed implicit function from sign guesses estimated at the edges of the discretization. Finally, we utilize the signs of the solved implicit function to sign the unsigned robust distance function.

4.2.3.1 Multi-Domain

We construct a multi-domain discretization to represent the piecewise-linear implicit function. The multi-domain is the union of two 3D sub-domains: (1) A thin envelope enclosing the input 3D point cloud P, where the unsigned Euclidean distance to P is smaller than I_R - the reach of P, and (2) A loose bounding sphere of P minus the above envelope. The above envelope is made reach-aware to separate thin structures and preserve fine details. Adding and discretizing a loose bounding sphere is later required to fill large holes with a smoothly graded 3D triangulation. Delaunay

refinement is used to discretize the multi-domain into well-shaped tetrahedra, with different refinement criteria for the two sub-domains.

More specifically, the first domain is a reach-aware envelope derived from an unsigned implicit function $I_u(x)$. The domain is defined as

$$I_E(x) = \{x : I_u(x) \leq I_R\}, \tag{4.8}$$

where I_R denotes the reach of the inferred surface and $I_u(x)$ is defined as

$$I_u(x) = \frac{\sum_{p \in N_k(x)} |(x-p)^T \cdot n_p| W_p(x)}{\sum_{p \in N_k(x)} W_p(x)}, \tag{4.9}$$

where $W_p(x) = e^{-||x-p||^2/h^2}$ denotes a Gaussian weighted function. Different from previous work [Kol08], the term $I_u(x)$ does not require oriented normals as we use the absolute value, and we are not interested in the zero iso-level of $I_u(x)$. Instead, our objective is to discretize the domain delineated by the sublevel at the reach I_R, i.e., the minimum of LFS.

The second domain is defined as the complement between the above envelope and a loose bounding sphere defined as

$$I_S(x) = \{x : ||x-c|| \leq r\}, \tag{4.10}$$

where c denotes the centroid of the bounding sphere, and r denotes the radius of the loose bounding sphere: $\arg\max_{p,q} |p-q|$, where $p \in P$, $q \in P$ and $p \neq q$.

We utilize the Delaunay refinement paradigm [JAYB15] to generate the multi-domain discretization. In order to generate well-shaped cells, we specify a constant cell radius-edge ratio in all sub-domains. We also specify a constant facet shape criterion for the boundary of the loose bounding sphere. We refer to [JAYB15] for more details.

4.2.3.2 Signing with Data Fitting

Our multi-domain discretization includes a reach-aware envelope $I_E(x)$ embedded in a 3D Delaunay triangulation Tr bounded by a sphere $I_S(x)$. The sublevel of the unsigned function inside $I_E(x)$ contains most of the inferred surface, but it remains to compute a signed function in order to fill holes and delineate the inferred surface as its zero level-set. We proceed by solving for a signed function defined at the vertices of the multi-domain. Unlike previous approaches [MDGD+10, GCSA13] that rely on random ray casting to construct a random graph and intersection tests on the edges of the graph, our solver relies upon sign guesses defined on the edges

of the multi-domain and includes a data fitting term. Our method does not require random ray casting because the sign guesses outside the envelope are already known. This simplifies the signing process and reduces the computational cost.

The sign guesses are computed as follows. $\text{sign}(e_{mn})$ denotes the binary sign guess for an edge $e_{mn} \in Tr$. If e_{mn} crosses the inferred surface, $\text{sign}(e_{mn})$ is set to -1 to reflect that it connects two vertices with opposite signs and set to $+1$ otherwise. An edge e_{mn} has three possible locations: outside, inside, and on the boundary of the envelope. If e_{mn} is outside or on the boundary of the envelope, $\text{sign}(e_{mn})$ is directly set to $+1$ as e_{mn} does not cross the inferred surface. If e_{mn} is inside the envelope, we check whether e_{mn} connects two vertices with opposite signs. Note that the edges of slivers (flat cells) inside the envelope may not cross the inferred surface. We perform a Lipschitz-guided recursive dichotomic search (Section 4.2.1) on e_{mn} inside the envelope to check if the unsigned distance function $I_u(x)$ is below a value ε, which is set as half of the surface reach I_R. If $I_u(x)$ is below ε, we set e_{mn} to -1, and $+1$ otherwise. Equipped with sign guesses at edges, we then solve a signed implicit function $d_s(\cdot)$ for all vertices V of Tr by minimizing the following objective function

$$E = \sum_{e_{mn}} [d_s(v_m) - \text{sign}(e_{mn}) \cdot d_s(v_n)]^2 + \lambda \sum_t \sum_{p_t} [\sum_{i=0}^{3} \alpha_i(p_t) \cdot d_s(v_i)]^2, \tag{4.11}$$

where $d_s(\cdot)$ denotes the signed implicit function solved for V, $\alpha_i(p_t)$ is the barycentric coordinates for the input point p_t located in the tetrahedron t regarding the vertex v_i.

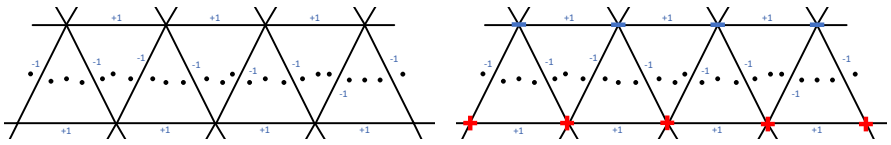
The first term of the above objective function optimizes the sign consistency on the multi-domain. The second term - data fitting - favors that the input points are located near the zero level-set of the interpolated signed implicit function. λ provides a means to balance between the two terms. For all shown experiments λ is set to 1.0.

To avoid the trivial solution, we enforce the sum of the signed implicit values to be a constant, i.e., $\sum_v d_s(v) = |V|$, where $|V|$ denotes the number of all vertices V. Figure 4.6 illustrates the signing process.

We can re-write the objective function in matrix form as

$$\begin{aligned} \min_x \quad & ||Sx||^2 + \lambda ||Bx||^2 \\ \text{s.t.} \quad & Cx = |V|, \end{aligned} \tag{4.12}$$

where S denotes the sign guess matrix. Each row corresponds to an edge of Tr and contains two elements, either 1 or -1. B is the barycentric coordinates of each input



(a) (b)

Figure 4.6: Solving signed implicit function from the sign guesses for edges. (a) The sign guess for an edge is determined by detecting the crossings with sublevel sets of the unsigned distance function to the points. An edge is labeled -1 if it crosses the point clouds, and $+1$ otherwise. (b) A signed implicit function is then solved from the sign guesses of the edges. Red crosses denote the positive vertices, and blue minus signs denote the negative vertices. The signed implicit function is piecewise linear.

point, and x is the values for the signed implicit function d_s that we need to solve for each vertex. C is a one-dimensional constraint matrix with the size of $1 \times |V|$ and is filled with element 1 everywhere. Both S and B are sparse. By applying the Karush–Kuhn–Tucker (KKT) conditions, we have

$$\min_{x,z} \quad (Sx)^T(Sx) + \lambda (Bx)^T(Bx) + z^T(Cx - |V|) \tag{4.13}$$

Finally, we solve the signed implicit function values x and the Lagrange multiplier z in the least squares sense

$$\begin{bmatrix} 2S^TS + 2\lambda B^TB & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ |V| \end{bmatrix} \tag{4.14}$$

We obtain a signed implicit function after solving (4.14). Since we solve the function from the sign guesses of edges that have two statuses, the distribution of function values at the vertices should have two peaks denoting positive and negative signs, respectively. Besides, the signed implicit function can also fill holes by propagating the signs. Our least-squares solver can propagate the sign on the triangulation vertices on hole regions correctly - see the holes in 2D, depicted by Figure 4.1.

4.2.3.3 Signed Robust Distance Function

The previous step solves for a signed implicit function defined on the 3D multi-domain. To obtain a robust implicit function that is more resilient to noise, we

use the binary sign value from the signed implicit function d_s to sign the unsigned robust distance function $\hat{d}_u$ to obtain a new implicit function $\hat{d}_s$ whose zero level-set is the target surface

$$\hat{d}_s = \text{sign}(d_s) \cdot \hat{d}_u, \tag{4.15}$$

where d_s denotes the solved signed implicit function values after solving (4.14).

4.2.4 LFS-Aware Meshing

A simple iso-surfacing method such as marching tetrahedra [DK91] would not yield the final reconstructed surface mesh with the desired properties (sizing, well-shaped tetrahedra). We utilize instead a Delaunay refinement algorithm [JAYB15] to extract the final size-varying isotropic mesh. The Delaunay refinement takes as input an implicit function with a mesh sizing function. It will output an isotropic triangular mesh contouring the zero level-set of the input implicit function with the mesh sizing function controlling the refinement termination. If sharp feature curves are provided, the final mesh can preserve sharp features. In our case, we feed both the signed robust distance function $\hat{d}_s(\cdot)$ together with mesh sizing function $\text{size}(\cdot)$, both of which are derived from the input point cloud directly, into the Delaunay refinement. Furthermore, we use NerVE [ZDC+23] to detect the sharp feature curves for shapes with sharp features.

The LFS-aware facet sizing function is defined as

$$\text{size}(x) = \frac{\text{lfs}(x) - I_R}{\text{lfs}_{\max} - I_R} \cdot (\text{size}_{\max} - \text{size}_{\min}) + \text{size}_{\min}, \tag{4.16}$$

where $\text{lfs}(x)$ denotes the local feature size, I_R denotes the surface reach, $\text{size}_{\max}$ denotes the user-specified maximal facet size, and $\text{size}_{\min}$ denotes the minimal facet size set as a user-specified ratio times the thickness of the envelope.

The facet sizing function $\text{size}(x)$ is also Lipschitz-continuous as it derives from a linear transformation of LFS. To avoid too large facet sizes and make the sizing function 1-Lipschitz, we use a breadth-first propagation to smooth the sizing function further. Starting from the point with a local minimum of the facet size values, we check if the size values from its neighbor satisfy the Lipschitz continuity. Refer to Algorithm 4 for details.

4.3 Experiments

The CGAL library [CGA09] was used to implement the multi-domain discretization and Delaunay refinement steps. We use the conjugate gradient solver from the Eigen

Algorithm 4: BFS_lipschitz_continuity_propagation

Input: knn graph of the input points, facet sizing function size(x)

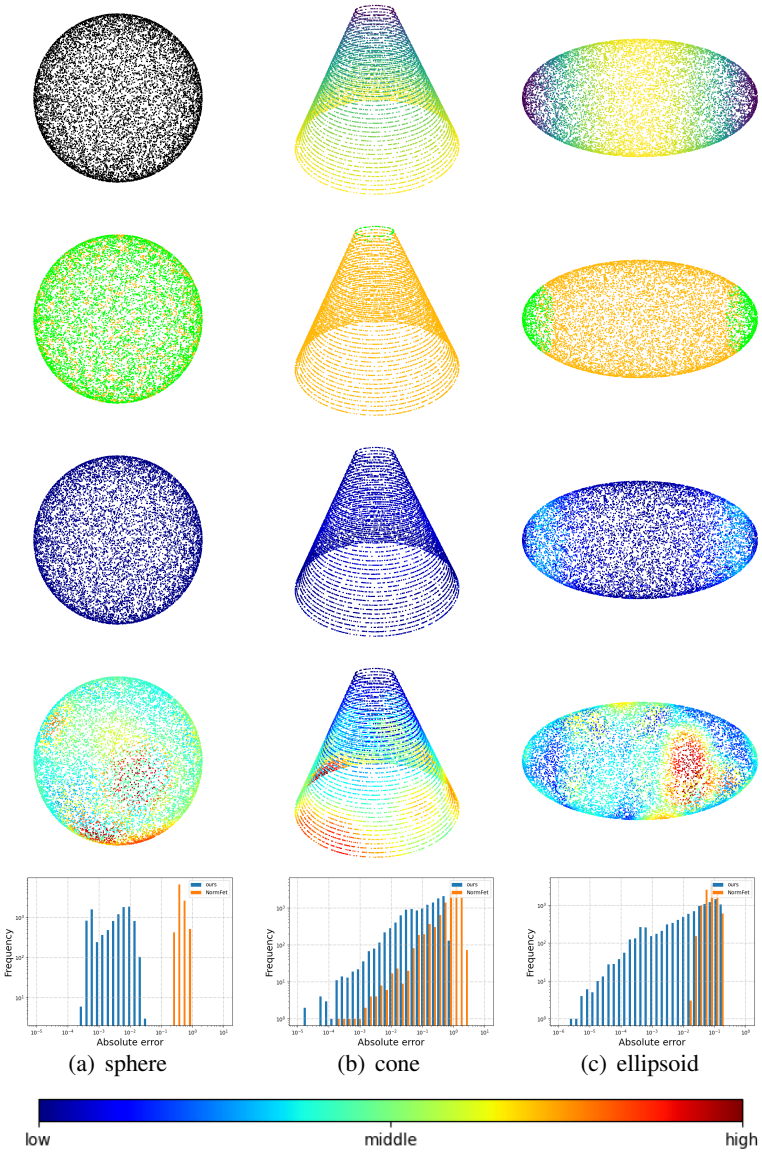
Output: smoothed facet sizing function size(x)

establish a priority queue Q whose top stores the minimum value ;
establish a map *visited* to mark if the point has been visited ;
for $p_i \in P$ **do**
    Q.push(size(p_i)) ;
end
while Q is not empty **do**
    size(p_{cur}) $\leftarrow$ Q.top() ;
    Q.pop() ;
    for $p_{\text{nei}} \in N_k(p_{\text{cur}})$ **do**
        if *visited*[p_{nei}] has been marked **then**
            continue ;
        end
        if size(p_{nei}) $>$ size(p_{cur}) **then**
            if size(p_{nei}) $-$ size(p_{cur}) $>$ $|p_{\text{nei}} - p_{\text{cur}}|$ **then**
                size(p_{nei}) $=$ size(p_{cur}) $+ |p_{\text{nei}} - p_{\text{cur}}|$
            end
        else
            Q.push(size(p_{cur}) ;
        end
    end
end

library [GJ+10] to solve the implicit function. Experiments are conducted on a Dell laptop with a 2.60GHz Intel i7-10750H CPU and 32GB memory.

4.3.1 LFS Estimation

Validation on canonical primitives. We first validate the proposed LFS estimation algorithm on 3D point clouds sampled on canonical primitives. We generate canonical primitives for which we know the ground-truth LFS analytically. We then measure the error made by our estimation algorithm (without smoothing) and compare it with NormFet [DS06], which computes the medial axis via constructing the Voronoi diagram. More specifically, we sample three primitives - sphere, cone,



(a) sphere (b) cone (c) ellipsoid

Figure 4.7: Visual analysis of LFS estimation on canonical primitives (sphere, cone, and ellipsoid) sampled non-uniformly, with little noise added. First row: ground truth LFS using the viridis color ramp. Second row: LFS type where green denotes curvature-based and orange denotes diameter-based. Third row: estimated LFS error depicted for our algorithm using the jet color ramp. Fourth row: estimated LFS error depicted for the NormFet algorithm. Fifth row: distribution of absolute estimation errors for all input points.

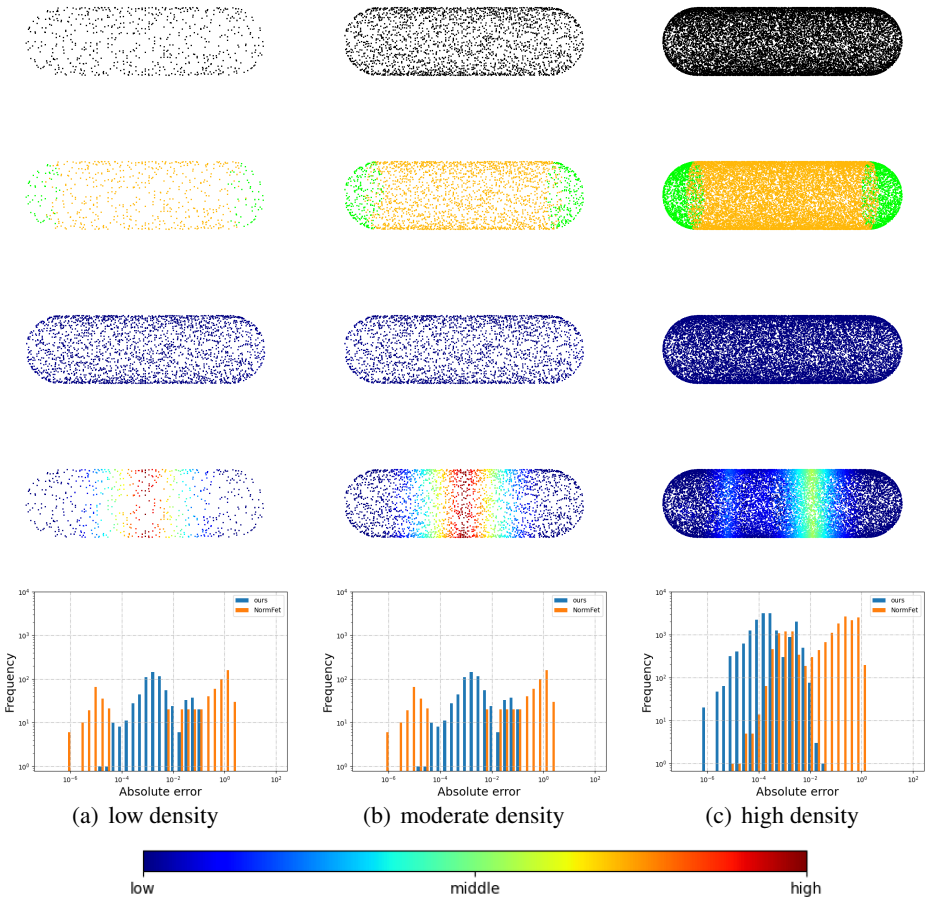
and ellipsoid - with non-uniform sampling and a small amount of added noise. We measure both the mean absolute error and max absolute error, as recorded in Table 4.1. Our experimental results show that our LFS estimation algorithm outperforms NormFet in terms of the mean and maximal absolute errors. The experiments highlight the robustness of our algorithm to non-uniform sampling and small amounts of noise. NormFet estimates LFS by first constructing the medial axis from the Voronoi diagram, but a good estimation of the medial axis requires a point cloud that is dense and noise-free. Remind that we define LFS as the minimum of curvature radius and half of the shape diameters. We can thus depict the origin of LFS, as depicted by Figure 4.7, second row. Green points depict the points with a curvature radius smaller than half of the shape diameter, and orange points depict the other points. As expected, the sphere - whose curvature radius equates to half of the shape diameter - yields a mixture of green and orange points. On the ellipsoid, the two tips correspond to a curvature-based LFS, while the rest are diameter-based. Figure 4.7 offers a comprehensive visual depiction.

Capsule	Sphere (#648)		Cone (#2610)		Ellipsoid (#16374)	
	mean(↓)	max(↓)	mean(↓)	max(↓)	mean(↓)	max(↓)
NormFet	4.079E-1	8.191E-1	9.413E-1	2.233	7.835E-2	1.917E-1
Ours	**5.511E-3**	**3.190E-2**	**2.1346E-1**	**7.751E-1**	**5.471E-2**	**1.786E-1**

Table 4.1: LFS comparisons with NormFet on canonical primitives. “mean” denotes the mean absolute error. “max” denotes the maximal absolute error.

Sampling density. We now evaluate the performance of our LFS estimation method (without smoothing) under different sampling densities and compare it with NormFet. To conduct this experiment, we construct a capsule model with ground-truth LFS, with a cylinder and two half-spheres whose radii are known as a priori. We then randomly sample points on the capsule with three different densities, see Figure 4.8 and Table 4.2. The experiments indicate that our algorithm provides a reasonable LFS estimation in all three sampling conditions. In contrast, the NormFet method exhibits higher sensitivity to the density of the point cloud. This sensitivity is reflected in higher LFS estimation errors when the point cloud is sparsely sampled.

Dual cone search. In order to evaluate the impact of parameters of the dual cone search, we conduct experiments with different settings for the apex angle θ and



(a) low density (b) moderate density (c) high density

Figure 4.8: LFS estimation with varying sampling density. From left to right: 648, 2,610, and 16,374 sample points. First row: input point clouds with increasing sampling density. Second row: LFS types: orange for diameter-based and green for curvature-based. Third row: estimated LFS error map for our algorithm, using the jet color ramp. Fourth row: estimated LFS error map for NormFet. Fifth row: distribution of estimated LFS errors for all sample points.

Capsule	Low (#648)		Moderate (#2610)		High (#16374)	
	mean(↓)	max(↓)	mean(↓)	max(↓)	mean(↓)	max(↓)
NormFet	5.216E-1	1.693	5.178E-1	1.692	2.136E-1	9.550E-1
(0°, 1)	7.130E-2	1.030	1.698E-1	1.030	1.848E-1	1.004
(5°, 10)	2.239E-2	9.743E-1	9.808E-3	9.915E-1	1.190E-3	9.978E-1
(5°, 100)	1.630E-2	9.743E-1	5.272E-3	9.779E-1	**7.781E-4**	**2.523E-2**
(10°, 10)	1.477E-2	9.743E-1	4.311E-3	**6.510E-2**	1.068E-3	**2.523E-2**
(10°, 100)	**1.023E-2**	**1.229E-1**	**4.168E-3**	**6.510E-2**	8.655E-4	**2.523E-2**

Table 4.2: LFS estimation on the capsule point cloud with increasing sampling density. (D°, N) denotes the parameters of the our dual cone search approach. D denotes the apex angle and N denotes the number of rays.

number of rays cast N on the capsule and the hippo point cloud. The dual cone search devised to estimate the local shape diameter at a sample point may not find the antipodal points when the parameters are not set properly, resulting in LFS estimation errors. Table 4.2 records estimation errors for the capsule point cloud with increasing density, apex angles, and numbers of casted rays. Shooting a single ray along the normal direction (0°, 1) always results in large errors for all sampling conditions. Increasing angle θ and the number of rays N helps reduce the errors significantly, but using small θ and N can still find the correct antipodal points. As the maximum absolute error remains the same, we are not required to set large θ and N. For example, for the capsule with high density (16,374 points), (5°, 100) and (10°, 100) have the same absolute errors even if θ differs. Similarly, (10°, 10) and (10°, 100) also have the same maximum absolute error even though the number of rays differs significantly. Figure 4.9 shows the LFS estimation on the hippo point cloud. The hippo point cloud is sparse in the body and dense in other parts, making it challenging to find the correct antipodal point for the body part if only shooting a single ray. Note that we lack ground truth LFS for the hippo. We perform min-max normalization in the log scale space to map the raw estimated LFS values for each hippo separately into the range $[0,1]$. Increasing θ and N can help to obtain a smoother color rmap.

Free-form point clouds. In this evaluation, we aim to assess the performance of our LFS estimation approach for point clouds sampled on free-form shapes, where ground-truth data is unavailable. To improve visualization, we apply smoothing to the raw estimated LFS values. Additionally, we perform min-max normalization

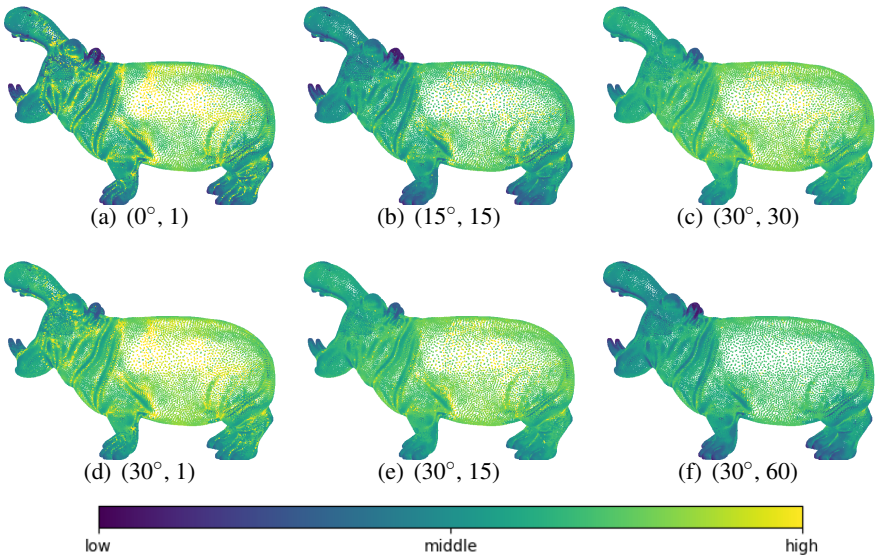
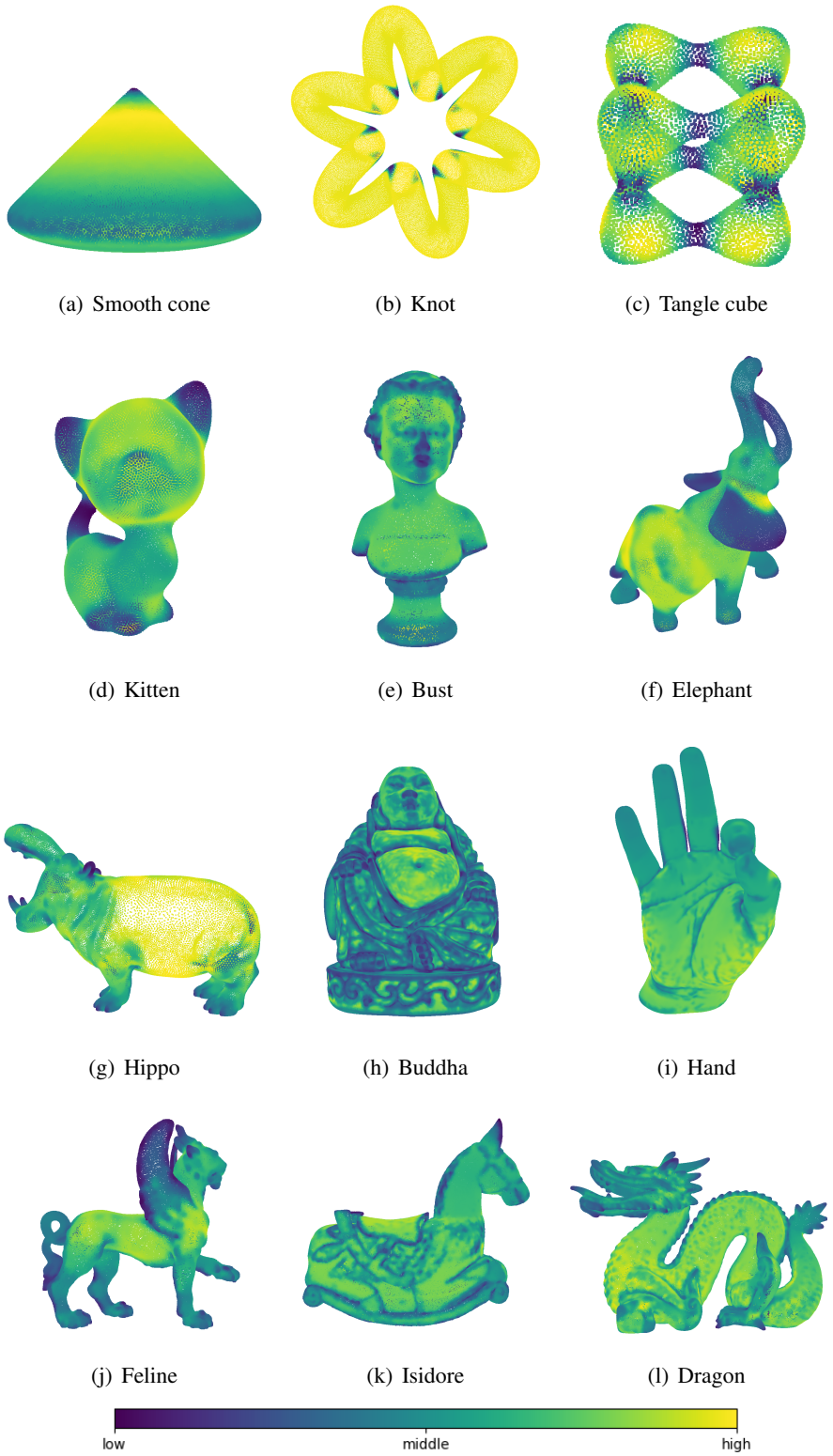


Figure 4.9: Visual ablation study on the hippo point cloud using different settings of the dual cone search. We vary the apex angle (D°) and the number of rays (N) to investigate their impact on the LFS estimation. The LFS values are visualized using the viridis color ramp without any smoothing applied.

in the log scale space to map the raw estimated LFS values for each point cloud separately into the range [0, 1]. This normalization allows us to highlight the distinctions in LFS variance across the point clouds. The visual results of our LFS estimation are presented in Figure 4.10. Through Figure 4.10, we observe the patterns and variations in LFS across different free-form shapes. In the case of the elephant, the ear region, being a thin structure, indeed relies heavily on the local thickness, resulting in a small LFS value. Conversely, the nose, characterized by a tube-like structure, is dominated by curvature, also leading to a relatively smaller LFS value compared to the rest of the body. However, for the body of the elephant, both the local curvature radius and thickness are larger than the nose and the ear region. This combination results in a much larger LFS value for the body.

4.3.2 Implicit Function

The implicit function is defined on the reach-aware 3D multi-domain, and represented as a piecewise linear function on a 3D Delaunay triangulation. The multi-domain is used to ensure that the triangulation is dense inside the envelope where the input point set is located, and sparser outside the envelope and inside the



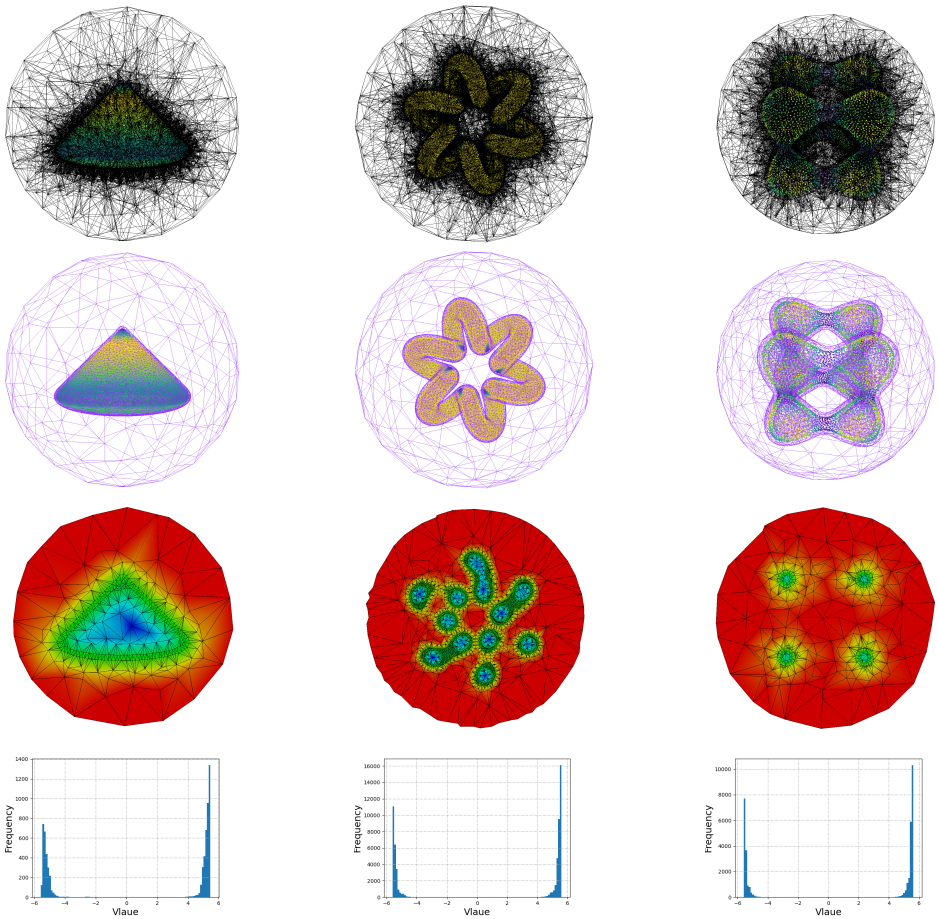
(a) Smooth cone (b) Knot (c) Tangle cube

(d) Kitten (e) Bust (f) Elephant

(g) Hippo (h) Buddha (i) Hand

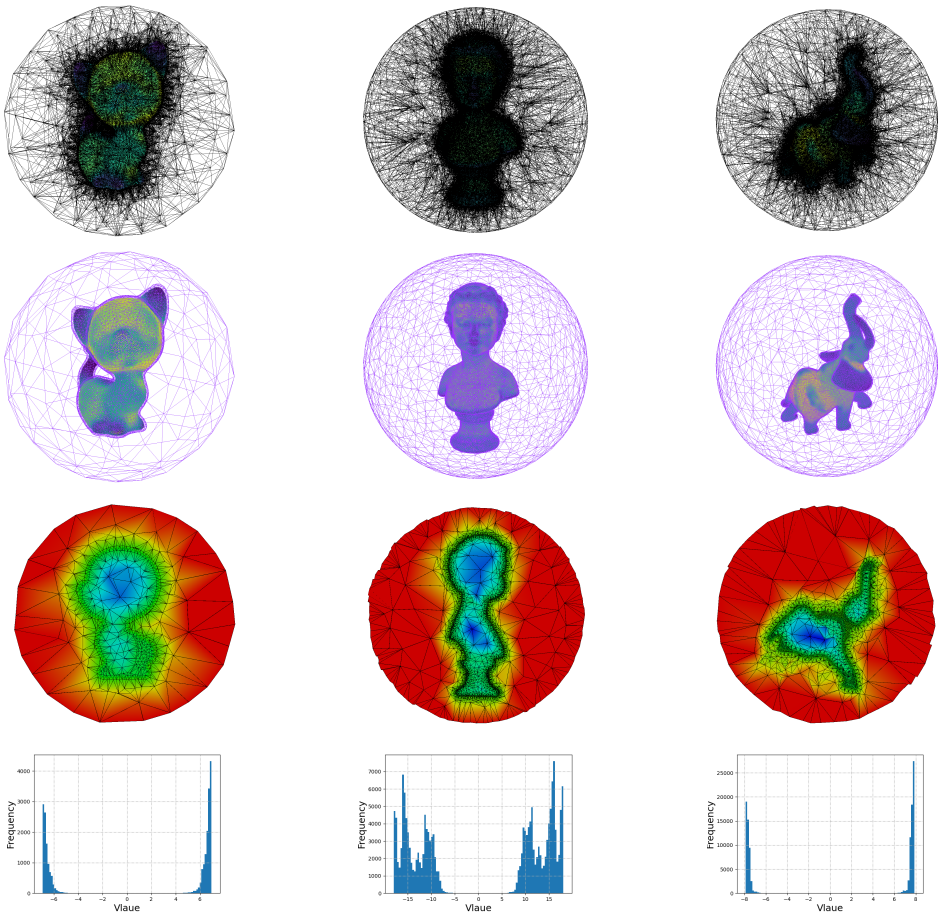
(j) Feline (k) Isidore (l) Dragon

Figure 4.10: LFS estimation for 3D point clouds sampled on free-form shapes. The viridis color ramp is used to map the smoothed LFS values for each point cloud.



(a) Smooth cone (b) Knot (c) Tangle cube

Figure 4.11: Implicit function for the smooth cone, knot, tangle cube 3D point clouds. First row: global depiction of the multi-domain. Second row: boundary of the multi-domain - the reach-aware envelope encloses the input points embedded in a bounded sphere. Third row: cut view of the piecewise linear implicit function. The tetrahedra are well-shaped by Delaunay refinement. Fourth row: distribution of solved signed implicit values at the triangulation vertices. The two main peaks correspond to two sets of vertices located inside and outside the inferred surface. Again, we find the thin envelope layer sandwiching the inputs, and the tetrahedrons are isotropic from the clip view.

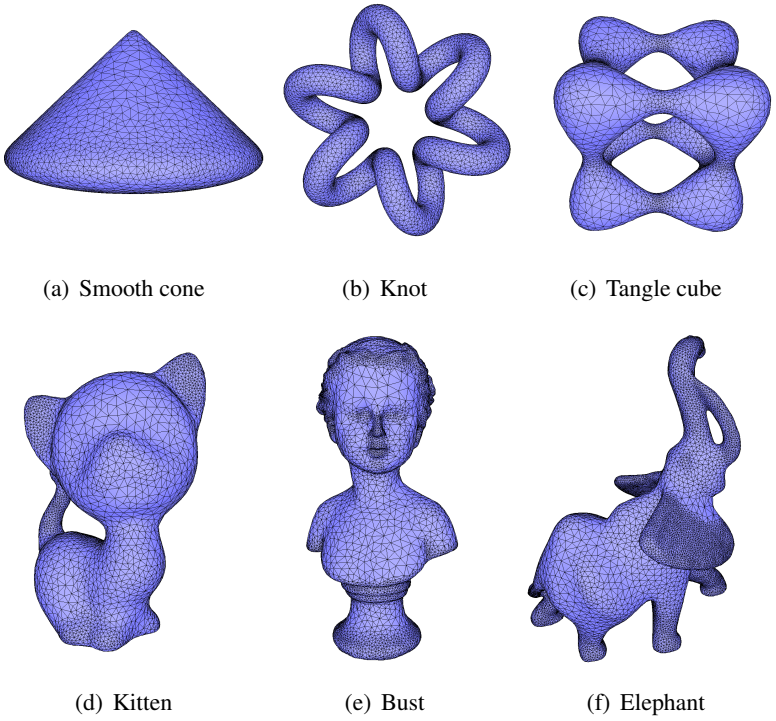


(a) Kitten (b) Bust (c) Elephant

Figure 4.12: More visual analysis of the implicit function for the kitten, bust, and elephant 3D point clouds. First row: global depiction of the multi-domain. Second row: boundary of the multi-domain - the reach-aware envelope encloses the input points embedded in a bounded sphere. Third row: cut view of the piecewise linear implicit function. The tetrahedra are well-shaped by Delaunay refinement. Fourth row: distribution of solved signed implicit values at the triangulation vertices. The two main peaks correspond to two sets of vertices located inside and outside the inferred surface. Again, we find the thin envelope layer sandwiching the inputs, and the tetrahedrons are isotropic from the clip view.

loose bounding sphere. The reach-aware property ensures that the piecewise linear function closely approximates the input point cloud. Meshing the entire bounding sphere increases the numerical stability of our solver. Figure 4.11 and Figure 4.12 depicts the implicit function defined on the reach-aware multi-domain.

4.3.3 LFS-Aware Meshing



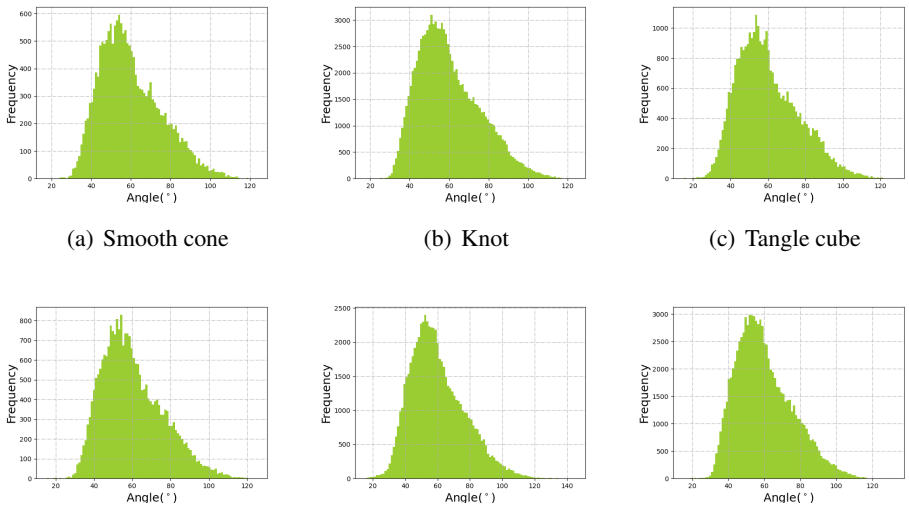
(a) Smooth cone (b) Knot (c) Tangle cube

(d) Kitten (e) Bust (f) Elephant

Figure 4.13: The reconstructed LFS-aware meshes with the histogram of triangle angles. The facet size varies regarding LFS, while the mesh remains isotropic.

The final output of our algorithm is an isotropic surface triangle mesh that dynamically adapts to the estimated LFS. We achieve this by generating the mesh through Delaunay refinement on the zero level-set of the signed robust distance function. To ensure the mesh is LFS-aware, we design a facet sizing function that makes facets larger in regions with larger LFS values and vice versa. In Figure 4.13, we present the output LFS-aware meshes, which showcase significant variation in facet sizing while accurately preserving the details of the input 3D point cloud. This adaptivity allows us to capture local features with higher precision, resulting in a more faithful representation of the underlying surface geometry with fewer triangles.

In addition to the LFS-aware surface reconstruction, we further verify the isotropy of the resulting mesh. For this purpose, we plot the distribution of facet angles for each mesh, as illustrated in Figure 4.14. The plot demonstrates that the distribution of facet angles centers around approximately 60°.



(a) Smooth cone (b) Knot (c) Tangle cube

(d) Kitten (e) Bust (f) Elephant

Figure 4.14: The distribution of facet angles.

We also investigate the influence of the user-defined parameter sizemax on the output meshes. This parameter plays a crucial role in controlling the maximum sizing of the facets during the meshing step, while sizemin serves as a fixed ratio to the thickness of the envelope. When sizemax is set to be equal to sizemin, the resulting output mesh becomes uniform and isotropic. However, as we increase size_{max}, the variance in facet sizing also increases, leading to a more adaptive mesh with varying facet sizes across the surface. To illustrate the impact of sizemax, we present visual results for the hippo 3D point cloud in Figure 4.15. This figure demonstrates how different values of sizemax affect the final mesh, highlighting the trade-off between mesh uniformity and adaptiveness.

4.3.4 Robustness

Noise, outliers, and missing data. We now evaluate the robustness of our surface reconstruction approach to noise, outliers, and missing data. We add Gaussian white noise at varying levels, ranging from 0.5% to 1.5% times the maximum edge length

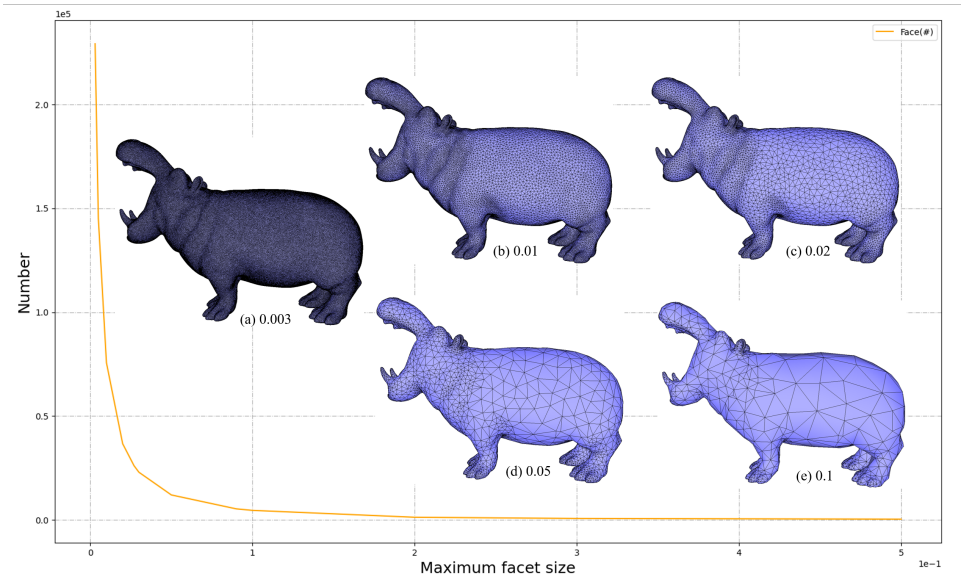
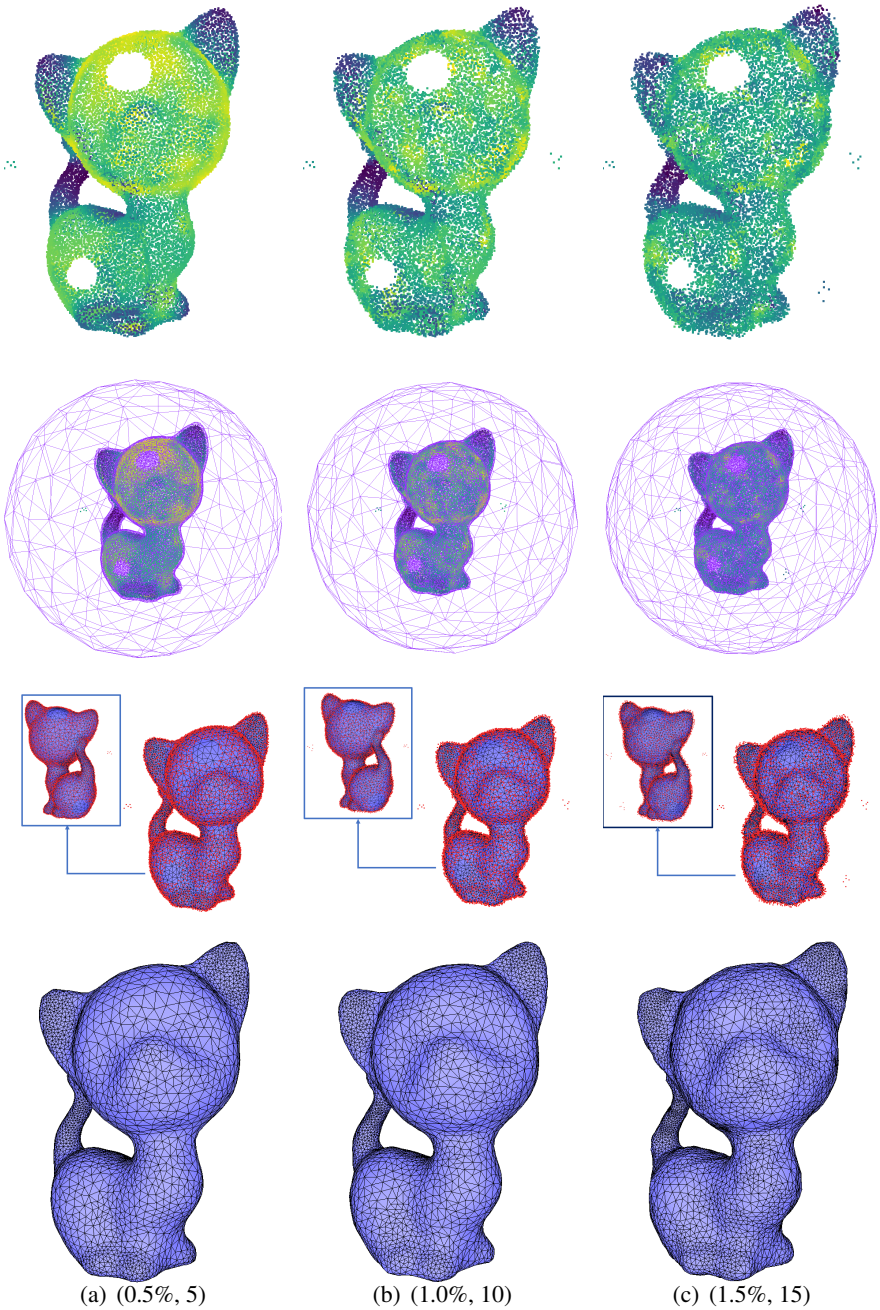


Figure 4.15: Output of reconstructed meshes with different size_{max} parameters on the hippo 3D point cloud. The numbers under each hippo denote size_{max}, which determines the maximum sizing of the facets in the output mesh. The curve plots the number of output facets against size_{max}.

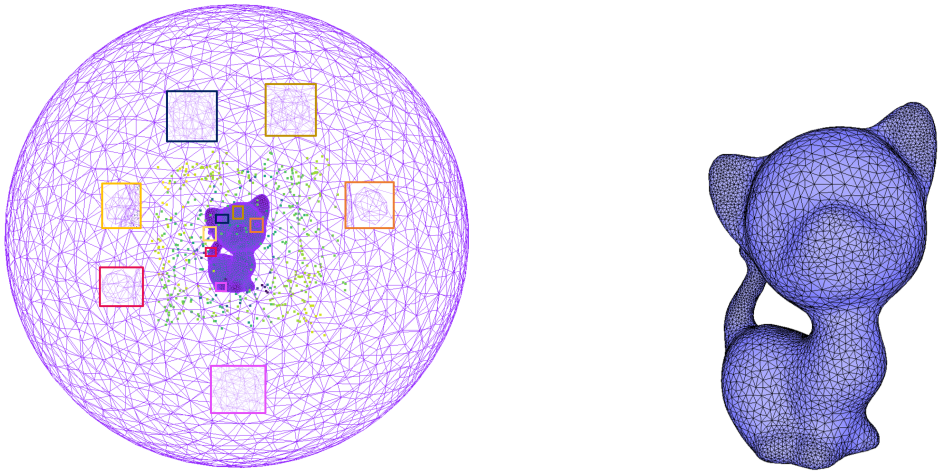
of the bounding box of the point cloud. Figure 4.16 illustrates the robustness of our approach on the kitten point cloud. We also introduce a moderate amount of random structured outliers at each noise level by generating 1 to 3 small clusters in the loose bounding sphere of the input, each containing 5 random points. Besides, we simulate missing data by creating two holes in the head and two holes in the body of the kitten point cloud. Our approach is robust to noise thanks to the property of the robust distance function. Our approach is also resilient to moderate amounts of outliers thanks to the Delaunay refinement process[JAYB15]. The Delaunay refinement can eliminate the outliers outside the envelope. Thus, there is no impact on the implicit function solver as all edges outside the envelope are considered to connect two vertices with similar signs.

We now evaluate the resilience to outliers of our implicit function solver by adding dense background outliers on the kitten point cloud. The dense outliers affect the Delaunay refinement, resulting in a handful of bubbles in the multi-domain discretization. Nevertheless, thanks to the stability of our implicit function solver, the bubbles do not affect the signing with the data fitting step. The final mesh has no artifacts, as shown by Figure 4.17.



(a) (0.5%, 5) (b) (1.0%, 10) (c) (1.5%, 15)

Figure 4.16: Evaluation of robustness on the kitten point cloud with increasing noise levels, outliers, and missing data. The noise level increases from 0.5% to 1.5%, and the number of outliers increases from 5 to 15. First row: estimated LFS. Second row: the boundary of the multi-domain ignores the outliers. Third and fourth rows: output LFS-aware meshes.

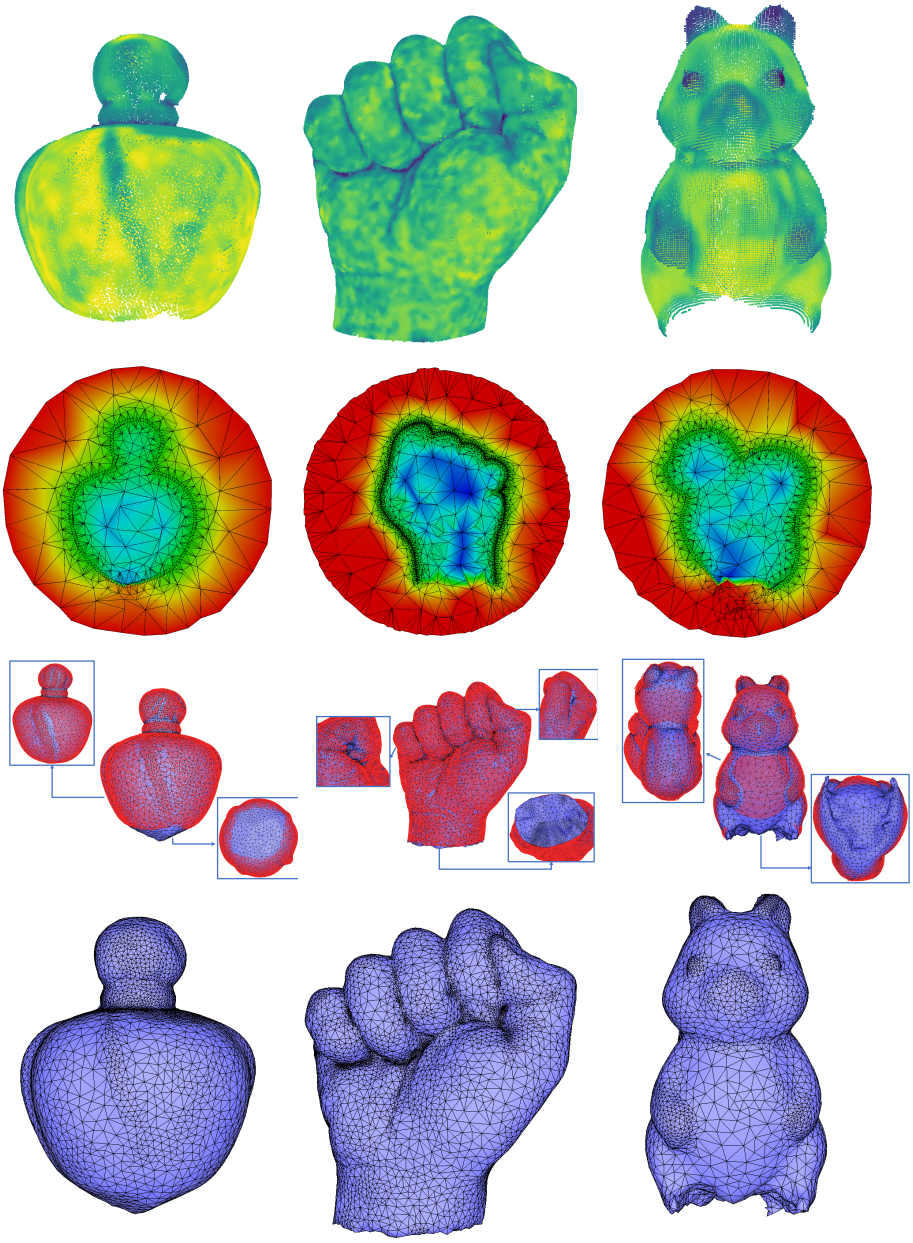


(a) Multi-domain (b) Mesh

Figure 4.17: Resilience to outliers evaluated on the kitten point cloud with 500 random background outliers. A handful of bubbles are generated around some outliers inside the multi-domain discretization. A closeup of selected bubbles is delineated with a colored rectangle. Note that some bubbles are inside the point cloud, thus blocked by the front views. The bubbles do not affect the final result because the implicit function solver erases them.

Large hole filling on real-scan data. In our experiments, we aim to validate the effectiveness of our algorithm in filling large holes present in real-world data acquired by laser scanners. Figure 4.18 showcases the results of hole-filling on such data, which is often characterized by non-uniform sampling and measurement noise, posing challenges in accurately filling the holes. The ability to fill large holes is made possible by our mesh solver. On areas where the point set is dense, we observe a high density of edges with opposite sign guesses. Areas near holes are more ambiguous, with a mixture of edges with opposite or similar signs. Intuitively, the signing step is robust to large holes (with many missing data) thanks to the least-squares solver that propagates the sign on the triangulation vertices via consolidating sign guess hypotheses emitted for all triangulation edges.

Complex topology. We evaluate the capability of our algorithm to deal with shapes with non-trivial topology by reconstructing from the filigree point cloud. Adding increasing levels of noise translates into reconstructions that are robust to low levels of noise and that degrade gracefully with noise. The high genus of the



(a) Perfume bottle (b) Hand kreon (c) Squirrel

Figure 4.18: Reconstructions on real-world data with large holes. First row: estimated LFS. Second row: clipped view of the signed robust distance function. Third and fourth rows: LFS-aware meshes.

filigree is captured even when the surface becomes distorted. For high levels of noise, the output mesh breaks into multiple connected components, demonstrating the limitations of the algorithm under extreme noise conditions. Refer to Figure 4.19 for the visual depiction.

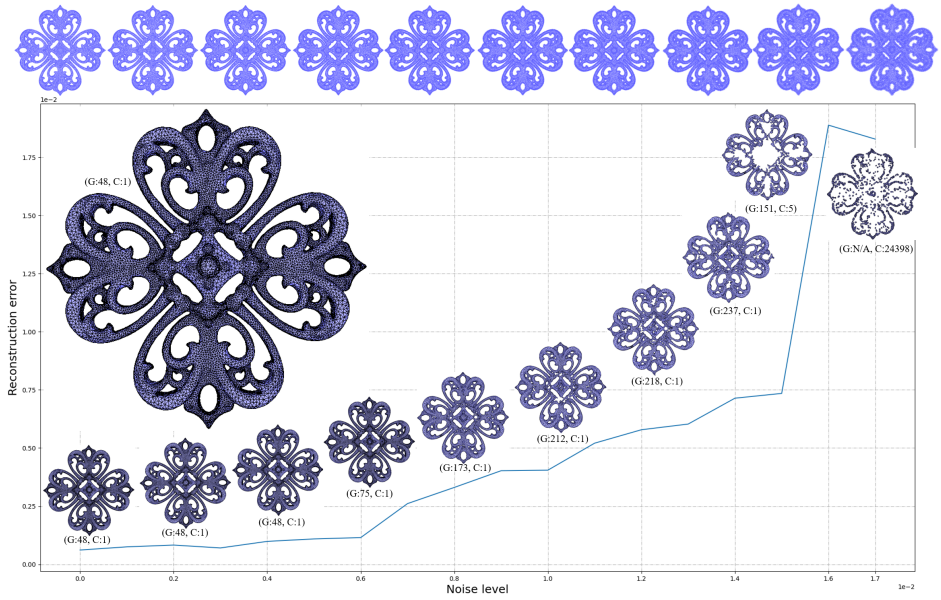
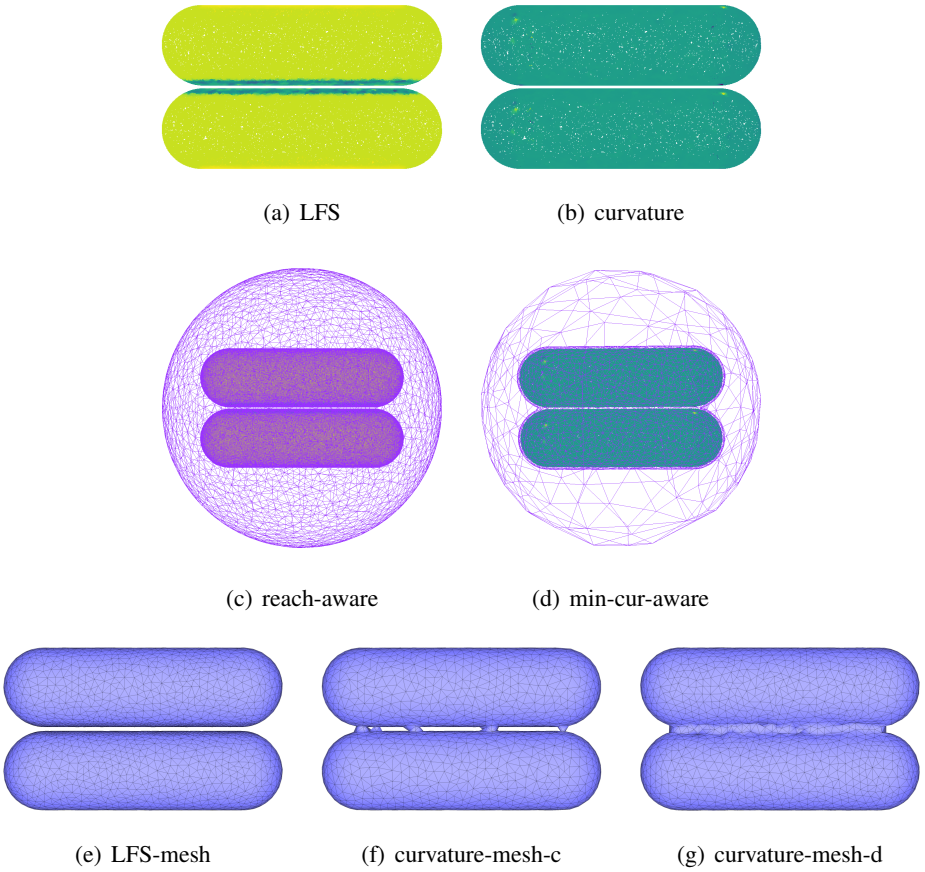


Figure 4.19: Reconstructing the filigree point cloud with increasing levels of noise. The largest mesh depicts the output for the noise-free case. The other series of meshes depicts the evolution of the reconstructed genus when noise increases. We record the genus ("G") and number of connected components ("C").

4.3.5 Ablation Studies

LFS vs. curvature. We now conduct experiments to verify the added value of using LFS instead of curvature. A first experiment is conducted on two adjacent capsules with a smaller separation distance than the curvature radius, see Figure 4.20. We use three settings: (1) The estimated minimum LFS is used to construct the reach-aware envelope, and LFS is used for sizing the facets; (2) We also construct a reach-aware envelope but use only the curvature for sizing the facets; (3) We use only the minimum curvature radius to construct the envelope and the curvature to size the facets. Setting (1) yields the best results where the curvature and topology are captured, while settings (2) and (3) fail to reconstruct the topology as the two capsules merge. More specifically, using only the minimum curvature considers



(a) LFS (b) curvature

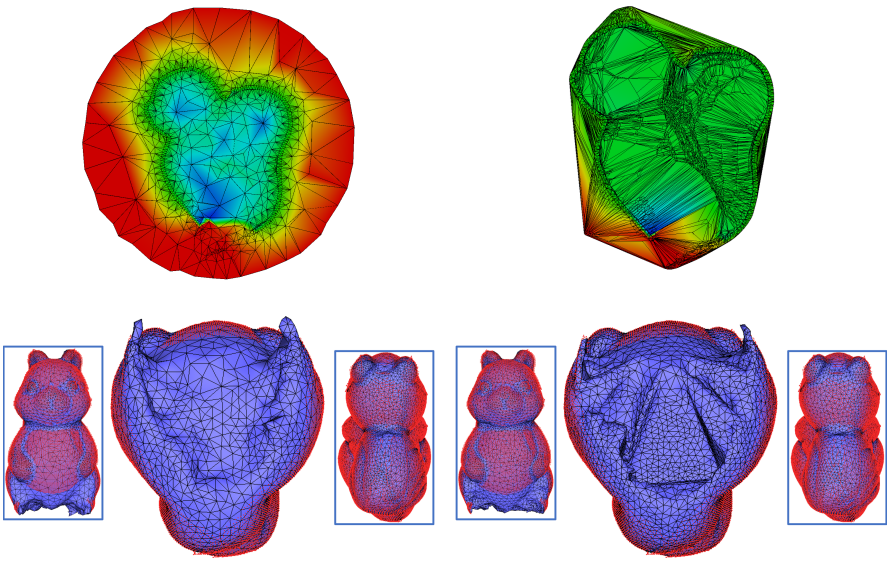
(c) reach-aware (d) min-cur-aware

(e) LFS-mesh (f) curvature-mesh-c (g) curvature-mesh-d

Figure 4.20: LFS vs. curvature. (a) "LFS" is the estimated LFS. (b) "curvature" is the estimated curvature. (c) "reach-aware" is the reach-aware multi-domain. (d) "min-cur-aware" is the minimum-curvature-aware multi-domain. (e) "LFS-mesh" is the LFS-aware mesh extracted from the implicit function solved on (c). (f) "curvature-mesh-c" is the curvature-aware mesh extracted from the implicit function solved on (c). (g) "curvature-mesh-d" is the curvature-aware mesh extracted from the implicit function solved on (d).

neither the thickness nor the separation, thus leading to a thicker envelope and insufficient resolution for the implicit function. In addition, using only the curvature for sizing the output mesh is insufficient even when the implicit function is solved on the reach-aware domain, as the Delaunay refinement process fails to separate nearby surface sheets see Figure 4.20(f). The middle of the two capsules is connected as the size of the facets derived from the curvature is larger than the separation distance.

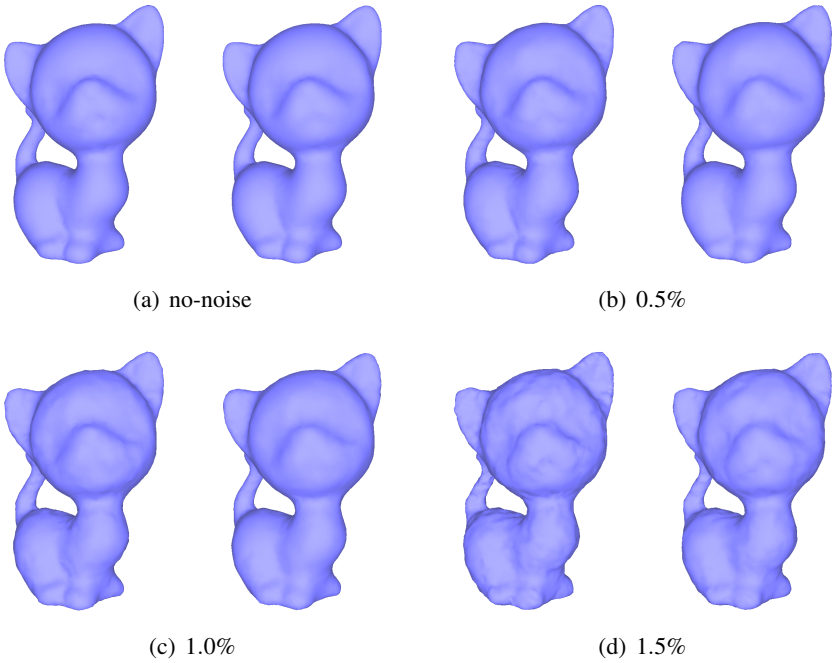
Multi-domain vs. single-domain. We now verify the added value of using a multi-domain discretization over a single-domain discretization. The single-domain is the 3D Delaunay triangulation of the reach-aware envelope whose boundary is the convex hull. In this case, the tetrahedron cells are isotropic inside the envelope and skinny outside the envelope. In contrast, the tetrahedron cells are isotropic everywhere for the multi-domain discretization, with smaller cell sizes inside and larger cell sizes outside the envelope. Such isotropic cells help fill holes with smoother surface patches, see Figure 4.21.



(a) Multi-domain (b) Single-domain

Figure 4.21: Multi-domain vs. single-domain on the squirrel point set (third column in Figure 4.18). The holes filled via multi-domain discretization are smoother than their single-domain counterpart.

Signed robust distance function. The robustness of the signed robust distance function $\hat{d}_s$ in Eq. (4.15) is illustrated by Figure 4.22. Solving for Eq. (4.14) yields the initial signed implicit function d_s. However, we observed that the zero level-set of this initial function is not sufficiently robust to noise. To address this issue and enhance the robustness of our approach, we utilize the sign of the solved implicit function to combine it with the unsigned robust distance function. This combination yields an increased resilience to noise, resulting in a more robust and accurate representation of the surface geometry.



(a) no-noise (b) 0.5%

(c) 1.0% (d) 1.5%

Figure 4.22: Signed implicit function d_s vs. signed robust distance function $\hat{d}_s$ under different levels of noise (0, 0.5%, 1.0%, and 1.5%) on the kitten point cloud. For each pair of kittens, the left is the mesh extracted from the signed implicit function, while the right is the mesh extracted from the signed robust distance function.

Why not remeshing. Our method can extract LFS-adaptive mesh directly from the input point clouds. Aside from the solved implicit function, an LFS-adaptive mesh sizing function is constructed from the LFS, which is also derived directly from the input point cloud. However, previous pipelines are cascaded, which involves implicit/explicit reconstruction followed by remeshing. Thus, if the initial reconstruction encounters difficulties, subsequent remeshing steps may struggle

to rectify the issues. In contrast, the implicit function solver and mesh sizing function are calculated separately. If one fails, it won't affect the other step. To illustrate the benefits, we use the feline point cloud whose point cloud is shown in Fig. 4.10, whose wings have a thin structure. Both iPSR [HWW+22a] and NKSR [HGA+23] failed to reconstruct the thing wings, thus affecting the following remeshing. In contrast, our method can preserve the thing wings while maintaining lower reconstruction errors (point-to-mesh Chamfer distance). Please refer to Fig. 4.23.

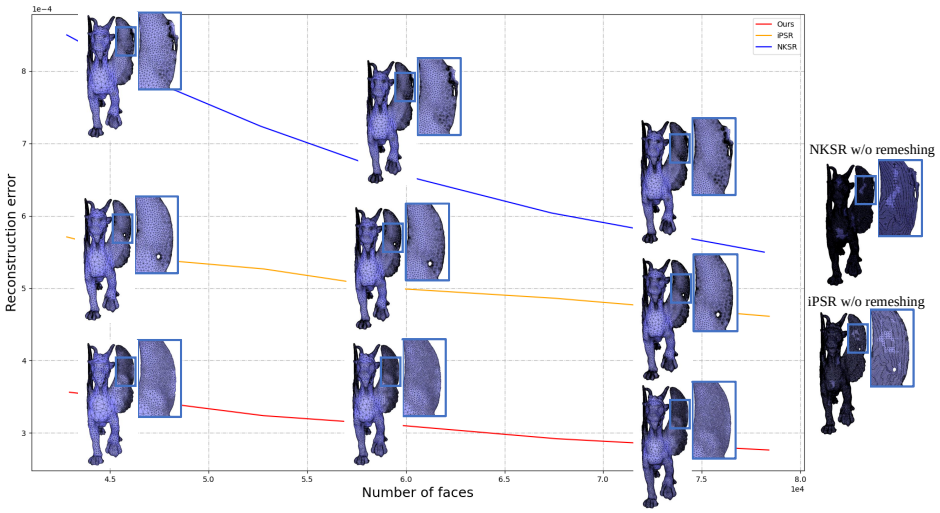


Figure 4.23: Experiments on the feline point cloud. We plot the reconstruction error against the number of faces and attach the reconstructed feline with and without the remeshing for iPSR [HWW+22a] and NKSR [HGA+23]. We also attach the reconstructed feline by methods with different numbers of faces. Note that all algorithms output roughly the same mesh complexity in terms of number of faces.

4.3.6 Sharp Feature Preservation

In our approach, we employ the Delaunay refinement algorithm [JAYB15] to extract the final mesh. Notably, if sharp feature lines are provided as input, our algorithm can effectively preserve these sharp features because of the Delaunay refinement. To detect these sharp feature lines from the raw point clouds, we utilize the NerVE algorithm [ZDC+23], as illustrated in Figure 4.24, which predicts sharp feature lines by detecting edge occupancy, edge orientation, and edge point locations from voxels. Figure 4.25 illustrates the results of sharp feature lines for point clouds

sampled from the CAD models.

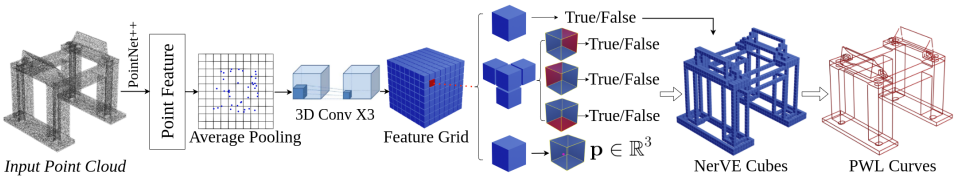


Figure 4.24: Overview of the NerVE network. Given a 3D point cloud, the network first utilizes a simplified PointNet++ [QYSG17] module and a 3D CNN module to obtain the feature grid. Three individual decoders are applied to process cube features in the grid to predict the corresponding three attributes of NerVE, i.e., edge occupancy, edge orientation, and edge point locations. By converting the NerVE cubes into piecewise linear curves, NerVE can obtain the parametric curves of the shape with post-processing. Image taken from [ZDC+23].

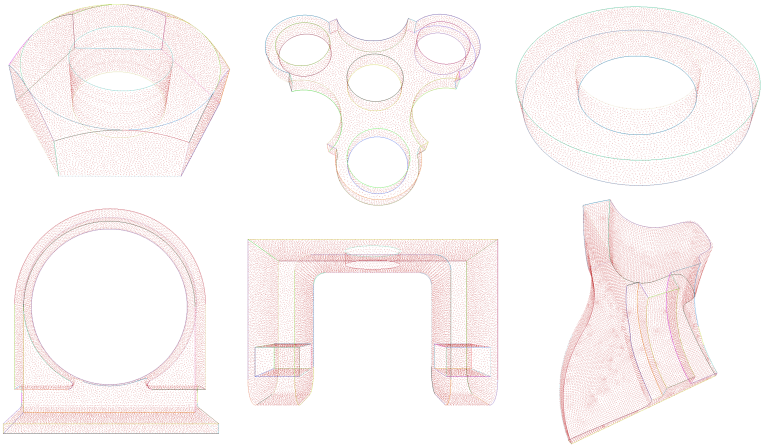


Figure 4.25: Sharp feature curves produced by NerVE. Red is used to depict the input 3D point cloud, and the colored line is the predicted sharp feature line.

Equipped with the sharp feature lines extracted by NerVE, we can plug them into our surface reconstruction algorithm to preserve the sharp features for the reconstructed mesh. We also compare our method with RFEPS [XWD+22], which preserves the sharp features implicitly. Based on our experiments, our approach can deal with more challenging cases than RFEPS. See Figure 4.26.

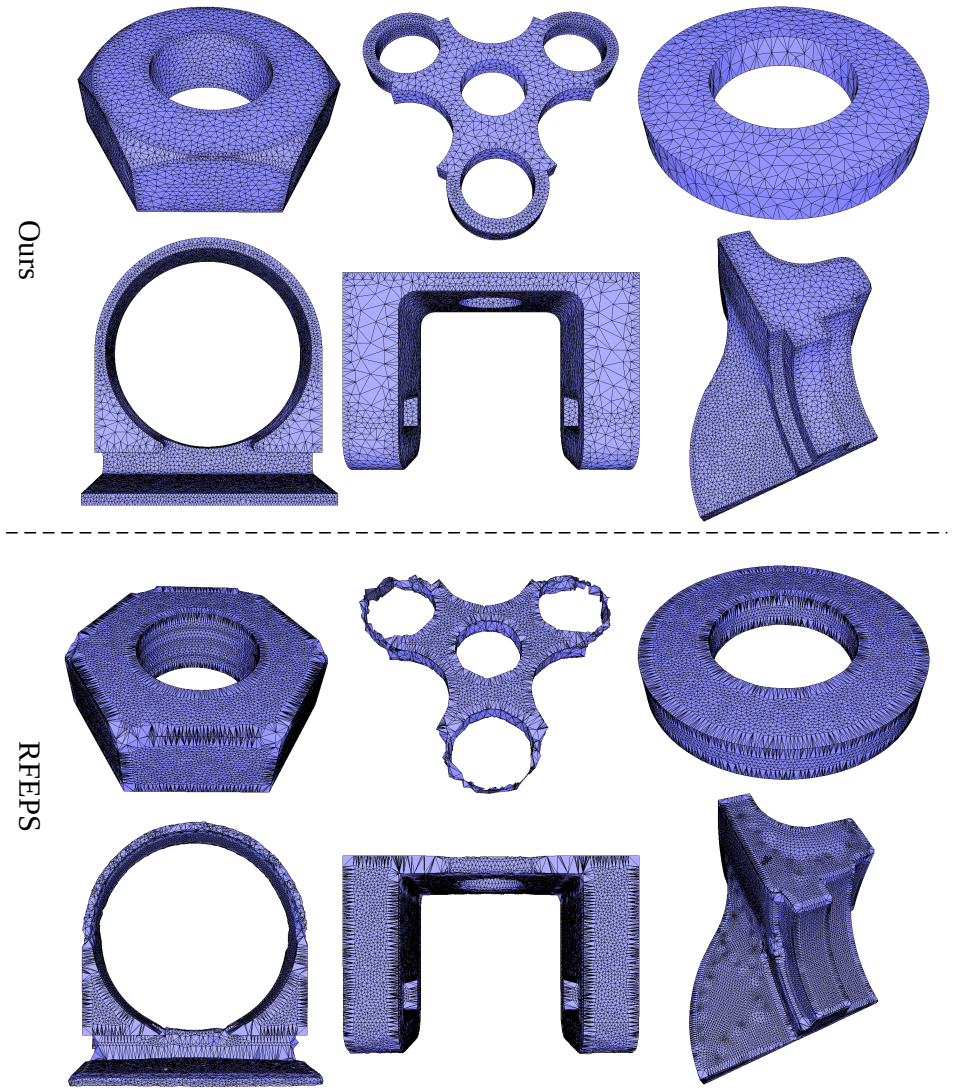


Figure 4.26: Comparisons of sharp feature preserving reconstruction with RFEPS [XWD+22].

4.3.7 Comparisons

We conduct more experiments to compare our algorithm with other baseline methods. Comparisons with RFEPS [XWD+22] on CAD models with sharp features are already presented in Fig. 4.26. We choose other baseline mehtods as AMLS [DS05], screened Poisson surface reconstruction (sPSR) [KH13], iPSR [HWW+22a] and NKSR [HGA+23]. AMLS fails into the type of explicit reconstruction that also uses LFS as a prior. In contrast, sPSR and iPSR fall into the category of implicit surface reconstruction, and NKSR is the learning-based method. In addition, to obtain isotropic meshes with LFS-adaptive size, we perform LFS-based remeshing using the Geogram library [LF15]. This involves sampling points on the reconstructed raw mesh and constructing the Voronoi diagram to estimate the LFS, which is then used to size the facets.

To make a fair comparison, we achieve roughly the same mesh complexity in terms of the number of triangles for all algorithms by controlling the meshing/remeshing process. We then evaluate the reconstruction fidelity using the point-to-mesh Chamfer distance and the Hausdorff distance. Additionally, we compute the average angles of triangle facets to verify the isotropy of the output meshes. Figure 4.27 and Figure 4.28 provides visual comparisons, and Table 4.3 records the evaluation metrics. Our approach achieves the smallest reconstruction errors for both the Chamfer and Hausdorff distance, and a visual inspection shows that it preserves more details, such as the back of the Buddha and the tail of the isidore horse point cloud. Our approach better captures the topology by leveraging the LFS estimation, while AMLS and NKSR fail to reconstruct the Metatron and PI. point cloud. Besides, the output of our method is water-tight, while NKSR can leave holes. We mark those holes in the red rectangle in the Figure 4.27 and 4.28.

Model	Chamfer(↓)					Hausdorff(↓)					Face(#)				
	A+T+R	sPSR+R	iPSR+R	NKSR+R	Ours	A+T+R	sPSR+R	iPSR+R	NKSR+R	Ours	A+T+R	sPSR+R	iPSR+R	NKSR+R	Ours
DC.(#724k)	4.88E-1	1.37E-2	1.19E-2	1.07E-2	**6.10E-3**	3.99	1.72E-1	1.65E-1	1.48	**1.63E-1**	95k	92k	92k	92k	92k
Metatron(#69k)	4.73E-1	6.35E-2	5.72E-2	6.32E-1	**4.76E-2**	4.01	6.11E-1	5.49E-1	5.98	**5.40E-1**	61k	61k	61k	64k	61k
PI.(#9k)	3.03E-1	1.79E-1	1.48E-1	1.53	**9.69E-2**	9.31E-1	1.92	7.20E-1	7.24	**4.02E-1**	15k	15k	15k	15k	15k
Buddha(#719k)	4.60E-4	3.50E-4	3.05E-4	3.13E-4	**2.49E-4**	3.81E-3	4.30E-3	3.78E-3	3.71E-3	**2.73E-3**	128k	128k	128k	128k	128k
Gargoyle(#963k)	6.12E-2	1.34E-2	1.03E-2	1.31E-2	**4.82E-3**	6.49E-1	1.89E-1	1.23E-1	2.17	**1.21E-1**	180k	180k	180k	180k	180k
Hand(#369k)	1.12E-3	2.09E-4	2.14E-4	2.28E-4	**2.06E-4**	7.08E-3	5.31E-3	5.60E-3	6.48E-3	**5.28E-3**	35k	35k	35k	35k	35k
Isidore(#235k)	4.12E-4	3.78E-4	3.62E-4	3.99E-4	**3.15E-4**	6.93E-3	7.09E-3	6.95E-3	7.18E-3	**6.91E-3**	72k	72k	72k	72k	72k
Dragon(#296k)	1.02E-3	3.05E-4	1.85E-4	3.10E-4	**1.60E-4**	3.29E-2	1.44E-2	1.37E-2	9.98E-3	**6.99E-3**	164k	158k	158k	158k	158k

Table 4.3: Comparisons with different algorithms. “A+T+R" denotes the pipeline of AMLS reconstruction, tight cocone, and remeshing.“sPSR+R" denotes the pipeline of screened Poisson surface reconstruction followed by remeshing. “iPSR+R" denotes the pipeline of iterative Poisson surface reconstruction followed by remeshing. “NKSR+R" denotes the pipeline of NKSR followed by remeshing.

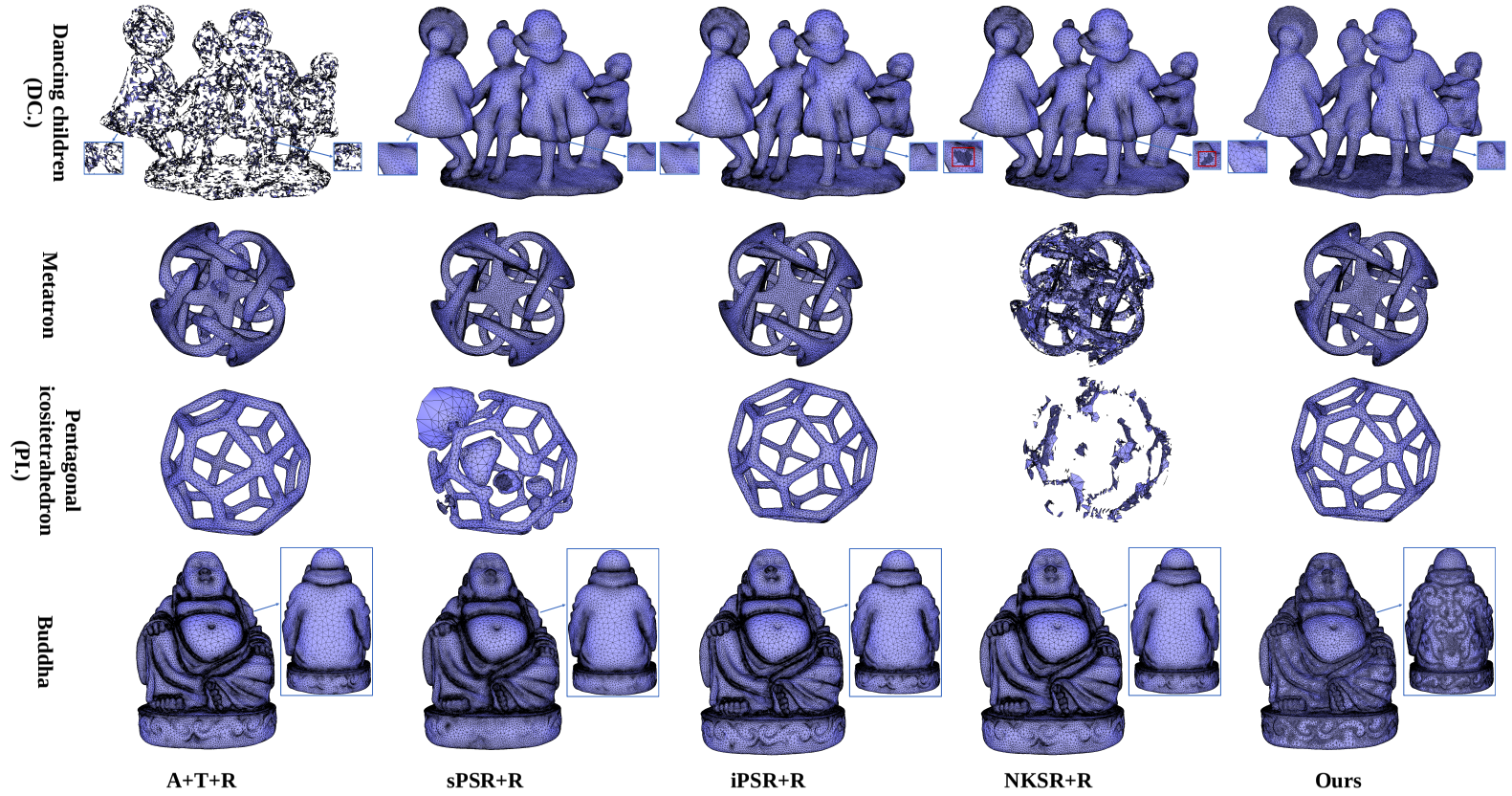
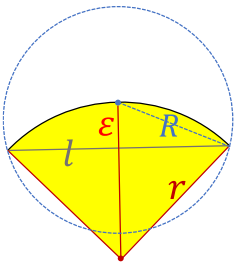


Figure 4.27: Visual comparisons with different algorithms.“A+T+R" denotes the pipeline of AMLS reconstruction, tight cocone, and remeshing.“sPSR+R" denotes the pipeline of screened Poisson surface reconstruction followed by remeshing. “iPSR+R" denotes the pipeline of iterative Poisson surface reconstruction followed by remeshing. “NKSR+R" denotes the pipeline of NKSR followed by remeshing.

Figure 4.28: Visual comparisons with different algorithms.“A+T+R" denotes the pipeline of AMLS reconstruction, tight cocone, and remeshing.“sPSR+R" denotes the pipeline of screened Poisson surface reconstruction followed by remeshing. “iPSR+R" denotes the pipeline of iterative Poisson surface reconstruction followed by remeshing. “NKSR+R" denotes the pipeline of NKSR followed by remeshing.

4.3.8 Reconstruction Error Analysis

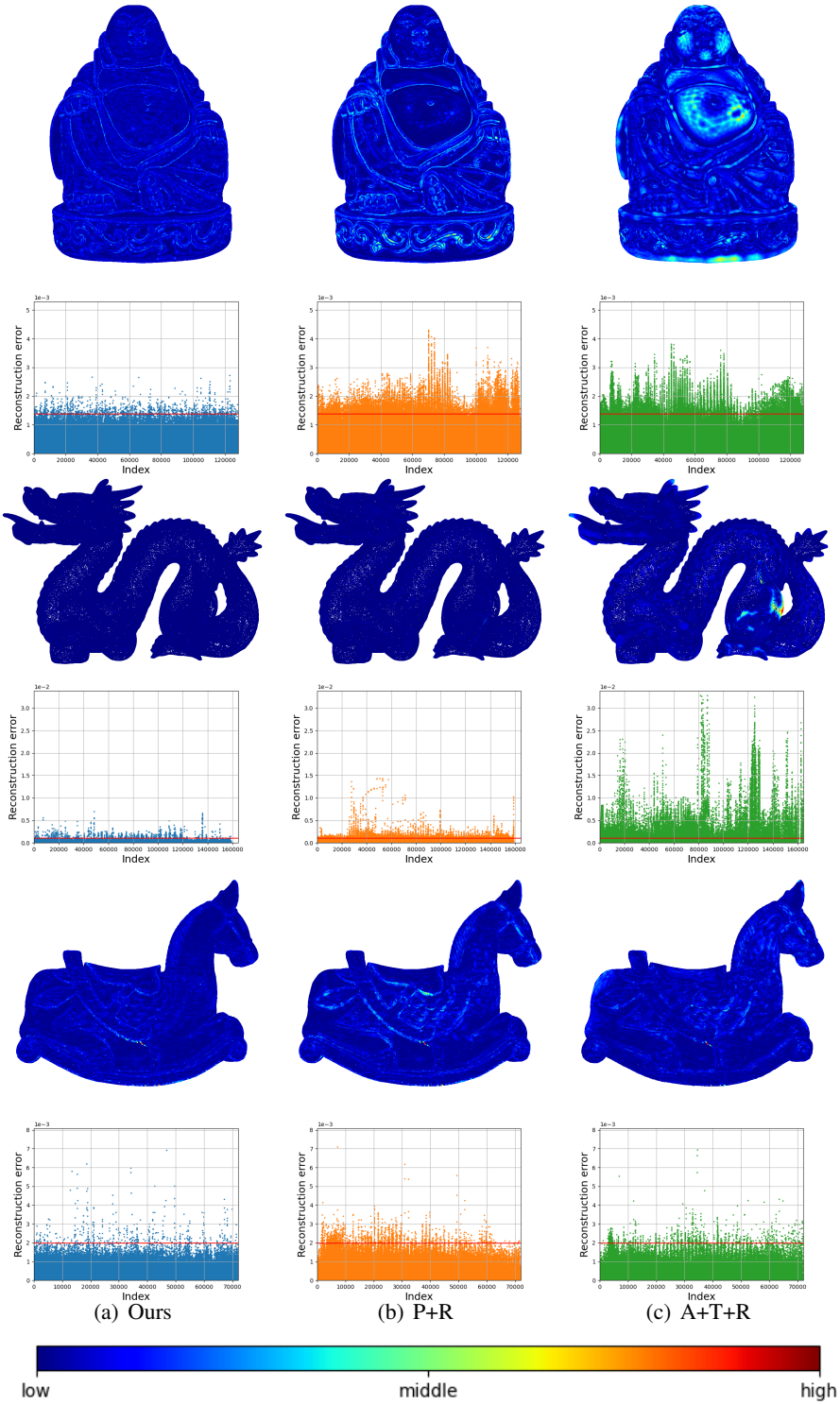
We now evaluate the reconstruction errors using the one-sided distance from the input points to the reconstructed surface mesh. For each facet in the 2D configuration, we use ε to denote the reconstruction error from the zero level-set of the implicit function to the reconstructed surface triangle mesh. We use l to represent the local facet size and R to represent the local surface Delaunay ball. The local surface Delaunay ball, as described in [JAYB15], has its center on the surface and circumscribes the local facet, with no other vertices located inside the ball. By taking into consideration the local curvature radius r and applying the Pythagorean theorem to this configuration, we derive an estimation for the reconstruction error ε in terms of r and the surface Delaunay ball radius R. The relationship is given as $\varepsilon = R^2/(2r)$.



The surface Delaunay ball radius R can provide an upper bound for the local facet size l during the Delaunay refinement process [JAYB15]. For simplicity and ease of analysis, we choose to use R instead of l in our reconstruction error analysis. Note that the surface reach, representing the minimum local feature size, corresponds to areas where the local curvature radius is smaller than the local thickness/separation, or vice versa. By assuming that the surface reach corresponds to a curvature radius, the surface reach area will achieve the maximum reconstruction error. Thus, we can analyze the maximum reconstruction error as a function of the estimated reach and the radius of the corresponding surface Delaunay ball as

$$\varepsilon \leq \frac{R_{min}^2}{2I_R}, \tag{4.17}$$

where R_{min} is the radius of the surface Delaunay balls for the smallest facet, and I_R is the estimated reach. This formula suggests an estimation of the upper error bound for the reconstruction. Fig. 4.29 verifies the consistency with respect to the formula and compares the per-facet reconstruction error with other methods. The red line depicts the maximum analytical error by Equation 4.17. Furthermore, the per-facet reconstruction error scatter plots show that our local maxima reconstruction errors are uniformly distributed compared to other methods.



(a) Ours (b) P+R (c) A+T+R

Figure 4.29: Reconstruction errors for the buddha, dragon and isidore horse point clouds. The jet color ramp is used to depict the reconstruction error for each facet. The red horizontal line is the upper reconstruction error bound from (4.17), which is also displayed in the plot of reconstruction error for other methods.

4.3.9 Time and Memory

We now measure the computational time and memory consumption through a series of experiments. We first generate a series of point clouds sampled on an ellipsoid using an increasing density. The memory usage is almost linear with respect to the number of input points. We also measure time and memory consumption on the hippo point cloud under different user-defined parameters (maximum facet size size_{max}). The computation time and memory usage for estimating LFS and solving the implicit function are constant because the input point cloud is unchanged. However, the total time and peak memory differ depending on size_{max}. Fig. 4.30 plots the computation time (in seconds) and memory usage (in MBytes) against the number of input points or size_{max}.

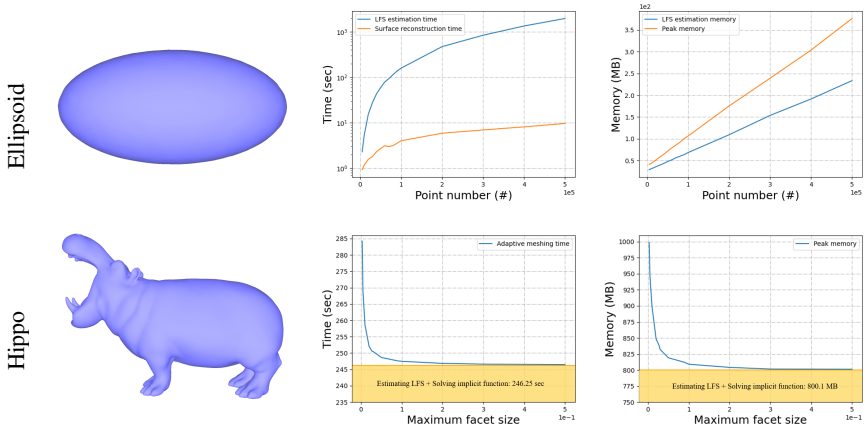


Figure 4.30: Computation time and memory consumption. First row: ellipsoid with increasing density of the input 3D point cloud. Second row: experiments on the hippo 3D point cloud (58,188 points) with varying the maximum facet size parameter (size_{max}).

4.4 Conclusion

This chapter introduces a novel approach for estimating the local feature size (LFS) and leveraging it for LFS-aware surface reconstruction from unorganized 3D point cloud data. The primary output of our method is a valid and intersection-free isotropic triangle surface mesh with facet sizing that captures the estimated LFS. One of the key contributions of our approach is the incorporation of LFS into the surface reconstruction process. By doing so, we generate a mesh with, ideally,

"just-enough" complexity, striking a balance between capturing fine details in the data while maintaining robustness to various data defects such as noise, outliers, or missing data. This results in a more accurate and visually pleasing reconstructed surface. Our experiments have demonstrated the robustness of our LFS estimation method, showing that it can handle moderate noise and non-uniform sampling effectively. Additionally, our surface reconstruction algorithm outperforms some existing approaches in terms of preserving details and accurately representing thin topological features. As part of our future work, we plan to extend our approach to address boundaries, which are essential for capturing more complex and intricate geometries accurately. By expanding our method's capabilities to handle these challenging scenarios, we aim to provide an even more comprehensive and versatile tool for surface reconstruction from point cloud data.

CHAPTER 5

Conclusion and Perspectives

5.1 Conclusion

The advancement of 3D scanning technologies has addressed an increasing number of use cases where vast amounts of 3D point cloud data are used for digital twinning. In response to the increasing demand for effective point cloud processing, this thesis has explored the problem of shape reconstruction from raw 3D point clouds. This problem, characterized by its multifaceted challenges, touches several domains including computer graphics and computer vision. The thesis has delved into two main facets of this problem: **primitive segmentation** and **surface reconstruction**.

Primitive segmentation In Chapter 3, we presented a novel approach for generic primitive segmentation, representing one contribution to this thesis. We also summarize the strengths and weaknesses of different theoretical frameworks in Figure 5.1. The conventional paradigm in primitive segmentation often involves addressing different primitive types as isolated entities, each requiring specialized optimization techniques. However, our approach seeks to avoid these constraints by offering a more generalized and adaptable framework. We introduced BPNet, a novel deep-learning framework to segment the primitives within 3D point clouds. Unlike conventional methods that treat different primitive types separately, BPNet offers a more generalized and adaptive approach. Furthermore, we conducted comprehensive experiments to evaluate the performance of BPNet, showcasing its capabilities in handling a wide array of primitive types and enhancing segmentation accuracy and efficiency. Since we use a generic primitive segmentation, our method is especially fast compared to the baselines. Moreover, by removing the constraints of predefined primitive categories, our approach can be applied to reconstruct shape while preserving sharp features by only considering the intersection of different patches.

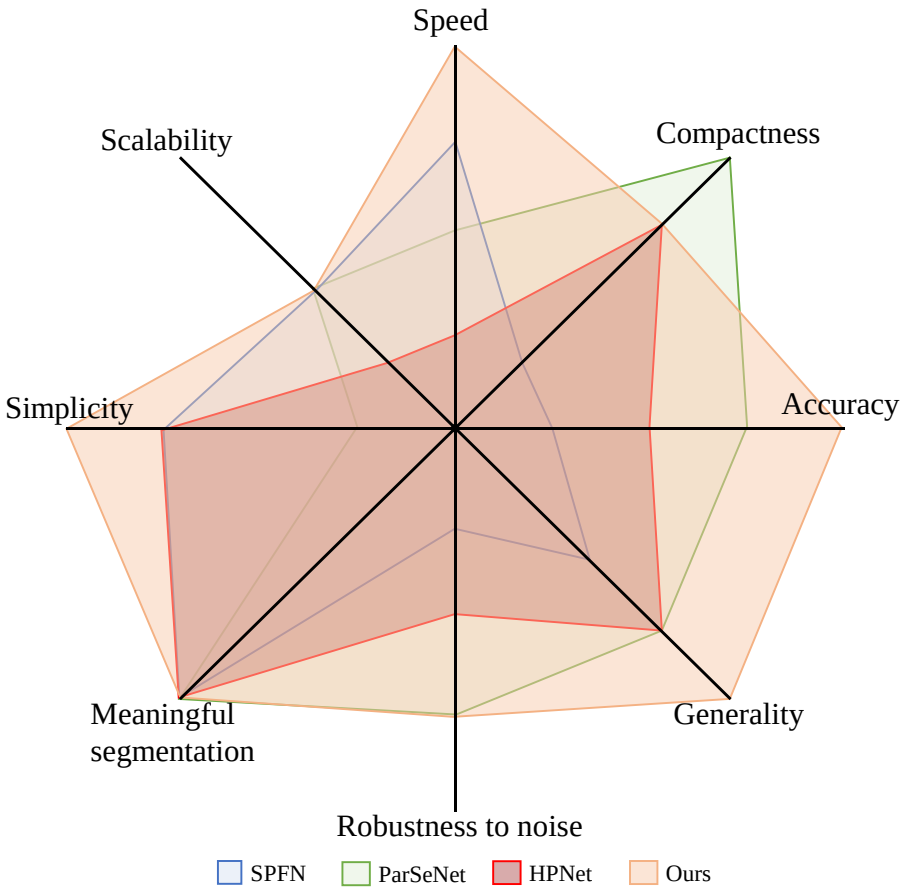


Figure 5.1: Qualitative comparison of the proposed primitive segmentation method with chosen baseline methods.

Surface Reconstruction In Chapter 4, we explored surface reconstruction, where we introduce a novel method for reconstructing isotropic triangular meshes from defect-laden point clouds. We summarize the strengths and weaknesses of different methodological frameworks in Figure 5.2. Traditional surface reconstruction methods prioritize fidelity in fitting the reconstructed mesh to the input 3D point cloud, thus ignoring the quality of the mesh triangles that should be well-shaped. Such fidelity-focused reconstructions often yield non-isotropic meshes, requiring post-processing steps like remeshing to improve the mesh quality. Our method, in contrast, places a primary emphasis on adaptability to local feature size (LFS) via a Delaunay refinement meshing, ensuring that the reconstructed meshes naturally

possess isotropy without the need for additional remeshing. This distinctive characteristic underscores the adaptability and robustness of our approach. Within this chapter, we conducted extensive experiments on LFS estimation, implicit function reconstruction, and LFS-aware mesh sizing, demonstrating how these components work in concert to produce high-quality, isotropic meshes directly from 3D point clouds. Our experiments further validate the resilience of this approach, even in the presence of defects such as noise, outliers, and missing data, showing the versatility of the proposed reconstruction method.

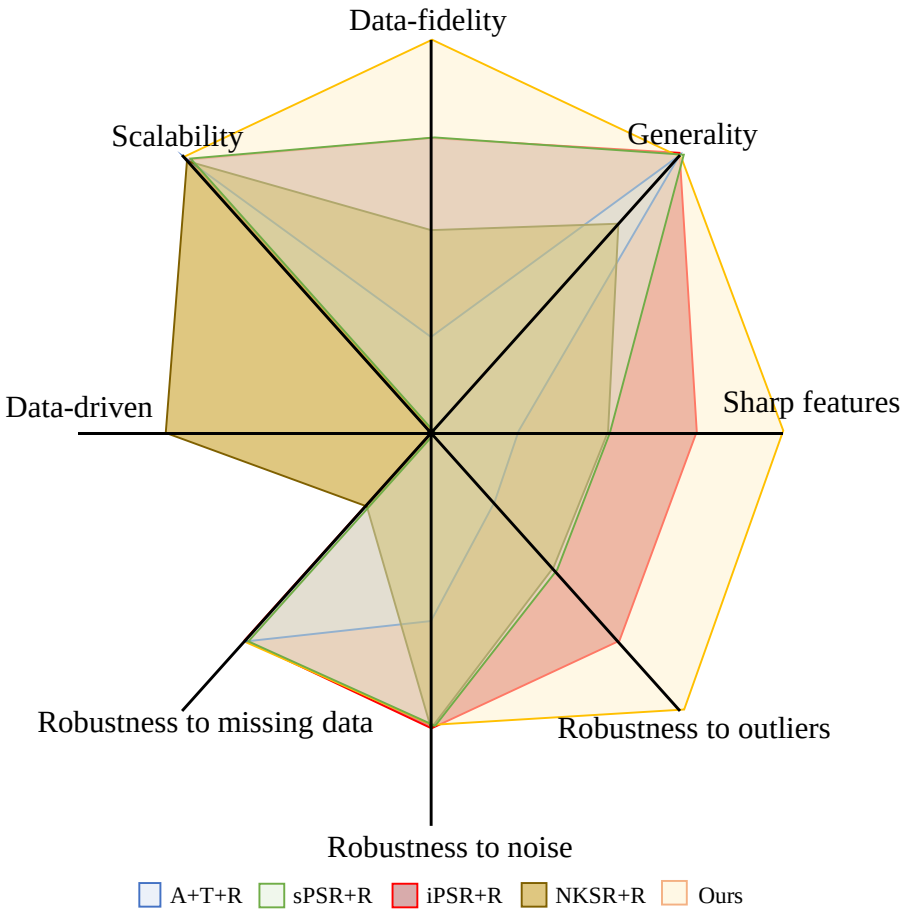


Figure 5.2: Qualitative comparison of the proposed LFS-aware surface reconstruction method with chosen baseline methods.

5.2 Perspectives

This thesis presents two major contributions in the field of shape reconstruction from raw point clouds: Bézier primitive segmentation and LFS-aware surface reconstruction. However, it is important to acknowledge that several challenges and opportunities for further improvements remain. In this section, we discuss these areas for future research and development, providing insights into how the field can continue to evolve and address its most pressing issues.

We will start by listing potential directions for improving the Bézier primitive segmentation. These directions encompass various aspects of our current methodology, aiming to refine and extend its capabilities.

1. **Linear Bézier patch integration:** The Bézier primitive segmentation we employ in our research is rooted in rational Bézier patches. However, due to the inherent non-linearity of this formulation, the process of fitting the parametric patch can be quite intricate. One potential avenue for improvement involves substituting the rational Bézier patch with a linear one, which could potentially enhance the segmentation results.

2. **Over-segmentation mitigation:** The Bézier decomposition can sometimes lead to over-segmentation by nature. To address this, another avenue for improvement is to explore techniques to mitigate over-segmentation issues that can arise during Bézier primitive segmentation, ensuring more accurate and coherent segmentation results.

3. **Scalability enhancement:** One of the limitations we encountered is related to the scalability of our BPNet due to the fixed input size requirement imposed by neural networks. To address this limitation, we had to implement downsampling techniques for real-scan data. However, further research could delve into more efficient and scalable approaches that eliminate the need for such downsampling, allowing our method to handle point clouds of varying sizes seamlessly. This would enhance the versatility and practicality of our approach, making it more suitable for a wider range of applications and scenarios.

4. **Direct Bézier model output:** Even though we provide post-processing steps to construct the full Bézier model, this step can be rather non-trivial. Therefore, a promising direction for future research is to investigate methods that enable the neural network to directly output the full Bézier model without

the need for extensive post-processing. This advancement would not only simplify the workflow but also enhance the efficiency and accuracy of our approach, potentially opening up new possibilities for shape reconstruction from 3D point clouds.

5. **Adjacent patch alignment:** We did not consider the alignment of adjacent patches during Bézier primitive segmentation. One future work is to ensure the continuity and coherence of segmented primitives, especially in scenarios involving shared boundaries. This refinement would contribute to a smoother and more accurate representation of complex surfaces with seamless transitions between primitives.

6. **Boundary inference for each patch:** Currently, we generate boundaries by expanding the Bézier patch and subsequently trimming it. A promising direction for future work is to infer the boundaries directly for each patch. This approach could improve the reconstruction quality of the full Bézier model and is particularly significant in applications like reverse engineering, where accurate boundaries are crucial for faithful reproductions.

In the realm of LFS-aware surface reconstruction, our current work leaves spaces for several possibilities for future research. These directions include:

1. **Improved LFS estimation:** Our current point-based LFS estimation exhibits resilience to minor noise levels. However, future research could investigate methods to enhance its robustness in scenarios with more pronounced noise, potentially through advanced filtering techniques, statistical models. This would further broaden its applicability in challenging real-world conditions.

2. **Adaptive Multi-domain discretization:** While our current multi-domain discretization is reach-aware, ensuring detailed representations in regions with significant geometric variations, it may consume excessive memory in flat areas where fine-grained discretization is unnecessary. Investigating adaptive multi-domain discretization approaches is a potential avenue for future work. These approaches could dynamically adjust the levels of details based on the local geometric complexity, efficiently allocating memory resources and maintaining accuracy while reducing computational overhead, particularly in flat or less intricate regions. This adaptation would enhance the memory efficiency and scalability of our surface reconstruction method.

3. **Implicit preservation of sharp features:** Currently, our surface reconstruction method addresses sharp features by explicitly detecting sharp feature curves from point clouds and incorporating them into the reconstruction process. This sharp feature detection relies on the NerVE algorithm, which is a learning-based approach trained on the ABC dataset. This reliance on specific training data can limit the applicability of our method to CAD models. Moreover, an ideal approach would be to preserve sharp features implicitly during the reconstruction process, eliminating the need for explicit detection.

4. **Preservation of boundaries:** An important aspect of surface reconstruction is the preservation of boundaries, especially in scenarios where objects have well-defined edges or boundaries that must be accurately represented. Future work can focus on enhancing our surface reconstruction method to ensure the faithful preservation of boundaries. This may involve the development of specialized algorithms or techniques that identify and reconstruct boundaries with high precision, resulting in more accurate and visually pleasing representations of objects, particularly those with distinct shapes or profiles.

5. **Sensor data integration:** Sensor data integration plays a vital role in ensuring the accuracy and completeness of the reconstructed models. Future research can explore methods to integrate data from multiple sensors or scanning devices. This includes addressing challenges related to data fusion, calibration, and synchronization to create a unified and comprehensive point cloud dataset. By effectively harnessing data from diverse sensors such as LiDAR, photogrammetry, and depth cameras, we can further improve the quality and fidelity of reconstructed surfaces, especially in complex and large-scale environments or objects.

6. **Interactive reconstruction:** Interactive reconstruction tools and methodologies can significantly enhance the usability of surface reconstruction techniques. Future work could focus on developing intuitive and user-friendly interfaces that allow users to actively participate in the reconstruction process. This might involve real-time feedback during point cloud processing, interactive editing of reconstructed surfaces, or the integration of virtual or augmented reality environments for immersive reconstruction experiences.

APPENDIX A

GUI Demo

We present a graphical user interface (GUI) demonstration of our LFS-aware surface reconstruction method. This interactive demo provides an accessible platform for users to explore and experience the capabilities of our innovative reconstruction technique in a user-friendly manner. Furthermore, it offers the ability to visualize each individual component, enhancing the user's understanding of the underlying processes.

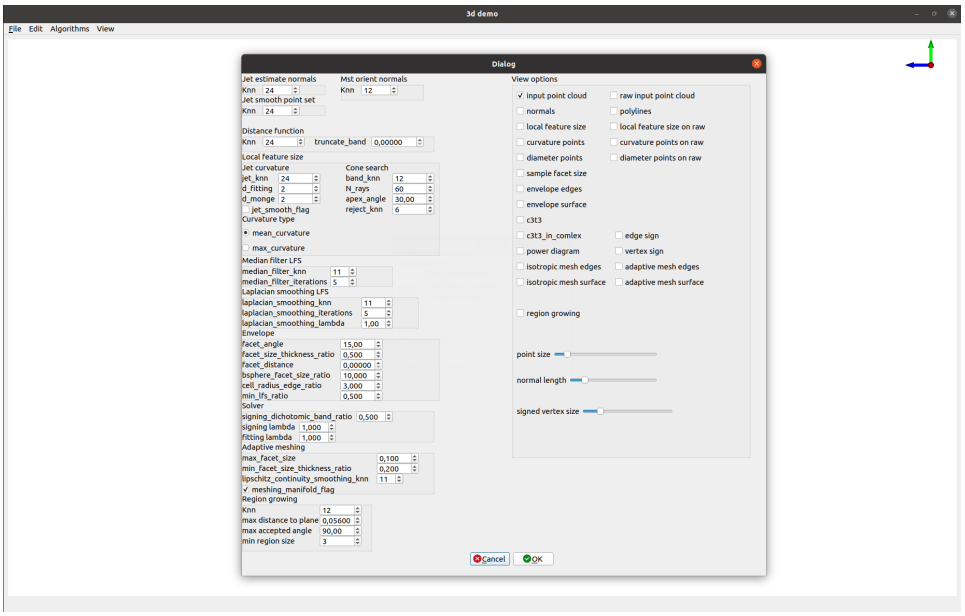


Figure A.1: Demo with the Menu.

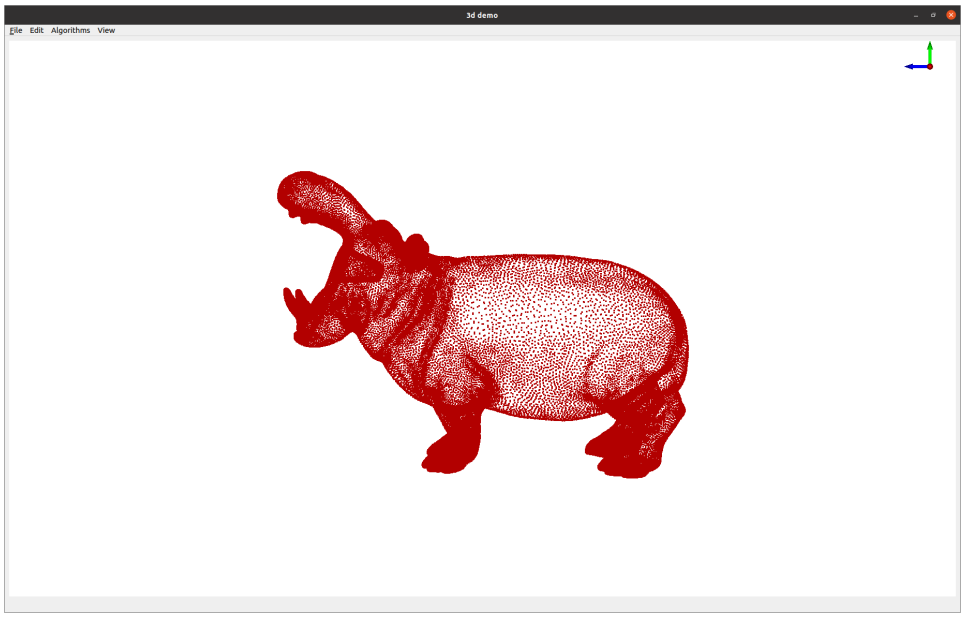


Figure A.2: Input point cloud.

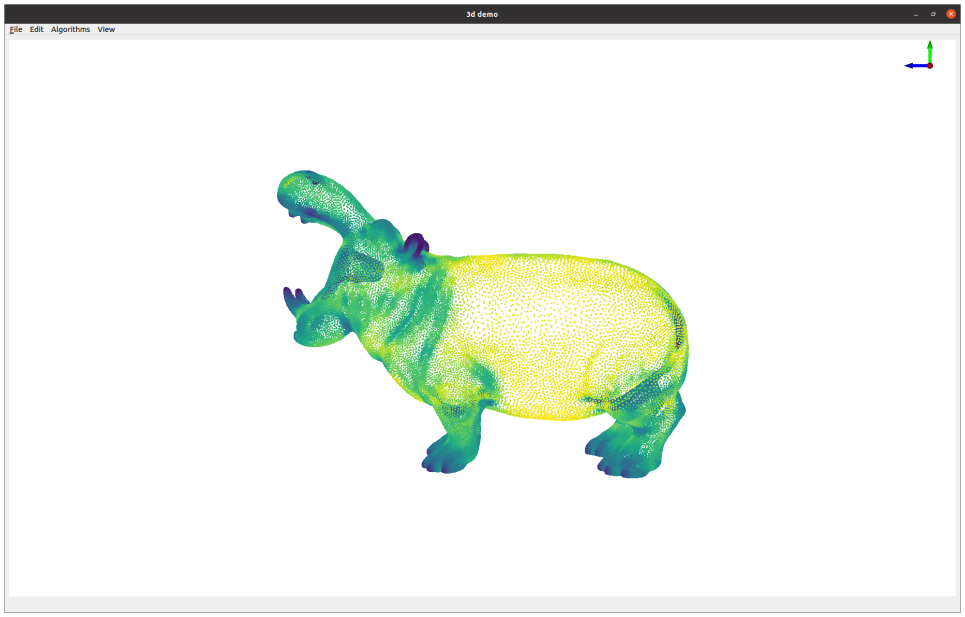


Figure A.3: LFS estimation.

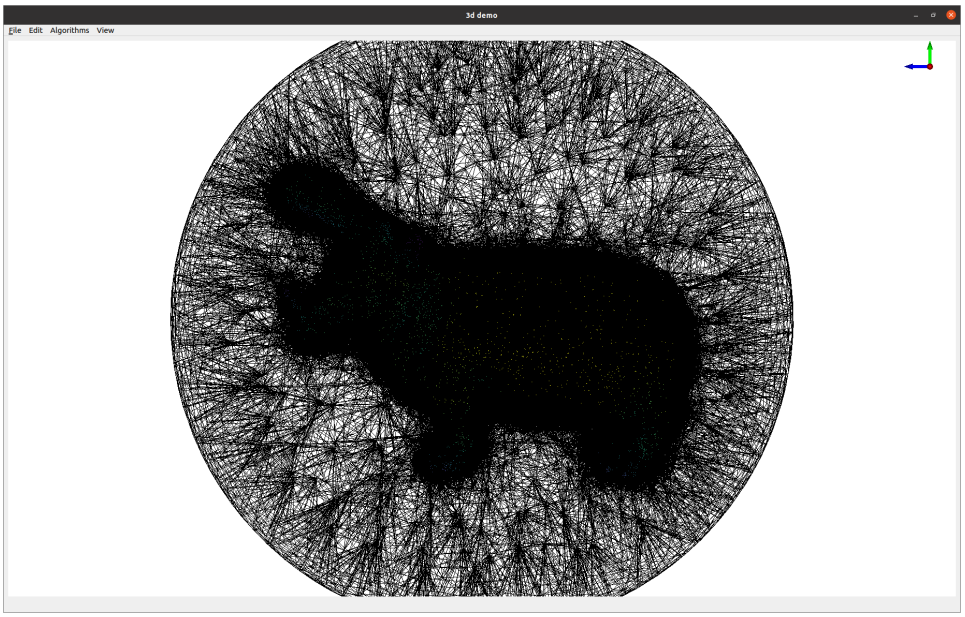


Figure A.4: Multi-domain discretization.

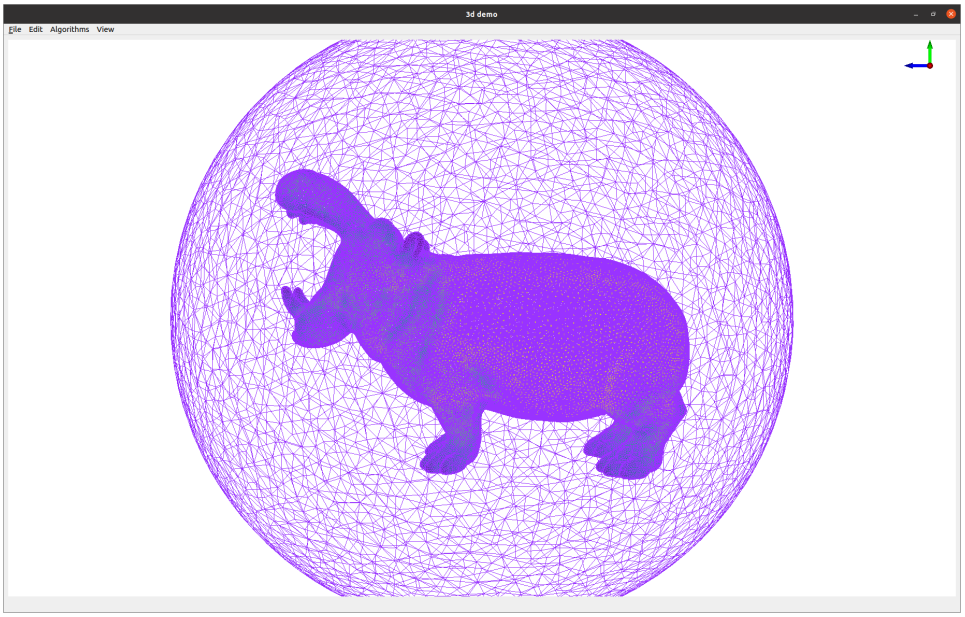


Figure A.5: Boundary of multi-domain discretization.

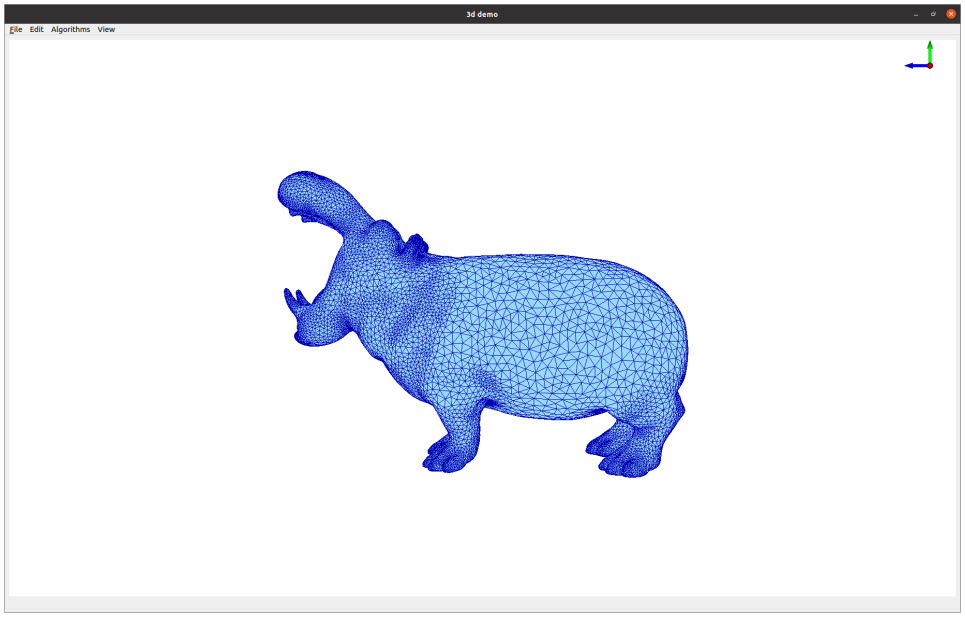


Figure A.6: LFS-aware mesh.

Bibliography

[AB98] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 39–48, 1998. (Cited on pages 23, 24 and 63.)

[ACDL00] Nina Amenta, Sunghee Choi, Tamal K Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 213–222, 2000. (Cited on page 25.)

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2-3):127–153, 2001. (Cited on pages 23, 24 and 25.)

[ACSTD07] Pierre Alliez, David Cohen-Steiner, Yiying Tong, and Mathieu Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Symposium on Geometry processing*, volume 7, pages 39–48, 2007. (Cited on pages 27 and 28.)

[AKC+19] Eddie Aamari, Jisu Kim, Frédéric Chazal, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman. Estimating the reach of a manifold. *Electronic journal of statistics*, 13(1):1359–1399, 2019. (Cited on page 23.)

[AUGA08] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. *Shape analysis and structuring*, pages 53–82, 2008. (Cited on pages 32 and 33.)

[BC01] Jean-Daniel Boissonnat and Frédéric Cazals. Natural neighbor coordinates of points on a surface. *Computational Geometry*, 19(2-3):155–173, 2001. (Cited on page 23.)

[BKP+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010. (Cited on page 3.)

[BL20] Jean-Philippe Bauchet and Florent Lafarge. Kinetic shape reconstruction. *ACM Transactions on Graphics (TOG)*, 39(5):1–14, 2020. (Cited on page 29.)

[BM22] Alexandre Boulch and Renaud Marlet. Poco: Point convolution for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6302–6314, 2022. (Cited on page 30.)

[BPG+20] Jan Bednarik, Shaifali Parashar, Erhan Gundogdu, Mathieu Salzmann, and Pascal Fua. Shape reconstruction by learning differentiable surface representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4716–4725, 2020. (Cited on page 22.)

[BSEH11] Michael J Borden, Michael A Scott, John A Evans, and Thomas JR Hughes. Isogeometric finite element data structures based on bézier extraction of nurbs. *International Journal for Numerical Methods in Engineering*, 87(1-5):15–47, 2011. (Cited on page 36.)

[BTS+17] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1):301–329, 2017. (Cited on pages 8 and 24.)

[CGA09] CGAL. The computational geometry algorithms library. `https://www.cgal.org/`, 2009. Accessed: 2023-05-15. (Cited on pages 3, 43, 56 and 75.)

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. *ACM Transactions on Graphics (TOG)*, 28(3):1–12, 2009. (Cited on page 46.)

[CP05] Frédéric Cazals and Marc Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121–146, 2005. (Cited on pages 24 and 69.)

[CSD04] David Cohen-Steiner and Frank Da. A greedy delaunay-based surface reconstruction algorithm. *The Visual Computer*, 20(1):4–16, 2004. (Cited on pages 26 and 55.)

[CTFZ22] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022. (Cited on pages 31 and 32.)

[CZ21] Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021. (Cited on pages 31 and 32.)

[DBSF20] Zhantao Deng, Jan Bednařík, Mathieu Salzmann, and Pascal Fua. Better patch stitching for parametric surface reconstruction. In *2020 International Conference on 3D Vision (3DV)*, pages 593–602. IEEE, 2020. (Cited on page 22.)

[DG03] Tamal K Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134, 2003. (Cited on page 26.)

[DHS16] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3150–3158, 2016. (Cited on pages 38 and 57.)

[DK91] Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*, 74(1):214–224, 1991. (Cited on pages 31, 63 and 75.)

[DS05] Tamal K Dey and Jian Sun. An adaptive mls surface for reconstruction with guarantees. In *Symposium on Geometry processing*, pages 43–52, 2005. (Cited on pages 31 and 98.)

[DS06] Tamal K Dey and Jian Sun. Normal and feature approximations from noisy point clouds. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science: 26th International Conference, Kolkata, India, December 13-15, 2006. Proceedings 26*, pages 21–32. Springer, 2006. (Cited on pages 23, 66 and 76.)

[EBF+20] Francis Engelmann, Martin Bokeloh, Alireza Fathi, Bastian Leibe, and Matthias Nießner. 3d-mpa: Multi-proposal aggregation for 3d semantic instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9031–9040, 2020. (Cited on page 21.)

[EGO+20] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J Mitra, and Michael Wimmer. Points2surf learning implicit surfaces from point clouds. In *European Conference on Computer Vision*, pages 108–124. Springer, 2020. (Cited on page 30.)

[Fal04] Bianca Falcidieno. Aim@shape project presentation. In *Proceedings of Shape Modeling International (SMI)*, pages 329–329. IEEE Computer Society, 2004. (Cited on page 53.)

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. (Cited on page 18.)

[FSG17] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613, 2017. (Cited on page 22.)

[GCSA13] Simon Giraudot, David Cohen-Steiner, and Pierre Alliez. Noise-adaptive shape reconstruction from raw point sets. In *Computer Graphics Forum*, volume 32, pages 229–238. Wiley Online Library, 2013. (Cited on pages 27 and 72.)

[GFK+18] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (TPAMI)*, pages 216–224, 2018. (Cited on page 21.)

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. (Cited on page 76.)

[GWH+20] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020. (Cited on page 21.)

[HCH22] Jiahui Huang, Hao-Xiang Chen, and Shi-Min Hu. A neural galerkin solver for accurate surface reconstruction. *ACM Transactions on Graphics (TOG)*, 41(6):1–16, 2022. (Cited on page 30.)

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992. (Cited on page 27.)

[HGA+23] Jiahui Huang, Zan Gojcic, Matan Atzmon, Or Litany, Sanja Fidler, and Francis Williams. Neural kernel surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4369–4379, 2023. (Cited on pages 30, 95 and 98.)

[HNAS11] Mark N Helfrick, Christopher Niezrecki, Peter Avitabile, and Timothy Schmidt. 3d digital image correlation methods for full-field vibration measurement. *Mechanical systems and signal processing*, 25(3):917–927, 2011. (Cited on page 5.)

[HT20] Joel Hass and Maria Trnkova. Approximating isosurfaces by guaranteed-quality triangular meshes. In *Computer Graphics Forum*, volume 39, pages 29–40. Wiley Online Library, 2020. (Cited on page 31.)

[HWW+22a] Fei Hou, Chiyu Wang, Wencheng Wang, Hong Qin, Chen Qian, and Ying He. Iterative poisson surface reconstruction (ipsr) for unoriented points. *arXiv preprint arXiv:2209.09510*, 2022. (Cited on pages 27, 29, 95 and 98.)

[HWW+22b] Zhangjin Huang, Yuxin Wen, Zihao Wang, Jinjuan Ren, and Kui Jia. Surface reconstruction from point clouds: A survey and a benchmark. *arXiv preprint arXiv:2205.02413*, 2022. (Cited on page 24.)

[HZS21] Jingwei Huang, Yanfeng Zhang, and Mingwei Sun. Primitivenet: Primitive instance segmentation with local primitive embedding under adversarial metric. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15343–15353, 2021. (Cited on page 21.)

[JAYB15] Clément Jamin, Pierre Alliez, Mariette Yvinec, and Jean-Daniel Boissonnat. Cgalmesh: a generic framework for delaunay mesh generation. *ACM Transactions on Mathematical Software (TOMS)*, 41(4):1–24, 2015. (Cited on pages 63, 72, 75, 87, 95 and 102.)

[JLSW02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002. (Cited on page 31.)

[JZS+20] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4867–4876, 2020. (Cited on page 21.)

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, page 0, 2006. (Cited on pages 27 and 28.)

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. (Cited on pages 27 and 98.)

[KK13] Philipp Krähenbühl and Vladlen Koltun. Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning (ICML)*, pages 513–521. PMLR, 2013. (Cited on page 39.)

[KMJ+19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. (Cited on page 43.)

[Kol08] Ravikrishna Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms (TALG)*, 4(2):1–25, 2008. (Cited on pages 27, 28 and 72.)

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. (Cited on page 39.)

[KYZB19] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A survey of simple geometric primitives detection methods for captured 3d data. *Computer Graphics Forum (CGF)*, 38(1):167–196, 2019. (Cited on pages 17 and 18.)

[LA13] Florent Lafarge and Pierre Alliez. Surface reconstruction through point set structuring. In *Computer Graphics Forum*, volume 32, pages 225–234. Wiley Online Library, 2013. (Cited on page 29.)

[LBKP21] Yonghyeon Lee, Jonghyuk Baek, Young Min Kim, and Frank Chongwoo Park. Imat: The iterative medial axis transform. In *Computer Graphics Forum*, volume 40, pages 162–181. Wiley Online Library, 2021. (Cited on page 23.)

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. (Cited on pages 31 and 63.)

[LDG18] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. (Cited on page 31.)

[LF15] Bruno Lévy and Alain Filbois. Geogram: a library for geometric algorithms, 2015. (Cited on page 98.)

[LGG+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. (Cited on page 39.)

[LPRM02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics (TOG)*, 21(3):362–371, 2002. (Cited on page 55.)

[LSD+19] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2652–2660, 2019. (Cited on pages 19, 39 and 45.)

[LXSW22] Siyou Lin, Dong Xiao, Zuoqiang Shi, and Bin Wang. Surface reconstruction from point clouds without normals by parametrizing the gauss formula. *ACM Transactions on Graphics*, 42(2):1–19, 2022. (Cited on page 27.)

[MDGD+10] Patrick Mullen, Fernando De Goes, Mathieu Desbrun, David Cohen-Steiner, and Pierre Alliez. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Computer Graphics Forum*, 29(5):1733–1741, 2010. (Cited on pages 27, 70 and 72.)

[Mic23] Microsoft. Azure kinect dk depth camera. https://learn.microsoft.com/en-us/azure/kinect-dk/depth-camera, 2023. (Cited on page 6.)

[MLM01] David Marshall, Gabor Lukacs, and Ralph Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, 23(3):304–314, 2001. (Cited on page 18.)

[MON+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. (Cited on page 30.)

[MP18] Nicolas Moyroud and Frédéric Portet. Introduction to qgis. *QGIS and generic tools*, 1:1–17, 2018. (Cited on page 2.)

[Nie04] Gregory M Nielson. Dual marching cubes. In *IEEE visualization 2004*, pages 489–496. IEEE, 2004. (Cited on page 31.)

[NLG15] Vincent Nivoliers, Bruno Lévy, and Christophe Geuzaine. Anisotropic and feature sensitive triangular remeshing using normal lifting. *Journal of Computational and Applied Mathematics*, 289:225–240, 2015. (Cited on page 33.)

[NW17] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2353–2361, 2017. (Cited on page 29.)

[Ope18] OpenCascade. Open cascade technology occt. https://www.opencascade.com/, 2018. Accessed: 2023-05-15. (Cited on page 43.)

[PFS+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. (Cited on page 11.)

[PJL+21] Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems*, 34:13032–13044, 2021. (Cited on page 30.)

[PM15] Jiju Peethambaran and Ramanathan Muthuganapathy. Reconstruction of water-tight surfaces through delaunay sculpting. *Computer-Aided Design*, 58:62–72, 2015. (Cited on page 27.)

[PNM+20] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020. (Cited on pages 10 and 30.)

[PT96] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996. (Cited on pages 36 and 55.)

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. (Cited on pages 19 and 96.)

[RAV+19] Daniel Rebain, Baptiste Angles, Julien Valentin, Nicholas Vining, Jiju Peethambaran, Shahram Izadi, and Andrea Tagliasacchi. Lsmat

least squares medial axis transform. In *Computer Graphics Forum*, volume 38, pages 5–18. Wiley Online Library, 2019. (Cited on page 23.)

[RDvdHV07] Tahir Rabbani, Sander Dijkman, Frank van den Heuvel, and George Vosselman. An integrated approach for modelling and global registration of point clouds. *ISPRS journal of Photogrammetry and Remote Sensing*, 61(6):355–370, 2007. (Cited on page 18.)

[RMvH16] Juergen Riegel, Werner Mayer, and Yorik van Havre. Freecad. *Freecadspec2002. pdf*, 2016. (Cited on page 2.)

[RNDA13] Xavier Rolland-Neviere, Gwenaël Doërr, and Pierre Alliez. Robust diameter-based thickness estimation of 3d objects. *Graphical Models*, 75(6):279–296, 2013. (Cited on pages 23, 68 and 69.)

[SDR+22] Gopal Sharma, Bidya Dash, Aruni RoyChowdhury, Matheus Gadelha, Marios Loizou, Liangliang Cao, Rui Wang, EG Learned-Miller, Subhransu Maji, and Evangelos Kalogerakis. Prifit: learning to fit primitives improves few shot point cloud segmentation. In *Computer Graphics Forum*, volume 41, pages 39–50. Wiley Online Library, 2022. (Cited on page 19.)

[SG03] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 20–30, 2003. (Cited on page 33.)

[SLC+19] Kehua Su, Na Lei, Wei Chen, Li Cui, Hang Si, Shikui Chen, and Xianfeng Gu. Curvature adaptive surface remeshing by sampling normal cycle. *Computer-Aided Design*, 111:1–12, 2019. (Cited on page 33.)

[SLM+20] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision (ECCV)*, pages 261–276. Springer, 2020. (Cited on pages 19, 20 and 45.)

[SLS+07] Andrei Sharf, Thomas Lewiner, Gil Shklarski, Sivan Toledo, and Daniel Cohen-Or. Interactive topology-aware surface reconstruction.

ACM Transactions on Graphics (TOG), 26(3):43–es, 2007. (Cited on pages 9 and 11.)

[SSCO08] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259, 2008. (Cited on pages 23, 24, 68 and 69.)

[SW04] Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, pages 70–76. IEEE, 2004. (Cited on page 31.)

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum (CGF)*, 26(2):214–226, 2007. (Cited on page 18.)

[TDS+16] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*, volume 35, pages 573–597. Wiley Online Library, 2016. (Cited on page 23.)

[TSG+17] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2635–2643, 2017. (Cited on page 19.)

[VdJD16] Sam Van der Jeught and Joris JJ Dirckx. Real-time structured light profilometry: a review. *Optics and Lasers in Engineering*, 87:18–31, 2016. (Cited on page 4.)

[WSL+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019. (Cited on pages 38 and 57.)

[Wu11] Changchang Wu. Visualsfm: A visual structure from motion system. http://www. cs. washington. edu/homes/ccwu/vsfm, 2011. (Cited on page 6.)

[WWX+22] Pengfei Wang, Zixiong Wang, Shiqing Xin, Xifeng Gao, Wenping Wang, and Changhe Tu. Restricted delaunay triangulation for explicit surface reconstruction. *ACM Transactions on Graphics*, 41(5):1–20, 2022. (Cited on page 27.)

[WYHN18] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2569–2578, 2018. (Cited on page 21.)

[WYLU22] Jiehui Wang, Tianqi Yi, Xiao Liang, and Tamon Ueda. Application of 3d laser scanning technology using laser radar system to error analysis in the curtain wall construction. *Remote Sensing*, 15(1):64, 2022. (Cited on page 4.)

[XWD+22] Rui Xu, Zixiong Wang, Zhiyang Dou, Chen Zong, Shiqing Xin, Mingyan Jiang, Tao Ju, and Changhe Tu. Rfeps: Reconstructing feature-line equipped polygonal surface. *ACM Transactions on Graphics (TOG)*, 41(6):1–15, 2022. (Cited on pages 96, 97 and 98.)

[YFST18] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 206–215, 2018. (Cited on page 22.)

[YHL+18] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018. (Cited on pages 39 and 46.)

[YWC+19] Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. Learning object bounding boxes for 3d instance segmentation on point clouds. *Advances in neural information processing systems (NIPS)*, 32, 2019. (Cited on page 21.)

[YY21] Siming Yan and Zhenpei Yang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF Inter-*

national Conference on Computer Vision (ICCV), 2021. (Cited on pages 19, 20 and 45.)

[YZW+19] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3947–3956, 2019. (Cited on page 21.)

[ZAB+21] Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. Progressive discrete domains for implicit surface reconstruction. In *Computer Graphics Forum*, volume 40, pages 143–156. Wiley Online Library, 2021. (Cited on page 31.)

[ZDC+23] Xiangyu Zhu, Dong Du, Weikai Chen, Zhiyou Zhao, Yinyu Nie, and Xiaoguang Han. Nerve: Neural volumetric edges for parametric curve extraction from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13601–13610, 2023. (Cited on pages 75, 95 and 96.)

[ZSC+14] Yanshu Zhu, Feng Sun, Yi-King Choi, Bert Jüttler, and Wenping Wang. Computing a compact spline representation of the medial axis transform of a 2d shape. *Graphical Models*, 76(5):252–262, 2014. (Cited on page 23.)

[ZYY+17] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 900–909, 2017. (Cited on page 19.)

[ZZW+23] Chen Zong, Jinhui Zhao, Pengfei Wang, Shuangmin Chen, Shiqing Xin, Yuanfeng Zhou, Changhe Tu, and Wenping Wang. A region-growing gradnormal algorithm for geometrically and topologically accurate mesh extraction. *Computer-Aided Design*, page 103559, 2023. (Cited on page 31.)