Pontifícia Universidade Católica
do Rio de Janeiro

Jan Jose Hurtado Jauregui

Denoising and simplification in the construction of 3D digital models of complex objects

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática.

Advisor: Prof. Marcelo Gattass

Rio de Janeiro
December 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Jan Jose Hurtado Jauregui

Denoising and simplification in the construction of 3D digital models of complex objects

Thesis presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática. Approved by the Examination Committee:

Prof. Marcelo Gattass
Advisor
Departamento de Informática – PUC-Rio

Prof. Waldemar Celes Filho
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. Luiz Henrique de Figueiredo
IMPA

Prof. Anselmo Antunes Montenegro
UFF

Rio de Janeiro, December 15th, 2021

Jan Jose Hurtado Jauregui

Graduated in computer science at the National University of Saint Augustine (Arequipa, Peru) in 2016 and obtained his M.Sc. degree in computer science at the Pontifical Catholic University of Rio de Janeiro (Rio de Janeiro, Brazil) in 2018. Visiting researcher at the Tel Aviv University (Tel Aviv, Israel) in 2015. Researcher and developer at the Tecgraf Institute (PUC-Rio). His research interests are geometry processing and analysis, image processing and analysis, computer graphics, and deep learning.

Bibliographic data

Hurtado Jauregui, Jan Jose

Denoising and simplification in the construction of 3D digital models of complex objects / Jan Jose Hurtado Jauregui; advisor: Marcelo Gattass. – 2021.

114 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2021.

Inclui bibliografia

1. Informática – Teses. 2. Remoção de ruído. 3. Estimação de normais. 4. Detecção de feições nítidas. 5. Modelo de malha CAD. 6. Simplificação. I. Gattass, Marcelo. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to thank my advisor, Prof. Marcelo Gattass, for his patience, motivation and guidance during the Master and PhD studies.

I thank professors Anselmo Montenegro and Alberto Raposo for their comments and suggestions considered for different parts of this work. I also thank professors Waldemar Celes and Luiz Henrique de Figueiredo, for being part of this research through the proposal's feedback.

I thank professors Cristian Lopez del Alamo, Ernesto Cuadros, Wilber Ramos, and Alex Bronstein for being part of my research career.

I would also like to give special thanks to my family and my beautiful wife, Fabíola Soares, for their continuous support and comprehension. Without you, this would not have been possible.

Finally, I would like to thank God for letting me through all the difficulties.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the Tecgraf Institute (PUC-Rio) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Abstract

Hurtado Jauregui, Jan Jose; Gattass, Marcelo (Advisor). **Denoising and simplification in the construction of 3D digital models of complex objects**. Rio de Janeiro, 2021. 114p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

As the digitalization process advances in several industries, the creation of 3D digital models is becoming more and more required. Commonly, these models are constructed by 3D designers, requiring considerable manual effort when the modeled object is complex. In addition, since the designer does not have an accurate reference in most cases, the resulting model is prone to measurement errors. However, it is possible to minimize the construction effort and the measurement error by using 3D acquisition techniques and previously constructed CAD models. The typical output of a 3D acquisition technique is a raw 3D point cloud, which needs processing to reduce the inherent noise and lack of topological information. CAD models are typically used to document an engineering design process, presenting high complexity and too many details irrelevant to many visualization processes. So, depending on the application, we must severely simplify the CAD model to meet its requirements. In this thesis, we focus on the construction of 3D digital models from these sources. More precisely, we present a set of geometry processing algorithms to automatize different stages of a typical workflow used for this construction. First, we present a point cloud denoising algorithm that seeks to preserve the sharp features of the underlying surface. This algorithm includes solutions for the normal estimation and sharp feature detection problems. Second, we present an extension of the point cloud denoising algorithm to process triangle meshes, where we take advantage of the explicit topology defined by the mesh. Finally, we present an algorithm for the extreme simplification of complex CAD models, which tends to approximate the outer surface of the modeled object. The proposed algorithms are compared with state-of-the-art methods, showing competitive results and outperforming them in most test cases.

Keywords

Denoising; Normal estimation; Sharp feature detection; CAD mesh model; Simplification.

Resumo

Hurtado Jauregui, Jan Jose; Gattass, Marcelo. **Remoção de ruído e simplificação na construção de modelos digitais 3D de objetos complexos**. Rio de Janeiro, 2021. 114p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

À medida que o processo de digitalização avança em diversos setores, a criação de modelos digitais 3D torna-se cada vez mais necessária. Normalmente, esses modelos são construídos por designers 3D, exigindo um esforço manual considerável quando o objeto modelado é complexo. Além disso, como o designer não tem uma referência precisa na maioria dos casos, o modelo resultante está sujeito a erros de medição. No entanto, é possível minimizar o esforço de construção e o erro de medição usando técnicas de aquisição 3D e modelos CAD previamente construídos. A saída típica de uma técnica de aquisição 3D é uma nuvem de pontos 3D bruta, que precisa de processamento para reduzir o ruído inerente e a falta de informações topológicas. Os modelos CAD são normalmente usados para documentar um processo de projeto de engenharia, apresentando alta complexidade e muitos detalhes irrelevantes para muitos processos de visualização. Portanto, dependendo da aplicação, devemos simplificar bastante o modelo CAD para atender aos seus requisitos. Nesta tese, nos concentramos na construção de modelos digitais 3D a partir dessas fontes. Mais precisamente, apresentamos um conjunto de algoritmos de processamento de geometria para automatizar diferentes etapas de um fluxo de trabalho típico usado para esta construção. Primeiro, apresentamos um algoritmo de redução de ruído de nuvem de pontos que visa preservar as feições nítidas da superfície subjacente. Este algoritmo inclui soluções para a estimativa normal e problemas de detecção de feições nítidas. Em segundo lugar, apresentamos uma extensão do algoritmo de redução de ruído de nuvem de pontos para processar malhas triangulares, onde tiramos proveito da topologia explícita definida pela malha. Por fim, apresentamos um algoritmo para a simplificação extrema de modelos CAD complexos, que tendem a se aproximar da superfície externa do objeto modelado. Os algoritmos propostos são comparados com métodos de última geração, apresentando resultados competitivos e superando-os na maioria dos casos de teste.

Palavras-chave

Remoção de ruído; Estimação de normais; Detecção de feições nítidas; Modelo de malha CAD; Simplificação.

Table of contents

List of figures

List of tables

1
General introduction

In the era of industry 4.0, the construction of 3D digital models to represent real environments and objects is essential for several tasks, such as planning, maintenance, training, and education. Commonly, these environments and objects involve multiple complex surfaces that are difficult to model, e.g., industrial facilities and manufactured objects. The modeling task is usually performed by 3D designers, using 3D modeling and texture mapping tools, such as Blender, 3Ds Max, Maya, or ZBrush [1]. The usage of these tools requires some expertise and effort when the modeled structure is complex. Also, since the designer does not have accurate references in most cases, the resulting 3D digital model is prone to measurement error. For this reason, methods capable of automatizing the modeling processes and measurements can be helpful to achieve more efficiency and fidelity to the real environment or object.

3D acquisition techniques [2] have been evolving rapidly in the last years. These techniques can be used on the target scene or object to obtain raw and accurate surface data. This data is typically represented as an unordered set of 3D points, a.k.a. 3D point cloud, which can be equipped with per-point color and normal information. Although the point cloud can be used for real-time visualization [3], it presents poor interaction. Thus, triangle mesh models can be generated from this data [4], which are the most likely representations for interactive applications and also for geometry processing algorithms (e.g., mesh decimation [5]). However, due to the limitations of the acquisition techniques, the obtained point cloud can present undesired noise. Most of the mesh generation algorithms can not deal correctly with noisy data, and some keep the noise in the resulting mesh. For this reason, it is common to introduce point cloud and mesh denoising methods in the processing pipeline.

Currently, in most industries, computer-aided design (CAD) modeling replaces traditional manual drafting. Thus, CAD models document the engineering design process, which involves detailed description and high fidelity to the modeled object. Although the CAD models are digital representations of real objects, their complexity makes them improper for interactive applications in most cases. So, by simplifying CAD models, it is possible to generate digital representations that satisfy interactive applications' requirements.

It is possible to minimize the effort and the measurement error on the construction of 3D digital models by using 3D acquisition techniques and previously constructed CAD models. This thesis presents a set of geometry processing algorithms to automate different stages of a typical workflow used to construct these models from point clouds and CAD models. First, we present a sharp feature-preserving point cloud denoising algorithm that consists of four main steps: anisotropic neighborhoods computation, normal filtering, sharp feature detection, and point updating. This algorithm includes solutions for the normal estimation and sharp feature detection problems on point clouds. Second, we present an extension of this point cloud-based algorithm for the processing of triangle meshes, where we use the mesh topology to guide the updating operations and minimize mesh artifacts. Finally, we present a multi-step algorithm for the extreme simplification of CAD models. This algorithm tends to approximate the outer surface of the modeled object. All of the mentioned algorithms are compared with state-of-the-art algorithms, showing competitive results and outperforming them in most test cases.

1.1 Main contributions

We can summarize the main contributions of this thesis as:

- We introduce a method for the computation of anisotropic neighborhoods on point clouds. This method uses multiple quadratic numerical optimization problems.
- We introduce a method for point cloud normal filtering that includes a novel normal correction operation used to improve the correctness of the normals.
- We introduce a novel method for the detection of sharp feature points on point clouds.
- We integrate all the mentioned methods in a denoising pipeline that focuses on sharp feature preservation.
- We introduce a method for triangle mesh denoising, which extends the point cloud-based method.
- We introduce an extreme simplification method of CAD mesh models that embody the outer shape of the model.

1.2 Outline

The rest of this thesis is structured as follows. Chapter 2 explains the proposed point cloud denoising algorithm, which includes the anisotropic neighborhood computation, the normal filtering, and the sharp feature detection methods. Chapter 3 presents the extension of the point cloud denoising method to process triangle meshes. Chapter 4 presents the proposed CAD model simplification algorithm. Each chapter follows a similar structure, presenting a more detailed introduction, the related work, the proposed method, the experimental results, and the corresponding conclusions. Chapter 5 presents more general conclusions of the thesis.

2
Point cloud denoising

2.1
Introduction

With the rapid growth of 3D acquisition methods, new geometry processing algorithms are necessary. Typically, acquired 3D data present undesired noise, making denoising a fundamental task for further processing. Usually, the raw data generated by the acquisition methods is a 3D point cloud, whose direct processing is more reliable than processing derived representation like a mesh.

Because the point clouds lack connectivity information and the surface details are hard to differentiate from noise, point cloud denoising is challenging. Early attempts try only to smooth the points. However, this is not enough to obtain good quality results for some applications, such as mesh generation or rendering, where feature preservation is essential. Several methods were proposed to address this problem, ranging from classic moving least square-based methods to deep learning-based methods. Most of them rely on a two-step-based scheme, where the first step consists of computing denoised normals, and the second step consists of updating point positions to fit these normals.

We propose a sharp feature-preserving point cloud denoising method that consists of four main steps. In the first step, we compute anisotropic neighborhoods using local quadratic optimization problems. In the second step, we use these anisotropic neighborhoods to filter the normal field, reducing noise. This step includes a normal corrector operation to minimize problems for the following steps. In the third step, using the filtered normals, we detect sharp feature points to treat them differently. We classify all the points into flat, edge, and corner points. Finally, using this classification and the filtered normal field, we update point positions. We compare our method numerically and visually with several state-of-the-art methods, obtaining competitive results and outperforming them in most test cases.

The rest of the chapter is structured as follows. Section 2.2 presents some related work to the denoising problem. Section 2.3 introduces the proposed denoising method. Section 2.4 explains the computation of connectivity information and geometric measurements useful for the method's steps. Sections 2.5, 2.6, 2.7, and 2.8 describe the four main steps of the proposed method. Section 2.9 presents the experiments and results. Finally, in Section 2.10, we give our conclusions and future work.

2.2 Previous Work

The point cloud denoising problem was addressed in different ways. Some approaches are based on the moving least squares (MLS) method, where the idea is to approximate an underlying surface and project the noisy points on it. This scheme was adapted to preserve details and achieve more robustness against noise [6, 7, 8, 9, 10, 11].

A different way to tackle the denoising problem is by using sparse representations adopted by several geometry processing algorithms. These methods assume that most noisy points can be approximated by piecewise smooth surfaces, where the transitions between them are sparse. These transitions represent the feature regions of the underlying surface. The denoising task is formulated as the sparse reconstruction of point normals and/or point positions, using regularization based on the metric ℓ_1 [12, 13], the metric ℓ_0 [14], or low-rank matrix approximations [15, 16].

The non-local self similarity methods proposed for images [17, 18] were extended for the denoising of point clouds. Using different patch representations, such as polynomial surfaces [19], variation of height fields [20], local displacements [21, 22], local probing fields [23], point normals [24], and sampled collaborative points [25], these methods exploit the similarity between non-local surfaces.

The relation between the points can be represented using graphs. To address the denoising problem, some methods use this representation to exploit graph properties and Laplacian regularizers in a local [26, 27, 28] and a non-local [29] fashion. Further, in [30], the authors propose a feature graph learning framework applied to the denoising problem, using point coordinates and normals as relevant features.

Two-step-based methods are widely used for surface denoising, where the first step denoises the normal field and the second step fits the noisy point positions to the denoised normals. These steps are complemented by neighborhood clustering [31, 32] and/or feature detection [33, 34, 35, 36]. In a different way, [37] introduces geometric structures named Wavejets to represent local surfaces. Then, these structures are used to reduce noise and enhance details by updating point positions and normals.

The locally optimal projection (LOP) method consists of sampling and projecting a set of uniform distributed particles on an underlying surface that the noisy point cloud represents [38]. This method was extended to be more robust against irregular distributions [39] and to deal with feature preservation [40, 41, 42, 43].

Recently, geometric deep learning methods became very popular due to their impressive results in computer graphics applications. The point cloud denoising problem was tackled by this kind of methods [44, 45, 46, 47, 48, 49, 50, 51], however, most of them do not deal correctly with feature preservation. [52] introduces a LOP-based network for feature-aware point cloud consolidation, which can process noisy inputs. [53] proposes a network capable of predicting feature points and noise-free normals. These predictions are used to update point positions iteratively until the noise is removed. Similarly, [54] proposes a normal refinement network that processes adaptive geometric descriptors, i.e. local height and normal fields. Then, the refined normals are used to update point positions.

Although the presented methods try to deal with detail preservation, most of them focus on the preservation of smooth features without dealing correctly with sharp features. Sparse-based and two-step-based methods have shown superlative performance on the preservation of sharp features, especially those that include feature detection procedures in their denoising pipeline. The proposed method also focuses on the preservation of sharp features and belongs to the family of two-step-based methods. More precisely, we introduce a new approach for the robust estimation of a clean normal field based on anisotropic neighborhoods and a novel normal correction operation. For the anisotropic neighborhood computation, we extend the idea proposed in [55], introducing a new functional and its discretization to deal with point clouds. Then, using the clean normal field, we introduce a novel sharp feature detection algorithm that is more selective than previous methods. This behavior helps to avoid the excessive agglomeration of points near the feature regions and the generation of gaps. Finally, point positions are updated using the clean normals and the feature detection information, similarly to [35].

2.3 Overview

The input of our denoising algorithm is a 3D point cloud which consists of a set of unorganized points $\mathcal{P} = \{p_i\}_{i=1}^{n}$ sampled from a piecewise 2-manifold $\mathcal{X}$ embedded in $\mathbb{R}^3$, where n is the number of points. This set of points should be equipped with a consistently oriented normal field. Let us denote the point

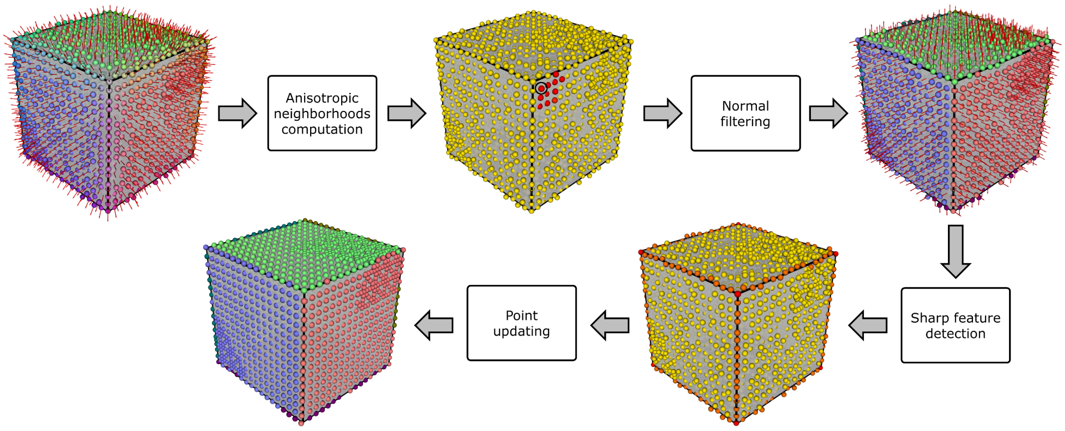


Figure 2.1: Denoising pipeline

coordinates and the corresponding normals as the sets of 3-dimensional vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $N = \{\mathbf{n}_1, \ldots, \mathbf{n}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^3$ and $\mathbf{n}_i \in \mathbb{R}^3$ represent the coordinates and the normal at point p_i, respectively. The output of the algorithm is a noise-free point cloud equipped with a clean normal field.

Since our algorithm requires an initial set of normals, we assume that a common point cloud should be pre-processed by a normal estimation method that consistently preserves the normal orientation. This method needs not be accurate in the computation of the normals; it just needs to keep the consistent orientation. In addition to the normal initialization, we normalize point positions to avoid high variation when tuning the algorithm parameters. We compute a simple average distance considering each point's distances to their corresponding 10 nearest neighbors. Then, the normalization scale factor for the point cloud is defined by the inverse of this average distance.

The algorithm works in an iterative manner, where each iteration consists of four main steps, described in the following. Firstly, as a preparing procedure useful to represent point cloud topology and geometry, we compute point neighborhoods, normals, areas, and distances.

In the first step, we compute anisotropic neighborhoods, which define normal-based piecewise smooth regions assigned for each point. These neighborhoods are obtained by solving local quadratic optimization problems that minimize normal variation and distances to the evaluated point.

In the second step, we compute a new set of normals based on the anisotropic neighborhoods. These normals enhance feature regions because they are fitted to piecewise smooth regions, preserving the hard transitions between them. Since these normals are computed using noisy point positions, they can also present noise. Thus, we use bilateral filtering to smooth them while preserving the enhanced regions. We also introduce a normal correction operation because the anisotropic neighborhood-based normals are prone to incorrect orientation due to the noisy input normals and the parameters used for the optimization problem. This corrector operation is based on evaluating the neighboring piecewise smooth regions for each point to then select the region that better fits it. If the current normal is considerably different from the average normal of the selected region, we assign this average normal as the new normal for the corresponding point. To define the neighboring piecewise smooth regions, we first select candidate feature points that may require this evaluation. We focus on potential feature points because points that belong to smooth regions do not require normal correction. Then, we use a clustering operation that aims to segment the points around the evaluated point based on their normals. Each cluster is considered as a neighboring piecewise smooth region, whose average normal can be used for normal correction. This normal filtering step allows us to define reliable normals and minimize artifacts in the following steps.

In the third step, we apply the same feature candidate selection and neighborhood clustering operations but now using the corrected normals. Then, using this information, we detect sharp feature points, classifying them into flat, edge, or corner. This classification is based on measuring the proximity of each point to the intersection of neighboring planes approximated to the corresponding piecewise smooth regions.

Finally, in the fourth step, we update the point positions to fit them to the filtered normals. The point updating scheme depends on the point class defined in the previous step.

The number of iterations for the full set of steps is defined by the parameter n_{ext} (external). However, because the computation of the anisotropic neighborhoods is a costly procedure, we also consider an iterative scheme within the full iteration that applies the following steps: normal filtering, feature classification, and point updating. The number of these internal iterations is defined by the parameter n_{int}. Algorithm 1 summarizes the proposed pipeline, where the color maps the main steps. Also, Figure 2.1 illustrates the proposed pipeline.

2.4 Preparing procedure

In this section, we describe how we compute connectivity information and geometric measurements that are necessary for the algorithm's main steps. Let us first denote the standard k nearest neighborhood of a point p_i as $\mathscr{N}_k(p_i)$, where p_i is also included. The *rough normal* of a point p_i is computed by using principal component analysis (PCA) on the points included in $\mathscr{N}_k(p_i)$. Because

Algorithm 1 Point cloud denoising

```
procedure DENOISE($\mathcal{P}$)
    for $it_{ext} \leftarrow 1$ to $n_{ext}$ do
        preparingProcedure($\mathcal{P}$,$\epsilon_d$, $k$, $r_s$, $r_r$, $r_b$)
        anisotropicNeighborhoods($\mathcal{P}$,$\alpha$, $\beta$, $\gamma$, $a_0$)
        for $it_{int} \leftarrow 1$ to $n_{int}$ do
            anisotropicNeighborhoodNormals($\mathcal{P}$,$\tau_u$)
            for $it_{ns} \leftarrow 1$ to $n_{ns}$ do
                normalSmoothing($\mathcal{P}$,$\sigma_{ns}$, $\sigma_{nn}$)
            roughFeatureClassification($\mathcal{P}$, $\tau_n$, $\tau_{ta}$)
            neighborhoodClustering($\mathcal{P}$) # without including $p_i$
            for $it_{nc} \leftarrow 1$ to $n_{nc}$ do
                normalCorrection($\mathcal{P}$, $\epsilon_{mn}$)
            roughFeatureClassification($\mathcal{P}$, $\tau_n$, $\tau_{ta}$)
            neighborhoodClustering($\mathcal{P}$) # including $p_i$
            pointConvexityAnalysis($\mathcal{P}$, $\delta_{cc}$, $\epsilon_{cc}$)
            sharpFeatureDetection($\mathcal{P}$, $\theta$)
            for $it_{fp} \leftarrow 1$ to $n_{fp}$ do
                flatPointUpdate($\mathcal{P}$, $\tau_o$, $\sigma_{ps}$, $\sigma_{pn}$, $\upsilon_f$)
            cornerAndEdgePointUpdate($\mathcal{P}$, $\tau_o$, $\upsilon_e$, $\upsilon_c$)
```

these normals have no consistent orientation, they are corrected (switching sign) by checking the orientation of the pre-computed normals.

Then, trying to define an analogous representation to the first ring neighborhood in a mesh-based representation, for each point p_i, we compute a 2D Delaunay triangulation using the $\mathcal{N}_k(p_i)$ points projected on the plane defined by p_i and its corresponding *rough normal*. Because the point cloud can present multiple points very close to each other, when we compute the triangulation, we ignore those points that are at a distance less than $\epsilon_d l_{ro}$ from p_i in the 3D space, where ϵ_d is a tolerance factor and l_{ro} is a *rough average distance* computed considering the distances from each point to their closest 6 points. The latter allows us to obtain a more reliable triangulation for the following operations. For each p_i, the points that correspond to the first ring in the triangulation, including p_i, comprise the neighborhood $\mathcal{N}_1(p_i)$ on the point cloud.

Similarly to the average edge length computed on mesh-based representations, we compute the *average distance* l_μ between points considering the distances from each p_i to $p_j \in \mathcal{N}_1(p_i)$, s.t. $j \neq i$. The area that a point p_i represents is obtained by computing the barycentric cell-based area of p_i on the corresponding triangulation in the 3D space. We denote all the areas as $\mathbf{a} = \{a_1, \ldots, a_n\}^T$, where a_i represents the area at point p_i.

Then, we define three types of neighborhood named small, regular and big, denoted as $\mathcal{N}_s$, $\mathcal{N}_r$ and $\mathcal{N}_b$, and defined as follows:

$$\begin{aligned}
\mathscr{N}_s(p_i) &= \{p_j \in \mathcal{P} | \|\mathbf{x}_j - \mathbf{x}_i\| < r_s l_i \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0\} \cap \mathscr{N}_k(p_i), \\
\mathscr{N}_r(p_i) &= \{p_j \in \mathcal{P} | \|\mathbf{x}_j - \mathbf{x}_i\| < r_r l_\mu \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0\} \cap \mathscr{N}_k(p_i), \\
\mathscr{N}_b(p_i) &= \{p_j \in \mathcal{P} | \|\mathbf{x}_j - \mathbf{x}_i\| < r_b l_\mu \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0\} \cap \mathscr{N}_k(p_i),
\end{aligned} \tag{2-1}$$

where l_i is the maximum distance from point p_i to any $p_j \in \mathscr{N}_1(p_i)$, and r_s, r_r and r_b are parameter ratios used to define the size of the neighborhoods, s.t. $(1 \leq r_s) \wedge (r_r < r_b)$. $\mathscr{N}_s$ maps the immediate interaction between points, $\mathscr{N}_r$ represents a local surface for each point, and $\mathscr{N}_b$ represents a larger local surface whose convexity or concavity is more evident. These neighborhoods are used in different steps of our denoising algorithm.

Finally, we compute the set of *regular normals* using the PCA-based method on the *regular neighborhood* points $\mathscr{N}_r(p_i)$, keeping the orientation consistency. These normals are then used for the computation of the anisotropic neighborhoods.

2.5
Anisotropic neighborhoods computation

We present an extension to point clouds of the mesh-based anisotropic neighborhood computation introduced in [55, 56]. The idea behind the anisotropic neighborhoods is to compute point-wise descriptors, which define the membership of neighboring points to a piecewise smooth surface region that the evaluated points represent. We assume that a smooth region presents low normal variation, such that the difference between two normals of any two points within this region is minimum. Also, the region's shape should be as regular as possible, centered at the evaluated point, represent a considerable amount of area on the underlying surface, and the normals at different points within it should be similar to the normal at the evaluated point.

In a continuous setting, let us consider the 2-manifold $\mathcal{X}$, the evaluated point $x' \in \mathcal{X}$, and the fuzzy membership function $u : \mathcal{X} \to [0, 1]$, describing the anisotropic neighborhood, where 0 means no membership and 1 means full membership. We aim to find an optimal membership function u by solving the following optimization problem:

$$\begin{aligned}
&\min_u \alpha \int_{x_i \in \mathcal{X}} \int_{x_j \in \mathcal{X}} \|n_i - n_j\| u_i u_j da da + \beta \int_{x_i \in \mathcal{X}} \|x' - x_i\| u_i da \\
&+\gamma \int_{x_i \in \mathcal{X}} \|n' - n_i\| u_i da \quad s.t. \quad u \in [0, 1] \quad \wedge \quad \int_{x_i \in \mathcal{X}} u da = a_0,
\end{aligned} \tag{2-2}$$

where n' is the normal of x', the first term penalizes the normal variation, the second term penalizes the distance to x', the third term penalizes the normal difference regarding n', the upper and lower bound constraints keep the values of u between 0 and 1, the linear constraint helps to avoid 0 area solutions by defining an area a_0 to be covered by u, and the parameters α, β and γ control the behavior of the solution. Differently from [55], we do not include a term to control the regularity of u, i.e. gradient norm penalization, because the point cloud connectivity is not as well defined as the mesh connectivity. We indirectly control the regularity of u by tuning β.

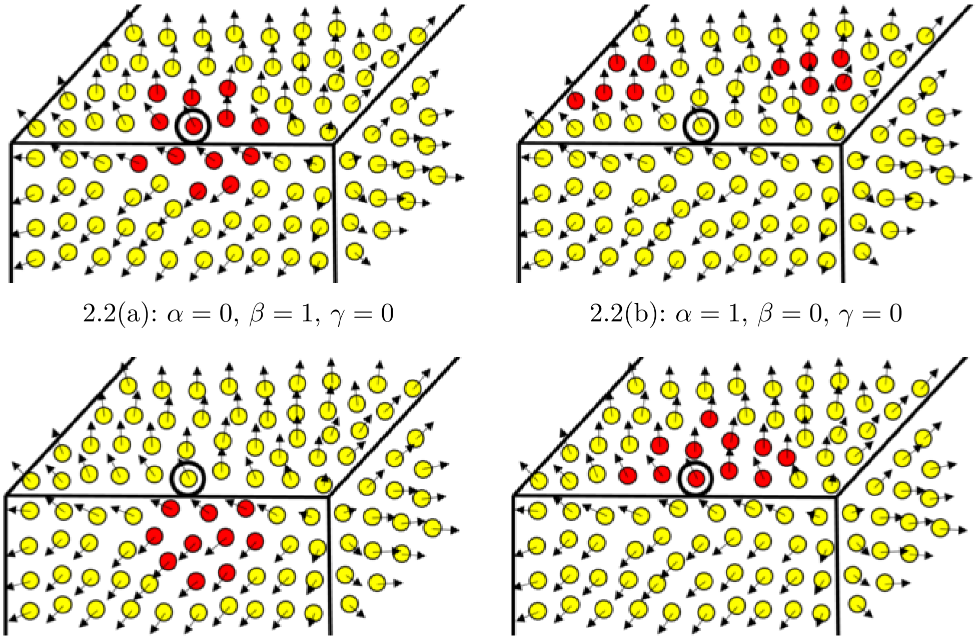
In a discrete setting, considering a point cloud $\mathcal{P}$ as a sample of $\mathcal{X}$, we can represent the coordinates of the evaluated point x' as $\mathbf{x}'$, n' as $\mathbf{n}'$, the membership function u as the vector $\mathbf{u} = \{u_1, \ldots, u_n\}^T$, the distances between all the points and $\mathbf{x}'$ as the vector $\mathbf{d} = \{d_1, \ldots, d_n\}^T$, where $d_i = \|\mathbf{x}_i - \mathbf{x}'\|$, the distances between all the point normals with $\mathbf{n}'$ as the vector $\mathbf{f} = \{f_1, \ldots, f_n\}^T$, where $f_i = \|\mathbf{n}_i - \mathbf{n}'\|$, and the area for each point as the vector $\mathbf{a}$, described previously. Then, the optimization problem can be described as follows:

$$\begin{aligned} \min_{\mathbf{u}} \alpha \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} + \beta \mathbf{d}^T a' \mathbf{A} \mathbf{u} + \gamma \mathbf{f}^T a' \mathbf{A} \mathbf{u} \\ s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0, \end{aligned} \tag{2-3}$$

where $\mathbf{Q}$ is a square matrix whose entries are defined by $q_{ij} = \|\mathbf{n}_i - \mathbf{n}_j\|$, $\mathbf{A}$ is a diagonal matrix containing the point areas as diagonal elements, i.e. $a_{ii} = a_i$, $\mathbf{0}$ is a vector of zeros, $\mathbf{1}$ is a vector of ones, and a' is the area of the evaluated point, used to alleviate the difference between the linear and the quadratic terms. For each point, we compute an optimal solution $\mathbf{u}$ that allows us to describe the anisotropic neighborhoods. For practical purposes, we constraint the domain of u to the *regular neighborhood* $\mathcal{N}_r$ of the evaluated point and define a_0 as a proportion of the total area represented by this neighborhood.

Figure 2.2 illustrates the behavior and the importance of each term in the proposed optimization problem. If we only consider the distance to the evaluated point, we obtain a solution similar to a regular geometric neighborhood, as shown in Figure 2.2(a). By just considering the normal difference within the neighborhood, we can obtain solutions that are not close enough to the evaluated point or irregular and sparse solutions, as shown in Figure 2.2(b). If we consider the distance to the evaluated point, we can control regularity and closeness regarding the evaluated point, as shown in Figures 2.2(c) and 2.2(d). However, Figure 2.2(c) presents a solution that lies on the incorrect face of the shape. Using the penalization of the normal difference regarding the evaluated point normal can be helpful to compute the desired neighborhood, as shown in Figure 2.2(d).

Although all the terms are important to obtain the desired solution, in our experiments, we give more importance to the normal difference within the neighborhood. Figure 2.3 shows an example of the computation of these



2.2(a): $\alpha = 0$, $\beta = 1$, $\gamma = 0$

2.2(b): $\alpha = 1$, $\beta = 0$, $\gamma = 0$

2.2(c): $\alpha = 1$, $\beta = 1$, $\gamma = 0$

2.2(d): $\alpha = 1$, $\beta = 1$, $\gamma = 1$

Figure 2.2: Anisotropic neighborhood computation behavior. For all cases, the evaluated point is marked by a black circle, the black arrows represent the point normals, and the point color defines if it corresponds to the neighborhood (red) or if not (yellow).

anisotropic neighborhoods on a noisy point cloud of a cube. We can see that the selected points present challenging situations. The first case shows a flat point close to a corner, where the computed neighborhood represents the correct face of the cube. The second case shows an edge point, where the computed neighborhood is defined on just one of the shared faces of the cube. The third case shows a corner point, where, as in the previous case, the computed neighborhood is defined on just one of the three possible faces. In both cases, the choice of one of the involved faces is not relevant for the rest of our denoising pipeline.

2.6 Normal filtering

Our normal filtering step consists of four main operations applied sequentially. These operations are explained as follows.

2.6.1 Normal estimation using anisotropic neighborhoods

Once we compute the anisotropic neighborhoods, we use them to estimate feature-preserving point normals. The anisotropic neighborhood for a point p_i is defined by a membership function $\mathbf{u}$ with values between 0 and 1. We

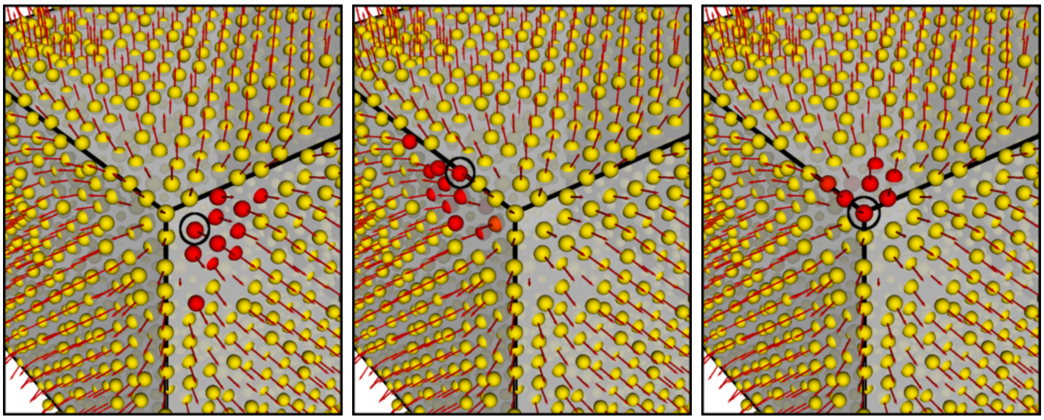


Figure 2.3: Examples of anisotropic neighborhoods computed using $\alpha = 1$, $\beta = 0.1$, and $\gamma = 0.5$ on the Cube point cloud. The point color maps the membership value, which goes from 0 to 1 (yellow to red). The evaluated point is marked by a black circle. The point regular normals are represented by the red arrows.

apply a simple threshold operation to select the member points and construct the neighborhood $\mathscr{N}_a(p_i) = \{p_j \in \mathcal{P} | u_j > \tau_u\}$, where τ_u defines which is an acceptable membership function value. Using all the points included in $\mathscr{N}_a(p_i)$, we compute the PCA-based normal, whose orientation consistency is controlled by the *regular normals*. These anisotropic neighborhood normals enhance feature regions, as shown in Figure 2.4.

The selection of τ_u is not critical because when using the appropriate values for α, β, and γ, the values of $\mathbf{u}$ tend to be close to 0 or 1. Differently from the functional proposed in [55], we do not force smooth $\mathbf{u}$ transitions between neighboring points, so the optimization process is guided by the normal difference and the spatial distance only. Since the input is a noisy point cloud, including more points with low membership function values (e.g., 0.2) increases the normal variability and the distance to the evaluated point, compared to a solution with a few points with values of $\mathbf{u}$ equal to 1.

2.6.2 Normal smoothing using bilateral filter

Since the anisotropic neighborhood points are noisy, the resulting PCA-based normals can be noisy too. For this reason, we smooth them using a bilateral filtering scheme, which aims to preserve high normal variation at feature regions. The bilateral filter is applied iteratively to the normal field by considering the spatial and normal distance. The new normal $\tilde{\mathbf{n}}_i$ for each iteration is computed as follows:

$$\tilde{\mathbf{n}}_i = \frac{1}{W_n(p_i)} \sum_{p_j \in \mathscr{N}_r(p_i)} w_{ij}\mathbf{n}_j, \tag{2-4}$$

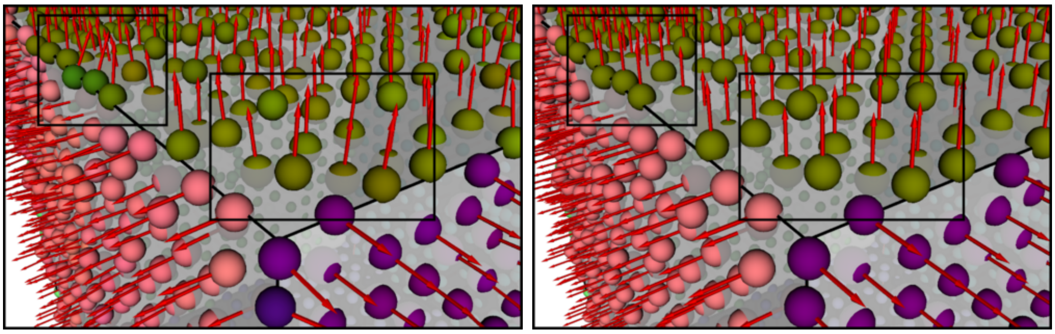


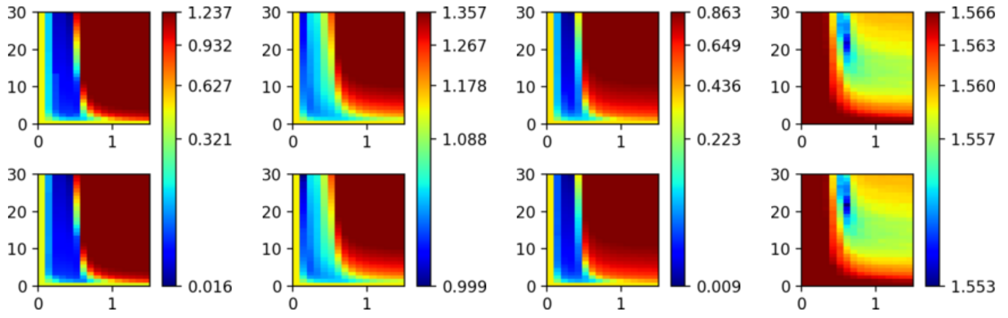
Figure 2.4: Normal smoothing using bilateral filtering on the Cube point cloud. The red arrows represent the normals. The point color maps the normal direction. Left: anisotropic neighborhood normals. Right: smoothed normals.

where $w_{ij} = K_{ns}\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right) K_{nn}\left(\|\mathbf{n}_i - \mathbf{n}_j\|\right)$, K_{ns} and K_{nn} are Gaussian kernel functions used to smooth spatial and normal distances, $W_n(p_i)$ is a normalization factor, i.e. $W_n(p_i) = \sum_{p_j \in \mathscr{N}_r(p_i)} w_{ij}$, and the behavior of the kernels K_{ns} and K_{nn} is defined by the standard deviations σ_{ns} and σ_{nn}, respectively. The number of smoothing iterations is defined by the parameter n_{ns}. Figure 2.4 shows an example where the noisy anisotropic neighborhood normals are processed, obtaining smoother normals and preserving high variation at feature regions. These normals better represent the underlying surface.

This operation is used to refine anisotropic neighborhood normals. Thus, we apply just a few iterations using a low value for σ_{nn} and fixing σ_{ns} to be a proportion of the average edge length l_μ. The first row in Figure 2.5 shows the normal estimation error on four test cases using different number of iterations and different σ_{nn} values. To measure normal estimation error, we use the *Root Mean Square measure with a threshold for multiple normals* ($RMSM_\tau$), introduced in [57]. The bottom left corner location of each sub-figure represents the error of the anisotropic neighborhood normals. Note that for all the cases, we can minimize the normal error by applying a few smoothing iterations with an appropriate σ_{nn} value. For the first 3 test cases, we should avoid values of σ_{nn} greater than 0.5. The fourth test case represents a surface with smooth features; for this reason, $\sigma_{nn} = 0.6$ seems to show greater improvement. For more details about the test cases and the error measurement, see Section 2.9.

2.6.3 Rough feature classification and neighborhood clustering

This operation is used for the normal correction operation and the sharp feature detection step. The idea is to estimate feature candidate points and cluster their *regular neighborhoods* $\mathscr{N}_r$, such that each cluster represents a piecewise smooth region regarding the processed normals.



2.5(a): Cube 2.5(b): Fandisk 2.5(c): Octahedron 2.5(d): RockerArm

Figure 2.5: Normal estimation error using $RMSM_\tau$ on the synthetic point clouds. For each sub-figure, the horizontal axis represents different values for σ_{nn}, the vertical axis represents different values for n_{ns}, and the color represents the $RMSM_\tau$ values. The color map follows an exponential behavior. The first row corresponds to the results after normal smoothing and the second row corresponds to the results after normal correction.

First, we select an initial set of candidate feature points $\mathcal{P}^m$, based on the maximum normal variation within their *small neighborhoods* $\mathscr{N}_s$:

$$\mathcal{P}^m = \left\{ p_i \in \mathcal{P} \middle| \max_{p_j, p_k \in \mathscr{N}_s(p_i)} \|\mathbf{n}_j - \mathbf{n}_k\| > \tau_n \right\}, \tag{2-5}$$

where τ_n is a threshold value used to define if the difference between normals is sufficient to consider the corresponding point as a feature.

As in [35], we define three types of points which are flat, edge, and corner. We assume that those points which are not included in $\mathcal{P}^m$ are flat. Then, considering just the candidate feature points, we cluster their corresponding *regular neighborhoods* $\mathscr{N}_r$ to obtain normal-based piecewise smooth regions. The clustering process uses the processed normals as input and is explained in the following.

The processed normals can be represented as points on the Gauss sphere, so we select the 3 farthest points and connect them, generating a triangle within the sphere. In the case of flat points, we expect that this triangle will present an area close to zero since the Gauss sphere points will be close to each other. In the case of edge points, we expect two dominant locations, generating an irregular triangle with an area close to zero. In the case of corner points, we expect distant points and a triangle with a larger area.

Based on these assumptions, we define a threshold triangle area τ_{ta} to decide if the evaluated point can be considered as a corner point candidate. If not, we check if it is an edge point candidate by measuring the largest distance between the 3 sampled points. If this distance is higher than τ_n, we consider the corresponding point as an edge point candidate. Otherwise, the point is considered as a flat point candidate. Although we filtered flat points in the

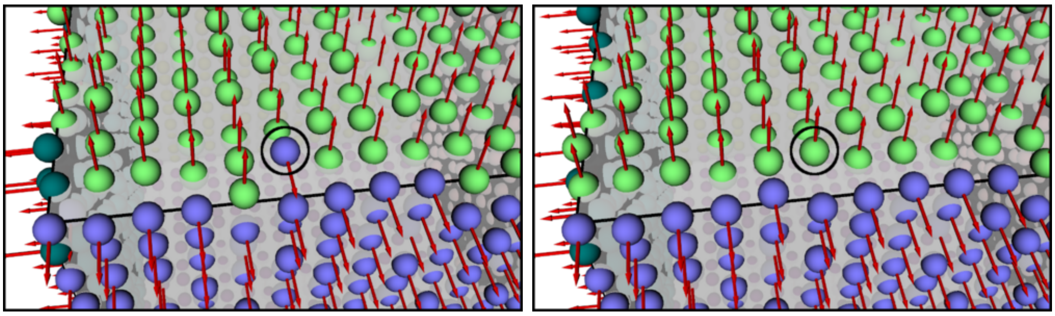


Figure 2.6: Normal correction on the Cube point cloud. The red arrows represent the normals. The point color maps the normal direction. The circled point presents an incorrect normal direction which is modified after normal correction. Left: normals generated after the smoothing operation. Right: corrected normals.

feature candidate selection operation, we filter them again because here we use $\mathscr{N}_r$ instead of $\mathscr{N}_s$ and $\mathscr{N}_r$ does not necessarily include $\mathscr{N}_s$. Let us define $\mathcal{P}^{fc}$, $\mathcal{P}^{ec}$, and $\mathcal{P}^{cc}$, as the sets containing the flat, edge and corner candidate points, respectively. $\mathcal{P}^{fc}$ also contains the points not included in $\mathcal{P}^{m}$. Based on this classification, we cluster the neighborhood points. If the evaluated point is a flat candidate, we assign a single cluster that contains all the *regular neighborhood* points. If the point is an edge candidate, we define two clusters whose seeds are the most distant point normals. Then, the points are assigned to the cluster with the closest seed, regarding normal difference. Finally, if the point is a corner candidate, we define three clusters whose seeds are the 3 sampled normals. The points are assigned as in the previous case.

As mentioned before, these operations are required in two different stages. In the case of the normal correction operation, we do not include the evaluated points in their corresponding clusters because the idea of normal correction is to select the neighboring cluster that better fits the evaluated point. In the case of the sharp feature detection step, we keep the evaluated points in their corresponding clusters because they are based on corrected normals, resulting in more reliable clusters used to define sharp feature regions.

We adopt these simple operations because we expect clean normals as input. So, τ_n and τ_{ta} can be tuned, ignoring the presence of noise. τ_n works as a feature threshold, and τ_{ta} defines possible corner points. Although the number of clusters is limited to the maximum number of seeds, i.e. 3, we can approximate various feature types using them.

2.6.4 Normal correction

Due to the initial noisy PCA-based normals and the usage of non-ideal parameters for the anisotropic neighborhood computation step, some estimated normals can be pointing in the wrong direction, especially for the points that are close to sharp feature regions. In this operation, we correct these undesired normals to better represent the underlying geometry and to avoid problems in the following steps.

Once we define the feature candidate points, i.e. $\mathcal{P}^{ec} \cup \mathcal{P}^{cc}$, we evaluate their regular neighborhood clusters to define which of them are the ones that better fit the corresponding points. Recall that the clustering applied for this operation does not include the evaluated points. To measure how well a cluster fits the feature candidate point, we use two types of point-to-cluster distances.

First, assuming that each cluster represents a plane, we compute the average point-to-plane distance considering all the planes formed by the cluster points and their corresponding normals. Considering the evaluated point p_i and a cluster $\mathcal{C}$, the average point-to-plane distance d_{plane} is defined as follows:

$$d_{plane}(p_i, \mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{p_j \in \mathcal{C}} |\langle \mathbf{n}_j, \mathbf{x}_i - \mathbf{x}_j \rangle| \,, \tag{2-6}$$

where $|\mathcal{C}|$ denotes the number of elements in $\mathcal{C}$.

Second, we assume that each cluster represents a bounded plane region. To define this region, we compute the average normal $\mathbf{n}_{\mathcal{C}}$ and the average position $\mathbf{x}_{\mathcal{C}}$ for the cluster $\mathcal{C}$, representing the cluster's plane. We project all the cluster points on this plane to use them to compute a 2D Convex Hull (CH) shape that represents the boundaries of the cluster. Let us denote the 2D points of the CH as the ordered set $\mathcal{C}^{\pi}_{ch}$ which also represents the CH polygon. The distance d_{ch} to this region is defined as follows:

$$d_{ch}(p_i, \mathcal{C}) = \begin{cases} d_{ch\_out}(p_i, \mathcal{C}) & \text{if } \mathbf{x}^{\pi}_i \text{ lies out of CH shape} \\ 0 & \text{otherwise} \end{cases} \,, \tag{2-7}$$

where $\mathbf{x}^{\pi}_i$ is the projection of $\mathbf{x}_i$ on the plane defined by $\mathbf{n}_{\mathcal{C}}$ and $\mathbf{x}_{\mathcal{C}}$. The function d_{ch_out} is defined as follows:

$$d_{ch\_out}(p_i, \mathcal{C}) = \min_{p_j \in \mathcal{C}^{\pi}_{ch}} d_{line}\left(\mathbf{x}^{\pi}_i, \mathbf{x}^{\pi}_j, \mathbf{x}^{\pi}_{(j+1) \bmod |\mathcal{C}^{\pi}_{ch}|}\right) , \tag{2-8}$$

where $|\mathcal{C}^{\pi}_{ch}|$ denotes the number of points in $\mathcal{C}^{\pi}_{ch}$, and d_{line} measures the point-to-line segment distance given the target point and the line segment end-points. In other words, $d_{ch\_out}(p_i, \mathcal{C})$ measures the distance from p_i to the CH polygon in 2D.

We define the final point-to-cluster distance as $d_{cluster} = 0.5d_{plane}(p_i, \mathcal{C}) + 0.5d_{ch}(p_i, \mathcal{C})$. Then, for each $p_i \in \mathcal{P}^{ec} \cup \mathcal{P}^{cc}$, we define as $\mathbf{n}_{\mathcal{C}}^{*}$, the average normal of the closest cluster, regarding the distance $d_{cluster}$. If $\|\mathbf{n}_i - \mathbf{n}_{\mathcal{C}}^{*}\| > \tau_n$ we assign $\mathbf{n}_{\mathcal{C}}^{*}$ as the new normal for p_i. We can apply this corrector operation iteratively, where the number of iterations is defined by the parameter n_{nc}. However, the more iterations, the more chance of introducing artifacts. In our preliminary experiments, we found that 2 or 3 iterations are enough to improve the smoothed normals. The second row in Figure 2.5 shows the $RMSM_{\tau}$ results when using this corrector operation, where we can perceive a little improvement. Figure 2.6 shows an example of how some normals are corrected, generating a more reliable normal field. This operation is one of the main novelties of our proposal, and it is necessary to define accurate edge lines for the underlying surface.

In the case of true edge and corner points, the distance to the non-selected clusters can be very similar to the distance to the closest cluster. That allows us to adopt a multi-normal scheme useful for the point updating step, particularly for the updating of flat points (more details in the next section). Let us denote the closest cluster as $\mathcal{C}^*$. We consider the average normal $\mathbf{n}_{\mathcal{C}}$ of another cluster $\mathcal{C}$ as an additional normal, if $d_{plane}(p_i, \mathcal{C}) < d_{plane}(p_i, \mathcal{C}^*) + \epsilon_{mn}$, where ϵ_{mn} is a tolerance value used to define the proximity criteria. We do not consider the distance d_{ch} for this estimation because the CH of the closest cluster can include consecutive edge points, generating considerable d_{ch} distance with the other clusters.

2.7 Sharp feature detection

Using the corrected normals, we aim to detect feature points precisely by estimating the edge lines of the underlying surface and selecting just the closest points to them as features, avoiding the inclusion of nearby points that can be considered part of the surrounding flat regions.

First, we apply the same process used for normal filtering to cluster point *regular neighborhoods* of feature candidate points, but, in this case, including the evaluated point p_i in the corresponding closest cluster $\mathcal{C}_1$. Let us define the other possible clusters as $\mathcal{C}_m$, where $m \in \{2, 3\}$ for corner candidate points and $m = 2$ for edge candidate points. Then, to define if p_i is a feature point, p_i should be the closest point to the edge line between $\mathcal{C}_1$ and $\mathcal{C}_m$, within a narrow neighborhood perpendicular to the edge direction. Consider $\mathscr{P}_i$ and $\mathscr{P}_j$ as the planes conformed by the coordinates and normals of p_i and a point $p_j \in \mathcal{C}_m$. The intersection between $\mathscr{P}_i$ and $\mathscr{P}_j$ is considered as the edge line

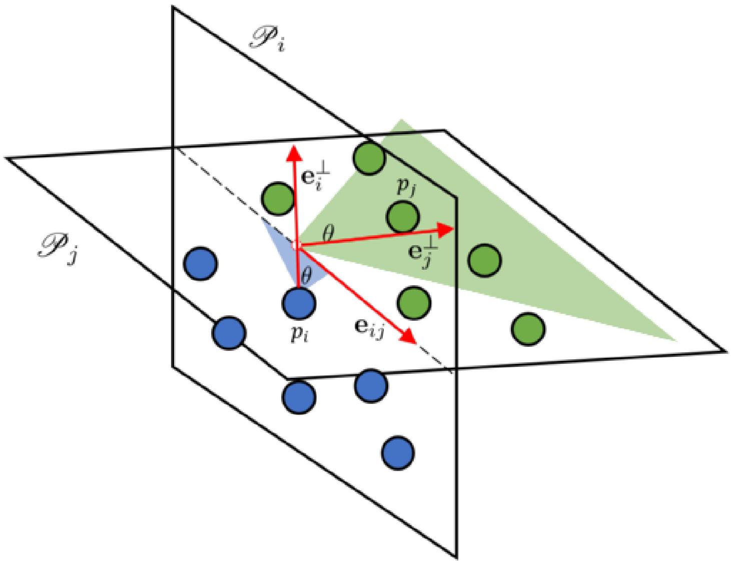


Figure 2.7: Sharp feature detection narrow neighborhood. The blue points correspond to $\mathcal{C}_1$. The green points correspond to $\mathcal{C}_m$. The shaded region defines the field of view on $\mathscr{P}_i$ and $\mathscr{P}_j$. The points that lie on this region are considered as part of the narrow neighborhood.

for this pair of points, whose direction we denote as $\mathbf{e}_{ij}$. Then, we compute a vector $\mathbf{e}_i^{\perp}$ on $\mathscr{P}_i$ pointing in a direction orthogonal to $\mathbf{e}_{ij}$, i.e. $\mathbf{e}_i^{\perp} = \mathbf{n}_i \times \mathbf{e}_{ij}$. Following the same idea, we compute a vector $\mathbf{e}_j^{\perp}$ on $\mathscr{P}_j$ orthogonal to $\mathbf{e}_{ij}$, i.e. $\mathbf{e}_j^{\perp} = \mathbf{n}_j \times \mathbf{e}_{ij}$. Using $\mathbf{e}_i^{\perp}$ and $\mathbf{e}_j^{\perp}$, we define the narrow neighborhood for both clusters as follows. For $\mathcal{C}_1$, we project all the cluster points on the plane $\mathscr{P}_i$. Then, we trace a line centered at $\mathbf{x}_i$ with direction $\mathbf{e}_i^{\perp}$. This line, combined with a tolerance angle θ, define a field of view on $\mathscr{P}_i$. Thus, if we trace a line from $\mathbf{x}_i$ to another projected point included in $\mathcal{C}_1$ and the angle between both lines is lower than $\theta/2$, we consider the corresponding point as part of the narrow neighborhood. For $\mathcal{C}_m$, first we define the projection of $\mathbf{x}_i$ on $\mathscr{P}_j$ as $\mathbf{x}_i^{\pi_j}$. Then, we project all points included in $\mathcal{C}_m$ onto the plane $\mathscr{P}_j$. Similarly to the previous case, we trace a line centered at $\mathbf{x}_i^{\pi_j}$ with direction $\mathbf{e}_j^{\perp}$ which combined with the same tolerance angle θ, defines a field of view on $\mathscr{P}_j$. As in the previous case, we consider the points that lie on the field of view as part of the narrow neighborhood. Figure 2.7 illustrates how this narrow neighborhood is defined.

Then, for each point p_j, we check if p_i is the closest point to the intersection line between the planes $\mathscr{P}_i$ and $\mathscr{P}_j$, just considering the points included in the corresponding narrow neighborhoods of $\mathcal{C}_1$ and $\mathcal{C}_m$. If p_i is the closest point for all $p_j \in \mathcal{C}_m$, we consider p_i as a feature point regarding $\mathcal{C}_m$. Using this information, we classify each point p_i into flat, edge, or corner as follows. If p_i is not the closest point considering all the available $\mathcal{C}_m$ clusters,

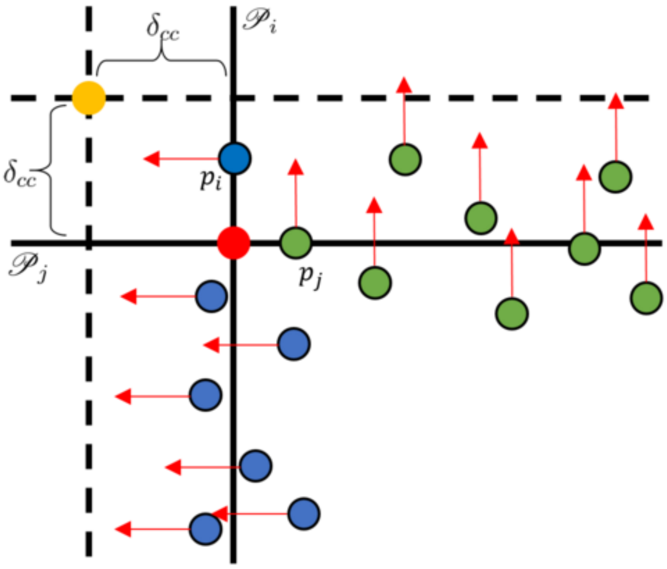


Figure 2.8: Convexity-based displacement of $\mathscr{P}_i$ and $\mathscr{P}_j$. The blue points correspond to $\mathcal{C}_1$. The green points correspond to $\mathcal{C}_m$. The red arrows represent the point normals. The red point represents the edge line between $\mathscr{P}_i$ and $\mathscr{P}_j$. The yellow point represents the edge line after applying the displacement δ_{cc}. Note that p_i is not the closest point to the red point but it is the closest to the yellow one.

we assign it to the set $\mathcal{P}^f$ of flat points. $\mathcal{P}^f$ also includes the flat candidate points $\mathcal{P}^{fc}$. If p_i is the closest point regarding just one cluster $\mathcal{C}'_m$, we assign it to the set $\mathcal{P}^e$ of edge points. For this set of points, we also include a global edge direction $\mathbf{e}_i$ by picking the direction $\mathbf{e}_{ij}$ of the closest edge line regarding the cluster $\mathcal{C}'_m$. If p_i is the closest point regarding two clusters, we assume that it is a possible corner. Then, to verify the latter, we measure if p_i is a local maximum or minimum within $\mathscr{N}_b(p_i)$, regarding an approximated tangent plane. We define this tangent plane using the *regular normal*, computed in the preparing procedure, and the point position $\mathbf{x}_i$. If all the points in $\mathscr{N}_b(p_i)$ are below this plane or all of them are over it, we assign p_i to the set of corner points $\mathcal{P}^c$; otherwise, we assign it to $\mathcal{P}^e$. This analysis do not consider all possible corner types, it focuses on salient corners, i.e. peaks and valleys.

Because the point cloud presents noise, the edge lines can pass through the feature candidate points, allowing non-ideal points to be considered as the closest ones. Figure 2.8 shows an example where the points are projected on the plane defined by $\mathbf{e}_{ij}$. The edge line between p_i and p_j is represented by a single point (red) which is the intersection between the corresponding planes $\mathscr{P}_i$ and $\mathscr{P}_j$, represented as black lines. We expect that p_i should be considered as the edge point in this case because the local neighborhood represents a convex surface. However, p_i is not the closest point to the edge line due to the noisy

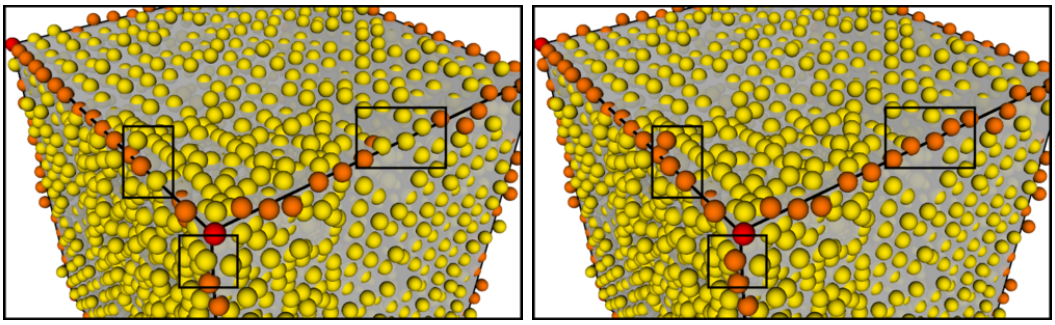


Figure 2.9: Sharp feature detection on the Cube point cloud. Flat points are colored in yellow, edge points are colored in orange and corner points are colored in red. Left: result using $\delta_{cc} = 0$. Right: result using $\delta_{cc} = 0.5 l_{\mu}$.

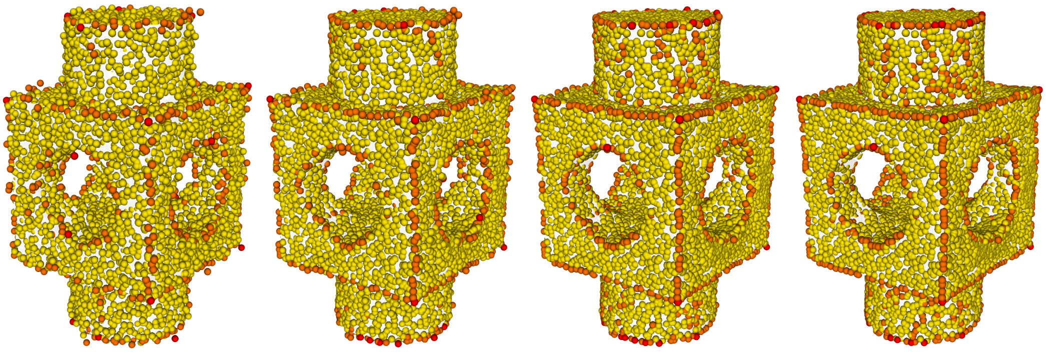


Figure 2.10: Sharp feature detection on the Block point cloud through different denoising iterations. Flat points are colored in yellow, edge points are colored in orange and corner points are colored in red. From left to right: first iteration, second iteration, third iteration, and fourth iteration.

point positions and the estimated normals. To avoid this problem, we estimate a more external edge line for convex regions and a more internal edge line for concave regions, i.e. we apply a displacement of δ_{cc} to the planes $\mathscr{P}_i$ and $\mathscr{P}_j$, following their normal directions for convex points and following their opposite normal direction for concave points. The dashed lines in Figure 2.8 represent the new plane positions, and the yellow point represents the new edge line. Now, p_i is the closest point to the edge line. Figure 2.9 shows an example of how this displacement affects the selection of the sharp feature points. The choice of an appropriate δ_{cc} value depends on the noise level. The more noise, the higher the value of δ_{cc} should be.

To define if a point p_i is convex or concave, we first compute a smooth version of the point cloud using regular Laplacian smoothing on the Riemannian graph defined by $\mathscr{N}_s$. For this process, we use 10 smoothing iterations and 0.2 as the step size. This smooth representation allows us to assess a cleaner local surface approximation for p_i. For each p_i, we compute the average *regular normal* $\overline{\mathbf{n}}_i$ of the neighboring points regarding $\mathscr{N}_s$, and the centroid $\mathbf{c}_i$ of the neighboring points regarding $\mathscr{N}_b$, using the point positions of the smooth representation. Then, let us define the plane defined by p_i on the smooth surface and $\overline{\mathbf{n}}_i$ as $\mathscr{P}$. If $\mathbf{c}_i$ is at a distance greater than ϵ_{cc} from $\mathscr{P}$ and is below it, we consider p_i as convex. If $\mathbf{c}_i$ is at a distance greater than ϵ_{cc} from $\mathscr{P}$ and is over it, we consider p_i as concave. Otherwise, we consider p_i as undefined, and no displacement is applied to $\mathscr{P}_i$ and $\mathscr{P}_j$. We introduced the *big neighborhood* $\mathscr{N}_b$ for the computation of $\mathbf{c}_i$ because $\mathscr{N}_r$ can be too small for the convexity analysis. If we increase the size of $\mathscr{N}_r$, we can disrupt other procedures.

Figure 2.10 shows an example of the behavior of the sharp feature detection method in different iterations of the denoising algorithm. We can see that the feature points are selected in a precise manner, even in the presence of high noise levels, and they rapidly start converging to the edge lines of the underlying surface. In Section 2.9, we show some examples of how the denoising algorithm benefits from this approach.

2.8 Point updating

Once we have a set of filtered normals and the point classification, we update point positions using an adaptation of the method proposed in [35], which applies a different point updating scheme depending on the point class. For all the updating operations, we consider the neighborhood $\mathscr{N}_p$, which denotes the small neighborhood without including p_i, i.e. $\mathscr{N}_p(p_i) = \mathscr{N}_s(p_i) - \{p_i\}$. We first update flat points in an iterative manner to then update edge and corner points simultaneously. As in [35], we constraint the possible displacement of each point to a Euclidean sphere with radius τ_o centered at the original noisy point position. In the following, we describe the updating scheme for each point class. For further details about the updating operations, please refer to [35].

Flat point updating: for each point $p_i \in \mathcal{P}^f$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \upsilon_f \left(\frac{1}{W_p(p_i)} \sum_{p_j \in \mathscr{N}_p(p_i)} w_{ij} \langle \mathbf{n}_j^*, \mathbf{x}_j - \mathbf{x}_i \rangle \mathbf{n}_i \right), \tag{2-9}$$

where $w_{ij} = K_{ps}\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right) K_{pn}\left(\|\mathbf{n}_i - \mathbf{n}_j^*\|\right)$, K_{ps} and K_{pn} are Gaussian kernel functions used to smooth spatial and normal distances, $W_p(p_i)$ is a normalization factor, i.e. the sum of all the used weights w_{ij}, the behavior of the kernels K_{ps} and K_{pn} is defined by the standard deviations σ_{ps} and σ_{pn}, respectively, $\mathbf{n}_j^* = \arg\max_{\mathbf{n}_k \in \mathcal{M}_j} K_{pn}\left(\|\mathbf{n}_i - \mathbf{n}_k\|\right)$, $\mathcal{M}_j$ denotes the set of multi-normals for the point p_j, computed during normal correction, and υ_f controls the amount of displacement for each update. The multi-normals allow the usage of close edge or corner points whose main normal is pointing in a quite different direction from $\mathbf{n}_i$. This updating operation is applied iteratively, where n_{fp} defines the number of iterations. The idea is to smooth flat points using the neighboring points with similar normals and constrain the displacement to the corresponding normal direction.

Edge point updating: for each point $p_i \in \mathcal{P}^e$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\begin{aligned}\tilde{\mathbf{x}}_i =&\mathbf{x}_i + \upsilon_e \left(\left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j^\pi \left(\mathbf{n}_j^\pi\right)^T + \mathbf{e}_i\mathbf{e}_i^T \right)^{-1} \right.\\ &\left. \left(\sum_{p_j \in \mathcal{N}_p(p_i)} \left(\mathbf{n}_j^\pi \left(\mathbf{n}_j^\pi\right)^T \mathbf{x}_j + \mathbf{e}_i\mathbf{e}_i^T\mathbf{x}_i \right) \right) - \mathbf{x}_i \right),\end{aligned} \tag{2-10}$$

where $\mathbf{n}_j^\pi = \mathbf{n}_j - \langle \mathbf{n}_j, \mathbf{e}_i \rangle \mathbf{e}_i$, and υ_e controls the amount of displacement. In this operation, we project all the points and their normals on the plane defined by the edge direction $\mathbf{e}_i$. Thus, we can define surrounding lines instead of planes using the projections of neighboring points. The idea is to minimize the distance to these lines, meeting an intersection point in the ideal case, i.e. without noise.

Corner point updating: for each point $p_i \in \mathcal{P}^c$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \upsilon_c \left(\left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j\mathbf{n}_j^T \right)^{-1} \left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j\mathbf{n}_j^T\mathbf{x}_j \right) - \mathbf{x}_i \right), \tag{2-11}$$

where υ_c controls the amount of displacement. The intuition of this operation is to minimize the distance of p_i to all the surrounding approximated planes. In the ideal case, i.e. without noise, the target position for p_i should be the intersection of all of these planes.

2.9 Results

Due to the nature of our method, we compare it against point cloud denoising, normal estimation, and feature detection methods. For numerical evaluation, we use four non-uniform mesh models whose vertices are considered as input point clouds and are corrupted with synthetic noise (See Figure 2.11). We name these synthetic point clouds as Cube (Figure 2.12), Fandisk (Figure 2.13), Octahedron, and RockerArm, which are corrupted with a Gaussian noise in random directions with $\sigma = 0.3l_{ro}$, $\sigma = 0.28l_{ro}$, $\sigma = 0.3l_{ro}$, and

$\sigma = 0.3l_{ro}$, respectively. These point clouds are used in [35]. For visual evaluation, we use point clouds generated from raw scans of objects with sharp features. We name them as Shutter (Figure 2.14), Iron (Figure 2.15), and Tool (Figure 2.16). This data is used in [40]. We also consider the Gargoyle point cloud (Figure 2.17), which is a partial scan included in [35], the Twelve and Cube2 point clouds (Figure 2.18), which are included in [43], the Block model (Figure 2.19) used in [55], and the Mug model included in [58] (Figure 2.20).

2.9.1 Parameter setting

Although we introduce several parameters in our denoising pipeline, we define the following default values: $\epsilon_d = 0.01$, $k = 50$, $r_s = 1.5$, $r_r = 3$, $r_b = 2.5r_r$, $\alpha = 1$, $\beta = 0.1$, $\gamma = 0.5$, $a_0 = 0.35\sum_{a_i \in \mathbf{a}} a_i$, $\tau_u = 0.3$, $\sigma_{ns} = 1.5l_\mu$, $\sigma_{nn} = 0.3$, $n_{ns} = 7$, $\tau_n = 0.2$, $\tau_{ta} = 0.2$, $n_{nc} = 2$, $\epsilon_{mn} = 0.1l_\mu$, $\theta = 110°$, $\delta_{cc} = 0.5l_\mu$, $\epsilon_{cc} = 0.2l_\mu$, $\tau_o = 2l_\mu$, $\sigma_{ps} = 2l_\mu$, $\sigma_{pn} = 0.5$, $\upsilon_f = 0.3$, $n_{fp} = 3$, and $\upsilon_e = \upsilon_c = 0.5$. These values were defined empirically by evaluating the corresponding operations independently on a set of test cases. The parameters that we should tune are n_{int} and n_{ext}, which define the number of internal and external iterations. These parameters depend on the level of noise of the input; the more noise, the more iterations are necessary. The default parameters can generate acceptable results for all our test cases, however, we tune some of them for a few test cases to improve visual results.

The tuned parameters for each point cloud are described as follows. Cube: ($n_{ext} = 9$, $n_{int} = 1$). Fandisk: ($n_{ext} = 1$, $n_{int} = 5$). Octahedron: ($n_{ext} = 1$, $n_{int} = 10$). RockerArm: ($n_{ext} = 1$, $n_{int} = 3$). Shutter: ($n_{ext} = 2$, $n_{int} = 5$). Iron: ($n_{ext} = 3$, $n_{int} = 5$, $k = 75$, $r_r = 4$). Tool: ($n_{ext} = 1$, $n_{int} = 5$). Gargoyle: ($n_{ext} = 1$, $n_{int} = 3$, $r_r = 2$, $n_{ns} = 1$). For the Block point clouds, we fixed $n_{int} = 1$ and n_{ext} was modified depending on the noise level. The normal precomputation for all cases is performed using the PCA-based normal estimation (20 neighbors) implemented in the MeshLab software [59].

The *regular neighborhoods* are used in several operations of the pipeline, and their size is based on k and r_r. Depending on the point cloud feature sizes, these parameters can be tuned to preserve them better. The smaller the feature size, the smaller the regular neighborhood should be. For example, the Gargoyle point cloud presents very small features represented by a few points, while the Iron point cloud presents bigger features represented by several points.

The bilateral filtering parameters, i.e. n_{ns}, σ_{ns}, and σ_{nn}, allows us to control the smoothness of the normal field. We do not require too many

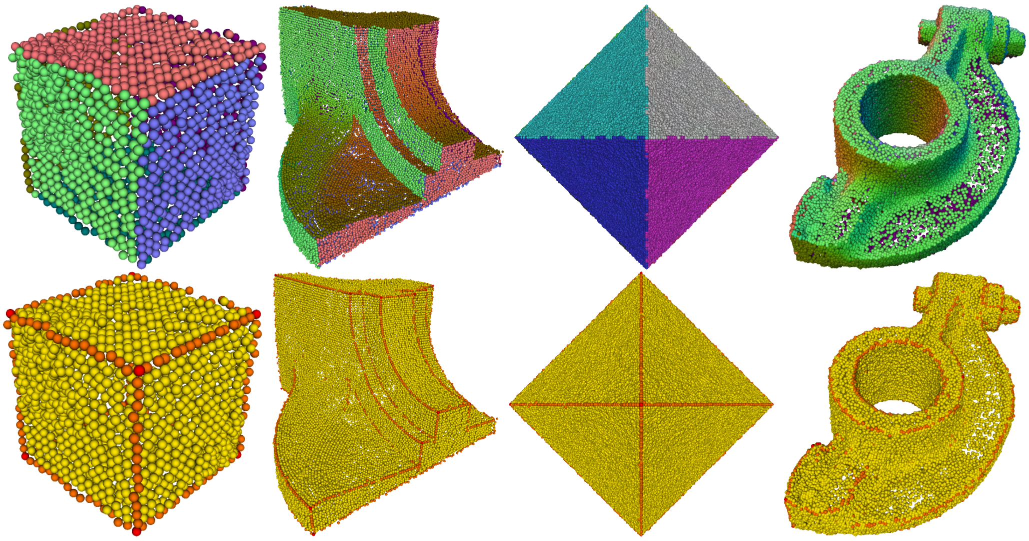


Figure 2.11: Normal estimation (first row) and feature detection (second row) results obtained from the first iteration of our denoising pipeline. For normal estimation, the point color maps the normal direction. For feature detection, flat points are colored in yellow, edge points are colored in orange, and corner points are colored in red. From left to right: Cube, Fandisk, Octahedron, and RockerArm.

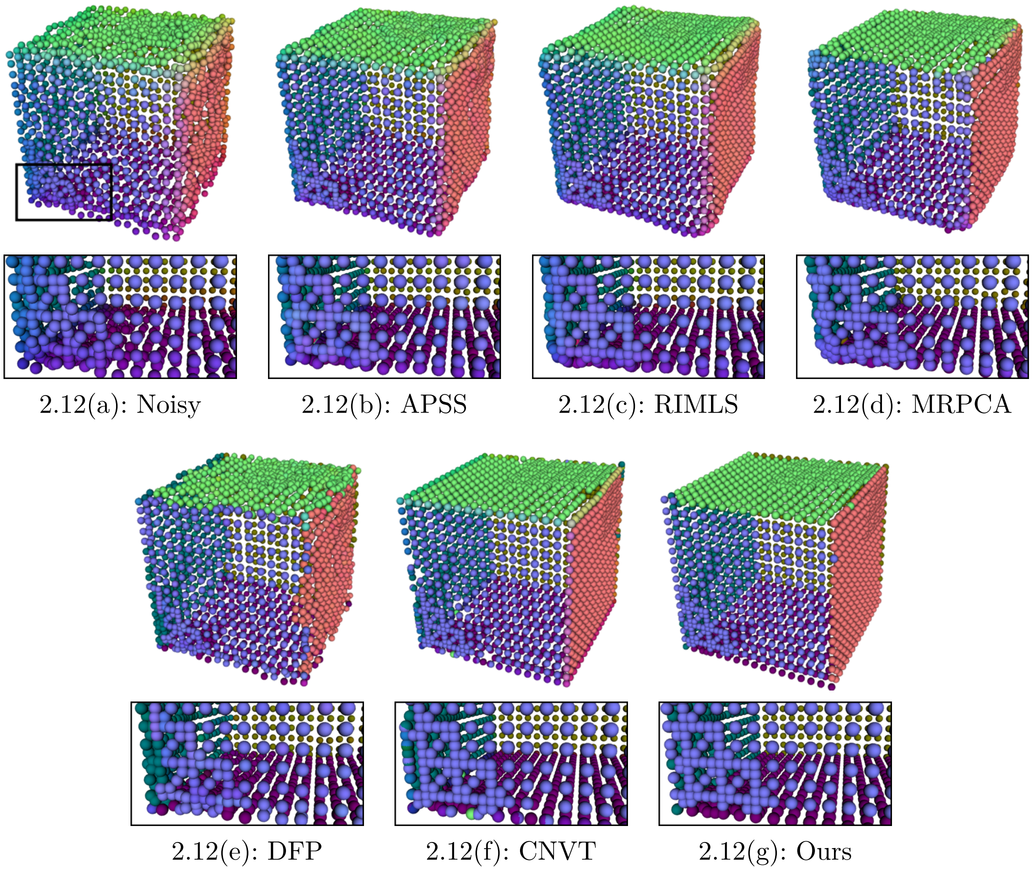
iterations when the noise level is low because the anisotropic neighborhood normals will present low perturbation. In case we want to deal with smooth features, we can tune these parameters to obtain a normal field that better fits the smooth surfaces, as shown in Figure 2.5. However, we can blur the sharp features.

In addition to n_{int} and n_{ext}, when the noise level is high, we should also tune δ_{cc} and τ_o to represent more external or internal edge lines for sharp feature detection and to increase the distance constraint to the original point positions for point updating.

Although the other parameters can be tuned, in our preliminary experiments, we did not achieve a considerable improvement in the denoising task. Further, many of them are independent of the evaluated point cloud, or their default values are based on relative measurements that make them robust.

2.9.2 Normal estimation evaluation

Normal estimation can be compared in different ways. In this experiment, we adopt the feature preserving evaluation described in [57]. Because the synthetic noisy point clouds correspond to the perturbation of the vertices of a clean mesh, we can compute the per-point ground-truth multi-normals by assigning the normals of the clean faces shared by the corresponding vertex.



2.12(a): Noisy 2.12(b): APSS 2.12(c): RIMLS 2.12(d): MRPCA

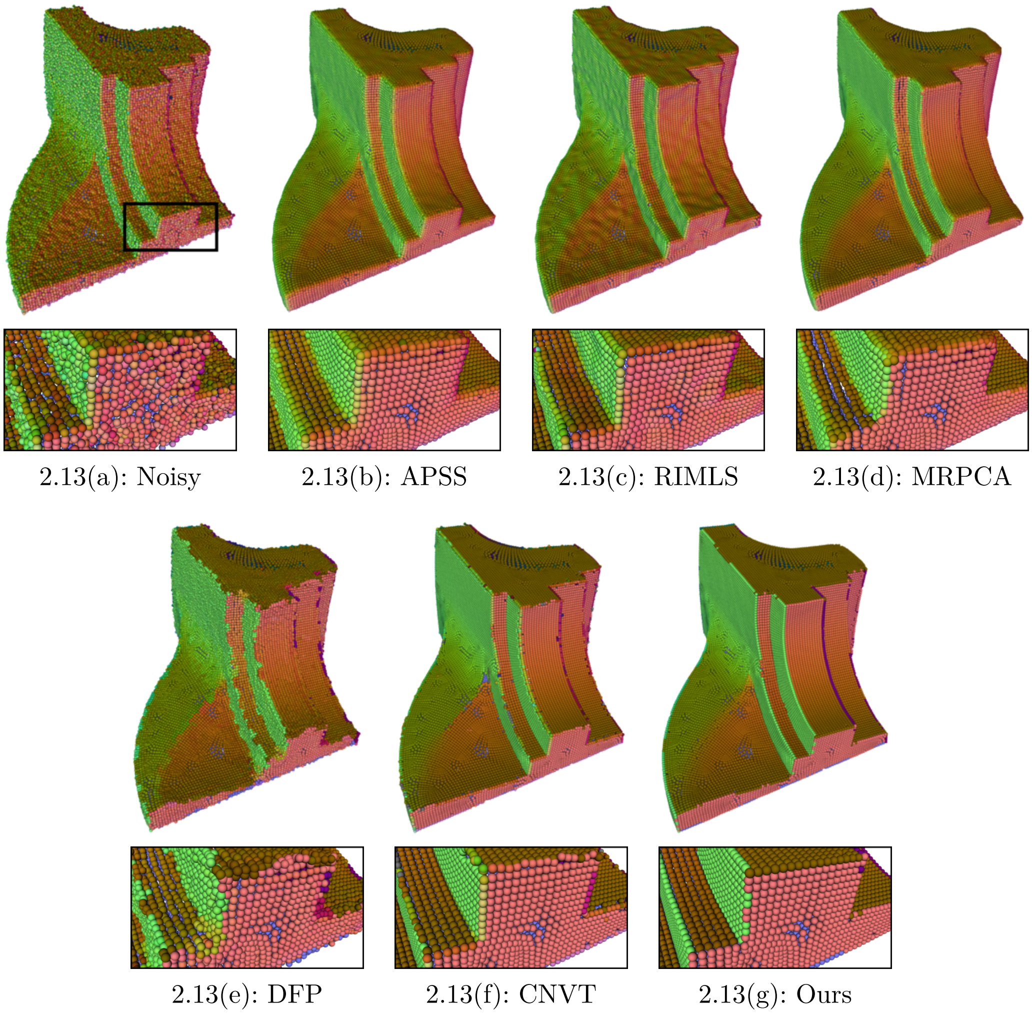
2.12(e): DFP 2.12(f): CNVT 2.12(g): Ours

Figure 2.12: Results obtained on the Cube point cloud. The point color maps the direction of the normals.

Then, given an estimated normal on the point cloud data, we can compare it with the most similar normal from the available ground-truth multi-normals. The $RMSM_\tau$ allows us to measure the normal estimation error.

For our method, we select the normal filtering results of the first iteration in the denoising pipeline. We compare them with the results of [60], [61], [62], [57], and [53], which we name as PCA, JET, VCM, PCV, and DFP, respectively. For the PCA, JET, and VCM methods, we use the implementation provided in the CGAL library [63]. In the case of PCA and JET, we use 18 neighbors since they generate the best results regarding $RMSM_\tau$. In the same sense, for the VCM method, we use $1.5l_\mu$ for both the offset radius and the convolutional radius. For the PCV method, we use the implementation provided by the authors, considering $S^* = 100$ for all the point clouds, as in their experiments. In the case of the DFP method, we use the implementation of the authors, selecting the predicted normals of the first iteration in their denoising pipeline.

Table 2.1 shows the $RMSM_\tau$ results on the four main synthetic point clouds. For the Cube and Fandisk cases, our method clearly outperforms the



2.13(a): Noisy

2.13(b): APSS

2.13(c): RIMLS

2.13(d): MRPCA

2.13(e): DFP

2.13(f): CNVT

2.13(g): Ours

Figure 2.13: Results obtained on the Fandisk point cloud. The point color maps the direction of the normals.

others. For the Octahedron, we obtain the best results together with the PCV method. In the case of the RockerArm, all the methods achieve results with similar $RMSM_\tau$. The first row in Figure 2.11 shows our normal results visually. Observe that our normals are piecewise smooth, and the transitions between flat regions are enhanced. Also, these normals are capable of keeping certain smoothness at curved surfaces, as shown in the Fandisk model. Note that the parameters used for normal estimation are the same for all the input point clouds.

2.9.3 Feature detection evaluation

The feature detection problem can be modeled as a classification problem, and we can evaluate it using the accuracy metric. To generate ground-truth feature points on the synthetic noisy point clouds, for each point we pick the corresponding vertex in the clean mesh and check if its shared face normals

Table 2.1: Normal estimation results using $RMSM_\tau$.

Method	Cube	Fandisk	Octahedron	RockerArm
PCA	0.89779	1.32275	1.33386	**1.56447**
JET	0.91207	1.32273	1.33400	1.56509
VCM	0.91020	1.31394	1.33260	1.56515
PCV	0.06235	1.26109	**1.30732**	1.56655
DFP	0.51596	1.29163	1.31418	1.56811
Ours	**0.03169**	**1.24833**	1.30740	1.56730

Table 2.2: Feature detection accuracy results.

Method	Cube	Fandisk	Octahedron	RockerArm
VCM	0.749	0.835	0.905	**0.857**
FREEUPC	0.844	0.789	0.915	0.839
DFP	0.724	0.784	0.879	0.713
Ours	**0.997**	**0.932**	**0.953**	0.811

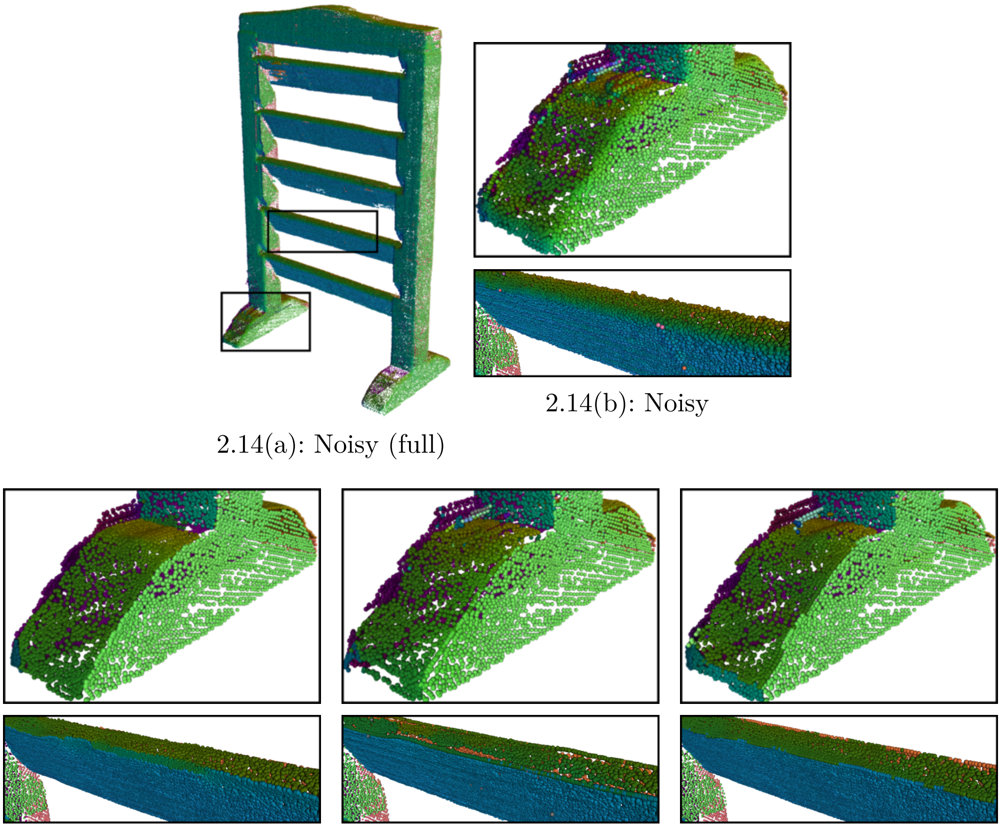
form an angle higher than 18°. If the latter occurs, we label the evaluated point as a feature point. This methodology for ground-truth feature points estimation is also used in [53].

We select $\mathcal{P}^e \cup \mathcal{P}^c$ from the first iteration in our denoising pipeline as the detected features. We compare our results with the results of [62], [64], and [53], named as VCM, FREEUPC, and DFP. We use the CGAL implementation of VCM with $3l_\mu$ as the offset radius and $1.5l_\mu$ as the convolutional radius. For the FREEUPC method, we use the implementation provided by the authors and the following parameters for each point cloud. Cube: ($k = 18$, $\sigma = 0.07$). Fandisk: ($k = 25$, $\sigma = 0.05$). Octahedron: ($k = 25$, $\sigma = 0.05$). RockerArm: ($k = 25$, $\sigma = 0.05$). For both methods, VCM and FREEUPC, we tuned the parameters to obtain the highest accuracy. For the DFP method we select the feature detections results of the first iteration.

Table 2.2 shows the accuracy results. For the Cube, Fandisk, and Octahedron, our method outperforms the others with accuracies higher than 0.93. In the case of the RockerArm, the VCM method achieves the highest accuracy. Our method works better when the features are sharp, which is not the case of the RockerArm. The second row of Figure 2.11 shows our results visually. We can see that our detected features tend to be thin, which is an important condition for our denoising pipeline.

2.9.4 Denoising evaluation

We use two different metrics to perform a numerical evaluation of the denoising results on the synthetic point clouds. First, we measure the double-sided average euclidean distance D_p between the denoised point positions and the ground-truth positions, i.e. the positions of the original point cloud. Second, we use the $RMSM_\tau$ to measure normal error between the denoised normals



2.14(a): Noisy (full)

2.14(b): Noisy

2.14(c): MRPCA

2.14(d): CNVT

2.14(e): Ours

Figure 2.14: Results obtained on the Shutter point cloud. The point color maps the direction of the normals.

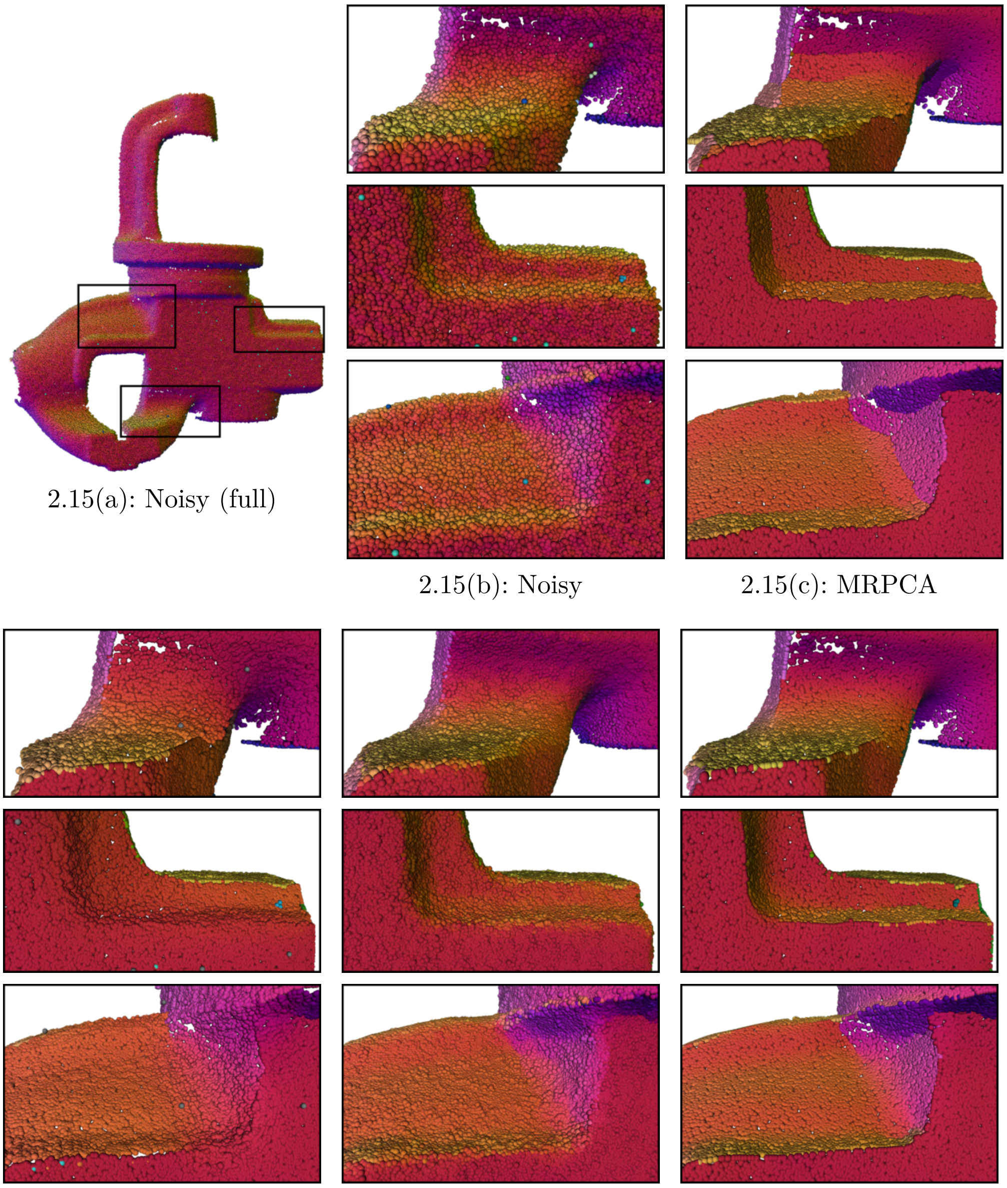
of the last denoising iteration and the ground-truth multi-normals defined on the original point cloud. The correspondance between the point sets is defined by selecting the corresponding closest points.

For the synthetic point clouds, we compare our method with [8], [10], [15], [53], and [35], named as APSS, RIMLS, MRPCA, DFP, and CNVT, respectively. For APSS and RIMLS we use the MeshLab implementation, while for the others, we use the implementations provided by the corresponding authors. The parameters used for the APSS, RIMLS, and CNVT, are the same described in [35]. For the MRPCA method, we tuned the parameters to obtain the best results regarding D_p. These parameters are defined as follows. Cube: ($k = 30$, $\sigma = 15$, $r = 5$). Fandisk: ($k = 30$, $\sigma = 15$, $r = 3$). Octahedron: ($k = 30$, $\sigma = 15$, $r = 4$). RockerArm: ($k = 30$, $\sigma = 15$, $r = 3$). In the case of DFP, we use 2, 2, 3, and 2 as the number of iterations for Cube, Fandisk, Octahedron, and RockerArm point clouds, respectively. These parameters obtained the best results regarding D_p.

Table 2.3 shows the D_p and the $RMSM_\tau$ results. For the Cube and the Octahedron point clouds, our method outperforms the others considering both metrics. In Figure 2.12 we can observe that the APSS, RIMLS, and

Table 2.3: Denoising results.

Method	**Metric**	**Cube**	**Fandisk**	**Octa.**	**Rock.**
APSS	D_p	0.01386	0.00619	0.00117	0.07060
	$RMSM_\tau$	0.79453	0.65859	0.44468	**1.56428**
RIMLS	D_p	0.01828	0.00602	0.00101	**0.06017**
	$RMSM_\tau$	0.87196	0.58685	0.44420	1.56535
MRPCA	D_p	0.01841	0.00577	0.00084	0.07746
	$RMSM_\tau$	0.48890	0.49439	0.31465	1.56547
DFP	D_p	0.01842	0.00711	0.00106	0.08189
	$RMSM_\tau$	0.50299	0.53786	0.36327	1.56814
CNVT	D_p	0.00645	0.01083	0.00043	0.16511
	$RMSM_\tau$	0.48283	0.38188	0.01685	1.56616
Ours	D_p	**0.00385**	**0.00308**	**0.00041**	0.07770
	$RMSM_\tau$	**0.00829**	**0.23504**	**0.00997**	1.56678



2.15(a): Noisy (full)

2.15(b): Noisy

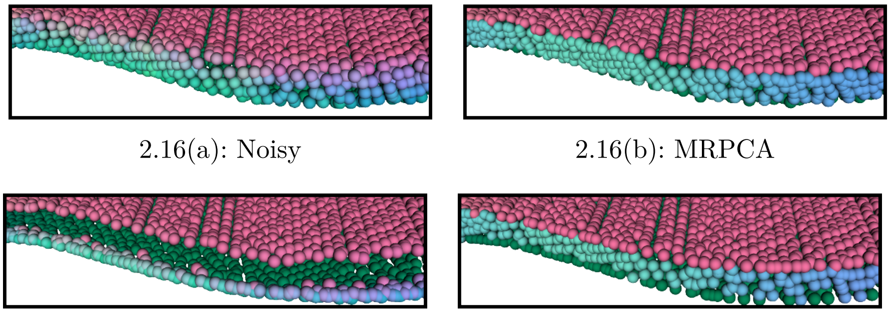
2.15(c): MRPCA

2.15(d): CNVT

2.15(e): FPF

2.15(f): Ours

Figure 2.15: Results obtained on the Iron point cloud. The point color maps the direction of the normals.



2.16(a): Noisy

2.16(b): MRPCA

2.16(c): CNVT

2.16(d): Ours

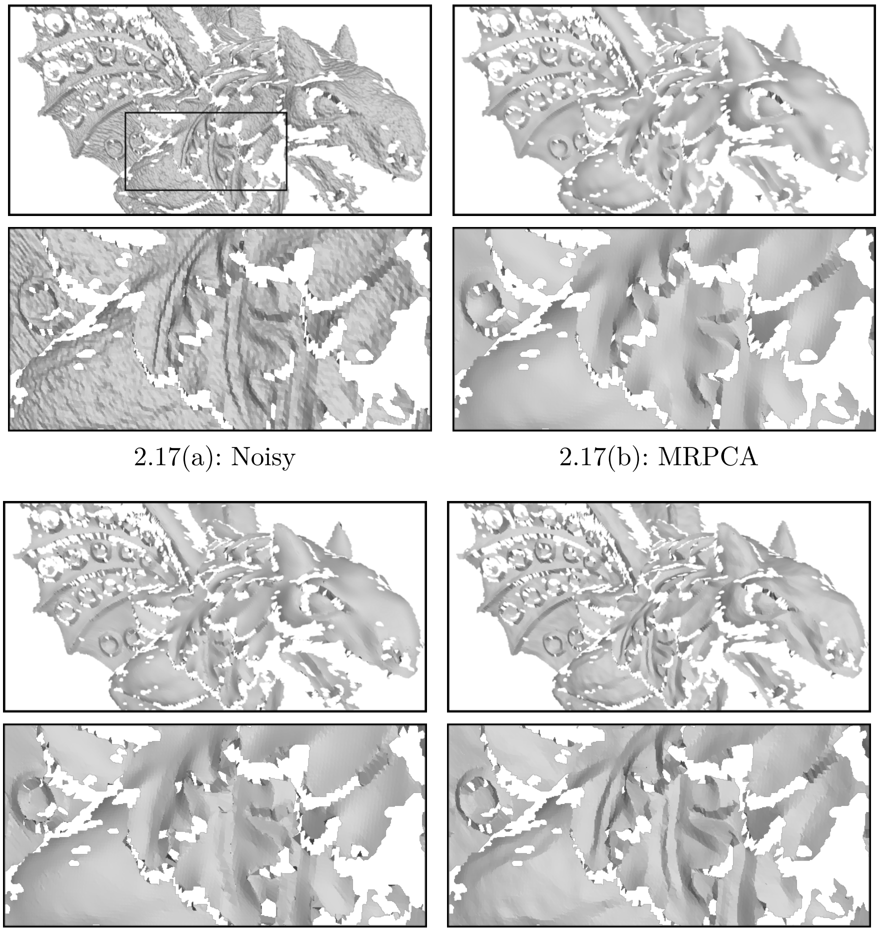
Figure 2.16: Results obtained on the Tool point cloud. The point color maps the direction of the normals.

MRPCA methods generate rounded regions at corners and edges, but the MRPCA produces sharper transitions between normals at edge regions. For some points, the DFP tries to preserve the sharp features; however, the overall results look unstable and noisy. The CNVT result is noise-free at plane regions and better preserves the edges and corners compared to the previous methods. Nevertheless, it presents gaps close to the edges and multiple points converging to the same position. This phenomenon occurs because the coarse feature regions detected by CNVT are displaced to the edge lines. On the other hand, our method presents a clean surface, enhances sharp feature regions, and keeps a more uniform distribution, trying to cover all the represented surface.

Differently from the Cube and the Octahedron, the Fandisk presents curved regions in addition to the flat and sharp regions. Our method also outperforms the others in this case, for both metrics, D_p and $RMSM_\tau$. Figure 2.13 shows the visual results, where we can see the same phenomena shown in the previous cases for the APSS, RIMLS, MRPCA, and DFP methods. Furthermore, the APSS and the RIMLS create false bumpy features. The CNVT generates clean flat and curved regions, but the edges are noisy. Our method is capable of enhancing the sharp feature regions while keeping the smoothness of curved regions.

As mentioned before, the features of the RockerArm are not sharp. The RIMLS method achieves the best result for this case and the CNVT the worst, regarding D_p. Our algorithm enhances the smooth features, generating sharp regions. Depending on the application, this effect may be expected or not. Although we can tune other parameters to obtain better results regarding D_p and a smoother surface, we use the default parameters to show their robustness against different examples.

In addition to the synthetic examples, we also evaluate our method visually on real scans of objects with sharp features. In this case, we compare



2.17(a): Noisy

2.17(b): MRPCA

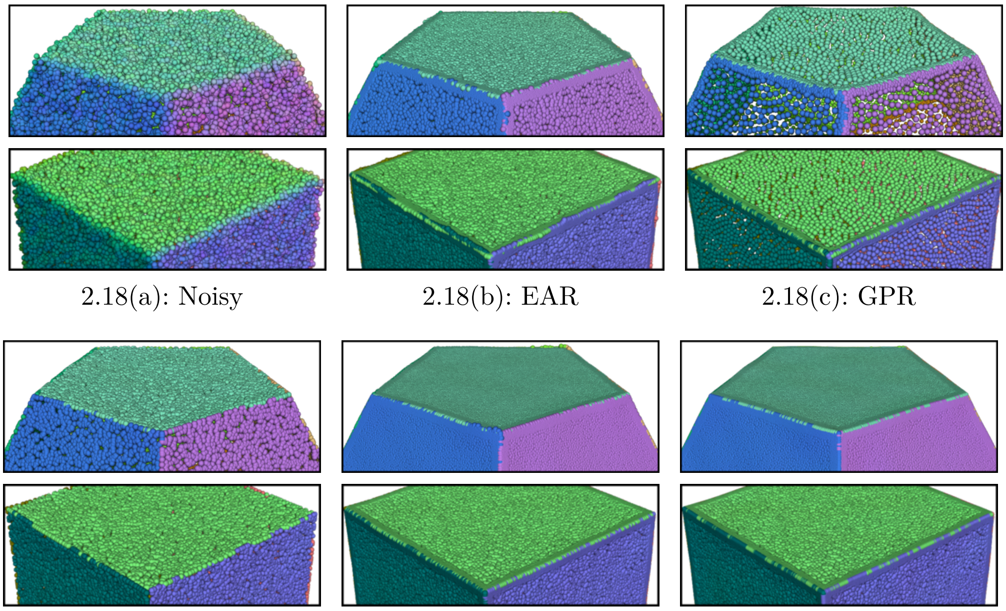
2.17(c): CNVT

2.17(d): Ours

Figure 2.17: Results obtained on the Gargoyle point cloud. For visualization purposes, we use the pre-defined triangulation. The models are rendered using flat shading.

our method with the MRPCA and the CNVT because they deal better with sharp features. We also include the result of [36] (FPF) for the Iron point cloud, provided by the authors. In the case of the MRPCA, we use the same parameters described in the paper for the Shutter and the Iron, and ($k = 30$, $\sigma = 15$, $r = 12$) and ($k = 30$, $\sigma = 15$, $r = 3$), for the Tool and the Gargoyle, respectively. In the case of the CNVT, we use the same parameters described in the paper for the Gargoyle, and the following parameters for the other point clouds. Shutter: ($\tau = 0.25$, $\rho = 0.9$, $p = 20$). Iron: ($\tau = 0.25$, $\rho = 0.9$, $p = 50$). Tool: ($\tau = 0.25$, $\rho = 0.9$, $p = 10$).

Figure 2.14 shows the denoising results on the Shutter point cloud. We can see that the MRPCA generates a clean point cloud; however, it shrinks the surface and generates noisy and blurred edges. The CNVT better preserves edges, but it accumulates several points close to them, generating gaps. Our method generates well-defined edges without generating gaps. Also,



2.18(a): Noisy

2.18(b): EAR

2.18(c): GPR

2.18(d): Ours

2.18(e): Ours + EAR ups.

2.18(f): Ours + EAR ups. + Ours

Figure 2.18: Comparison with LOP-based methods. First row of each subfigure: Twelve point cloud. Second row of each subfigure: Cube2 point cloud. The point color maps the direction of the normals.

our method can recover the flat surface in the front of the base of the object, which is ignored by the other methods. Note that both the CNVT and our method keep high-fidelity to the surfaces the input point cloud represents. Figure 2.15 shows the results on the Iron point cloud, where, as in the other figures, the point color of the noisy point cloud maps the direction of the pre-computed normals. Observe that the noisy point cloud presents some outlier normals due to the limitations of the normal estimation method. For this case, the FPF does not remove the noise properly and does not enhance the edges. The CNVT preserves some edges, but others are blurred. Also, the outlier normal points are not treated. The MRPCA generates piecewise smooth regions, even for curved regions, as shown in the first zoomed view. Although these regions enhance features, the edges are noisy, and the corners are blurred. Our method deals with flat and curved regions, preserving well-defined edges and corners. However, our method is also sensitive to the normal outliers, as shown in the second zoomed view. Figure 2.16 shows the results on the Tool point cloud, where we can highlight the importance of the precise feature detection. In the CNVT result, the edge points on the thin object's region converge to a single edge line, generating undesired gaps. Our method is capable of keeping both edge lines. The Gargoyle point cloud, shown in Figure 2.17, presents features at different scales. For this case, our method is capable to better preserve the small features such as those located close to the

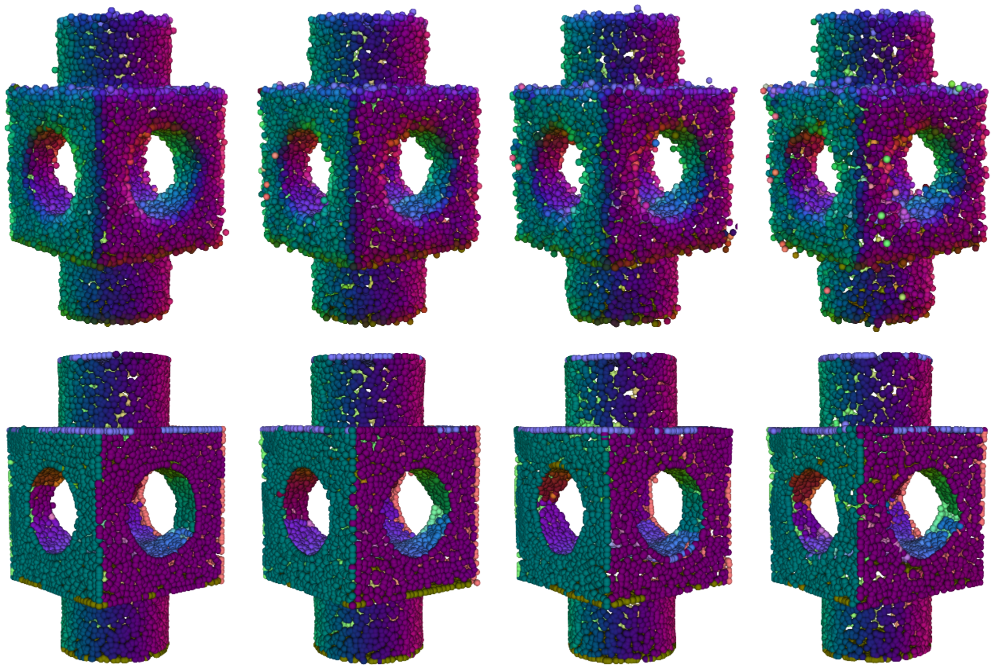

2.19(a): $\sigma = 0.4l_{ro}$ 2.19(b): $\sigma = 0.5l_{ro}$ 2.19(c): $\sigma = 0.6l_{ro}$ 2.19(d): $\sigma = 0.7l_{ro}$

Figure 2.19: The Block point cloud corrupted with different levels of noise. First row: noisy point clouds. Second row: our results. The point color maps the direction of the normals.

neck.

Figure 2.18 shows a visual comparison with the LOP-based methods EAR [40] and GPF [43] on the Twelve and Cube2 point clouds. For both methods, we use the parameters provided in [43]. Our method is applied in the following pipeline. First, we denoise the input point cloud using $(n_{ext} = 2, n_{int} = 5)$. Second, we apply EAR upsampling using the same parameters used in [43]. Third, to correct upsampling issues, we denoise the upsampled point cloud using $(n_{ext} = 1, n_{int} = 5, \delta_{cc} = 3l_{\mu}, \tau_o = 4l_{\mu})$. Observe that EAR produces noisy edges and GPF rounded edges. Furthermore, GPF expands the surface the point cloud represents and generates undesired curved regions. The combination of our method with EAR upsampling is capable of generating high-quality results.

To show the robustness of our algorithm against different levels of noise, in Figure 2.19 we show the results on the Block point cloud corrupted with different noise intensities. Our method presents consistent results, even in the presence of high noise levels.

Our method can be used to remove geometric textures if we use large enough regular neighborhoods and certain normal field smoothness. Using the Mug mesh model, we sampled $140K$ points using Poisson disk sampling (MeshLab [59] implementation). Then, we use our method to remove the

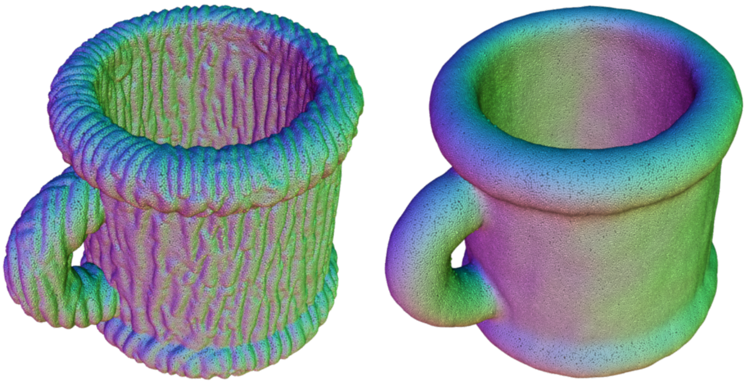


Figure 2.20: Geometric texture removal applied on the Mug point cloud. Left: point cloud with geometric texture. Right: texture removal result using our method. The point color maps the direction of the normals.

geometric texture from the sampled point cloud using the following parameters: $(n_{int} = 1, n_{ext} = 1, k = 75, r_r = 4, n_{ns} = 10, \sigma_{nn} = 0.8, \sigma_{nn} = 2l_\mu, n_{fp} = 50, \tau_o = 4l_\mu)$. Figure 2.20 shows this result.

2.9.5 Execution time

Our denoising algorithm is implemented in C++ and uses CPLEX [65] for numerical optimization. We measure the execution time on a 64-bit Intel(R) Core(TM) i7-8750H CPU 2.20GHz with 32GB RAM and Windows 10 operating system. For the main denoising test cases, Table 2.4 shows the corresponding time in seconds. We include partial timings, percentages regarding the full execution time, and number of executions for each step of the algorithm, including the preparing procedure.

The computation of the anisotropic neighborhoods is the most computationally expensive step in our pipeline because several small quadratic optimization problems have to be solved. However, since we constraint the maximum size of optimization problems to the maximum size of regular neighborhoods, i.e. k, this step presents a linear growth regarding the number of points. In the Iron point cloud case, we increased k and r_r to obtain better visual results, resulting in a considerable higher computational time compared to $k = 50$ and $r_r = 3$.

2.10 Conclusion and future work

We introduced a new method for point cloud denoising, which includes solutions for the normal estimation and feature detection problems. We also presented an extension to point clouds of the mesh-based anisotropic neighbor-

Table 2.4: Execution time in seconds of the proposed denoising algorithm on the main test cases.

$\mathcal{P}$	n	Prep. procedure			Aniso. Neigh.			Normal filtering			Feature detection			Point update			Full time
		exe	time	%	exe	time	%	exe	time	%	exe	time	%	exe	time	%	
Cube	1.9K	9	3.56	10.5	9	28.02	82.9	9	1.93	5.7	9	0.20	0.6	9	0.09	0.1	33.79
Fand.	25.9K	1	4.99	7.8	1	47.68	74.5	5	9.29	14.5	5	1.35	2.1	5	0.71	1.1	64.03
Octa.	40.2K	1	7.94	7.9	1	69.22	68.8	10	18.84	18.7	10	2.80	2.8	10	1.89	1.9	100.69
Rock.	24.1K	1	4.59	7.1	1	47.96	74.1	3	10.17	15.7	3	1.48	2.3	3	0.53	0.8	64.73
Shutt.	291.2K	2	129.55	6.9	2	1464.63	78.4	10	202.71	10.8	10	48.00	2.6	10	23.69	1.3	1868.58
Iron	161K	3	145.91	5.7	3	2055.44	80.7	15	265.00	10.4	15	61.59	2.4	15	19.84	0.8	2547.78
Tool	81.4K	1	16.60	7.8	1	162.46	76.6	5	25.99	12.2	5	4.81	2.3	5	2.34	1.1	212.19
Garg.	54.9K	1	11.41	8.4	1	108.04	79.4	3	13.57	10.0	3	1.94	1.4	3	1.04	0.8	136.00

hood computation introduced in [55]. Various experiments demonstrate that our method outperforms state-of-the-art methods in the case of point clouds with sharp features.

Combining the anisotropic neighborhood normals with a normal corrector operation allows us to avoid a strong dependence on tuning α, β, and γ, which was pointed as a critical task in [55]. Furthermore, as shown in the experiments, most of the parameters can be fixed without compromising the generation of acceptable results.

Our denoising method focuses on the preservation and enhancement of sharp features. Although we can tune the parameters to deal with smooth features, other methods like the RIMLS can work better than ours.

A maximum of 3 clusters is used for regular neighborhood clustering. This limitation does not directly affect the point updating step, but it can introduce some noise for normal correction and feature detection. As future work, we can introduce a more flexible clustering algorithm that allows us to represent a wider range of point types (e.g., non-manifold corners), involving additional considerations for feature detection.

3
Mesh denoising

3.1
Introduction

Although the point cloud is the typical raw representation of 3D acquisition methods, a triangle mesh can be generated using noisy point positions. The latter is not recommended because mesh generation methods, such as Ball Pivoting [66], expect noise-free points. The common geometry processing pipeline for triangle mesh generation includes a point cloud denoising step before the triangulation. However, mesh denoising is also an important task when the triangulation can be inferred from the acquisition method or when the mesh generation method can deal correctly with noisy data (e.g., [67]). As in the point cloud case, the mesh denoising task should preserve surface details while removing noise.

We propose an extension of the presented point cloud denoising method to deal with triangle meshes. We include a different preparing procedure, a mesh-based relaxation procedure in the point updating operation, and a face-based smoothing step. The rest of the point cloud denoising algorithm remains the same. We also focus on meshes that represent surfaces with sharp features.

The rest of the chapter is structured as follows. Section 3.2 presents related work to the mesh denoising problem. Section 3.3 explains the proposed method. In Section 3.4 we show our experiments and results. Finally, in Section 3.5 we give our conclusions.

3.2
Previous work

Based on a diffusion process, numerous anisotropic filters for mesh denoising were proposed [68, 69, 70, 71, 72] extending the idea of anisotropic diffusion of 2D grids to 3D surfaces. Hildebrant and Polthier used a prescribed mean curvature flow simplifying the diffusion process [73]. He and Schaefer proposed a method for sharp features preservation [74] using L_0 minimization.

The bilateral filter for images was an important inspiration for many anisotropic mesh filters. The adaptation of this filter was introduced by

Fleishman et al. [75] and Jones et al. [76], and then generalized by Solomon et al. [77]. Two-step-based methods, consisting of normal field filtering followed by vertex updating, were proposed adopting an anisotropic behavior [78, 79, 80]. Using a bilateral filter for normal field filtering, Zheng et al. proposed an iterative and global scheme for mesh denoising [81]. Wei et al. introduced a bilateral normal filtering using face normals and vertex normals to reach more robustness [82]. Using a guidance signal generated by computing an average normal from consistent patches, Zhang et al. proposed an extension of the joint bilateral filter [83]. Later, Li *et al.* tried to improve the consistent patch definition proposing a new metric [84].

Recently, using binary optimizations, Yadav *et al.* proposed a normal voting tensor to denoise the normal field and then update vertex positions [85]. Then, the same authors proposed an edge-weighted Laplace operator to avoid face normal flip and to be more robust to high-intensity noise [86]. They use a bilateral normal filtering with a Tukey's bi-weight function as bilateral weighting. Wei *et al.* proposed the usage of consistent neighborhoods, generated from a tensor voting analysis, to compute new vertex positions [87]. In [55], the authors proposed the computation of adaptive patches using local quadratic optimization problems. These patches are used to filter the normals, and then the vertex positions are updated. Similarly, [88] and [89] introduced new metrics for the computation of consistent patches that are used to guide normal filtering. Wang et al. proposed a feature-aware trilateral filter [90], considering the distance, feature intensity, and normal guidance.

Data-driven methods were also proposed for the mesh denoising problem. Wang et al. [91] proposed a cascade normal regression method and a new denoising dataset for training and testing. [92] introduced a two-step normal variation learning method that minimizes surface detail blurring. Li et al. [93] proposed a new deep normal filtering network that processes local patches and preserves feature regions. Other methods use convolutional neural networks in different ways for the mesh denoising problem [94, 95, 96, 97]. Although data-driven methods generally do not require parameter definition, they are strongly dependent on the training dataset.

3.3 Extension to triangle meshes

In this section, we explain how to modify the proposed point cloud denoising algorithm in order to process triangle meshes. We assume that the input is a noisy mesh with consistently oriented faces, and the output is the same mesh with modified vertex positions, such that the new positions minimize the noise. Let us denote a triangle mesh as the tuple $(\mathcal{V}, \mathcal{F}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_m\}$ is a set of m vertices, $\mathcal{F} = \{f_1, \ldots, f_n\}$ is a set of n faces, i.e. triangles, and $\mathcal{E} = \{e_1, \ldots, e_l\}$ is a set of l undirected edges. The vertex positions are represented by the set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, where $\mathbf{x}_i$ is the position of the vertex v_i. Each face is represented as a tuple of vertices (v_i, v_j, v_k) whose corresponding normal $\mathbf{n}^f$ can be computed as follows: $\mathbf{n}^f = (\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)$. So, we can represent all face-based normals as the set $N^f = \{\mathbf{n}^f_1, \ldots, \mathbf{n}^f_n\}$, where $\mathbf{n}^f_i$ is the normal of f_i. Also, consider a vector $\mathbf{a}^f = \{a^f_1, \ldots, a^f_n\}^T$, where a^f_i denotes the area of f_i, and a set of face centroids $C^f = \{\mathbf{c}^f_1, \ldots, \mathbf{c}^f_n\}$, where $\mathbf{c}^f_i$ is the centroid of f_i. Each edge describes the connection between two vertices and is represented as the tuple (v_i, v_j). Thus, the mesh denoising problem can be interpreted as the computation of new values for each $\mathbf{x}_i \in X$.

We introduce three main modifications to the point cloud denoising algorithm. First, we replace the preparing procedure with a new mesh-based process that considers mesh topology to compute local neighborhoods and geometric measurements. Second, we include an additional point updating operation for mesh relaxation. This operation allows us to minimize mesh artifacts. Third, we include a mesh-based normal bilateral filtering step in the iterative scheme. The rest of the algorithm remains the same, as shown in Algorithm 2, where the red statements denote the introduced modifications.

Since the algorithm's operations require a set of points as input, we use the mesh vertices for this. For example, feature classification is applied to every $v_i \in \mathcal{V}$, such that each vertex is classified as flat, edge, or corner. A detailed description of each modification is presented in the following subsections.

3.3.1 Mesh-based preparing procedure

In the point cloud denoising algorithm, the preparing procedure aims to compute a set of point neighborhoods with different sizes. The intuition behind this computation is to define local topological structures on the unstructured set of points based on Euclidean proximity. However, in the mesh case, the topology can be inferred from $\mathcal{F}$ and $\mathcal{E}$. For this reason, we redefine the neighborhoods $\mathscr{N}_s$, $\mathscr{N}_r$, and $\mathscr{N}_b$, using this information.

Let us define the standard vertex-based k-ring $\mathcal{R}_k$ for a vertex $v_i \in \mathcal{V}$ as follows:

$$\begin{aligned} &\mathcal{R}_k(v_i) = \left\{ v_j \in \mathcal{V} \,\middle|\, (v_j, v_s) \in \mathcal{E} \wedge v_s \in \mathcal{R}_{k-1}(v_i) \wedge v_j \notin \bigcup_{l=0}^{k-1} \mathcal{R}_l(v_i) \right\}, \\ &\mathcal{R}_0(v_i) = \{v_i\}. \end{aligned} \tag{3-1}$$

Algorithm 2 Mesh denoising

```
procedure DENOISE($\mathcal{V}$, $\mathcal{F}$)
    for $it_{ext} \leftarrow 1$ to $n_{ext}$ do
        bilateralNormalFiltering($\mathcal{V}$, $\mathcal{F}$, $\sigma^f_{ns}$, $\sigma^f_{nn}$, $n^f_{ns}$, $n^f_{vu}$)
        meshPreparingProcedure($\mathcal{V}$, $\mathcal{F}$, $\epsilon_d$, $k$, $r_s$, $r_r$, $r_b$)
        anisotropicNeighborhoods($\mathcal{V}$, $\alpha$, $\beta$, $\gamma$, $a_0$)
        for $it_{int} \leftarrow 1$ to $n_{int}$ do
            if $it_{int} \neq 1$ then
                bilateralNormalFiltering($\mathcal{V}$, $\mathcal{F}$, $\sigma_{ns}$, $\sigma_{nn}$)
            anisotropicNeighborhoodNormals($\mathcal{V}$, $\tau_u$)
            for $it_{ns} \leftarrow 1$ to $n_{ns}$ do
                normalSmoothing($\mathcal{V}$, $\sigma^f_{ns}$, $\sigma^f_{nn}$, $n^f_{ns}$, $n^f_{vu}$)
            roughFeatureClassification($\mathcal{V}$, $\tau_n$, $\tau_{ta}$)
            neighborhoodClustering($\mathcal{V}$) # without including $p_i$
            for $it_{nc} \leftarrow 1$ to $n_{nc}$ do
                normalCorrection($\mathcal{V}$, $\epsilon_{mn}$)
            roughFeatureClassification($\mathcal{V}$, $\tau_n$, $\tau_{ta}$)
            neighborhoodClustering($\mathcal{V}$) # including $p_i$
            pointConvexityAnalysis($\mathcal{V}$, $\delta_{cc}$, $\epsilon_{cc}$)
            sharpFeatureDetection($\mathcal{V}$, $\theta$)
            for $it_{fp} \leftarrow 1$ to $n_{fp}$ do
                flatPointUpdate($\mathcal{V}$, $\tau_o$, $\sigma_{ps}$, $\sigma_{pn}$, $\upsilon_f$)
            cornerAndEdgePointUpdate($\mathcal{V}$, $\tau_o$, $\upsilon_e$, $\upsilon_c$)
```

The k-rings are adjacency data structures useful to navigate the mesh and to define topology-based neighborhoods, which are widely used in mesh filtering algorithms (e.g., [75, 74]). Figure 3.1 shows an example of the k-rings computed on a triangular mesh. Similarly to other mesh filtering algorithms, we define $\mathcal{N}_s$, $\mathcal{N}_r$ and $\mathcal{N}_b$ as follows:

$$\begin{aligned}\mathcal{N}_s(v_i) &= \mathcal{R}_0(v_i) \cup \mathcal{R}_1(v_i),\\ \mathcal{N}_r(v_i) &= \mathcal{R}_0(v_i) \cup \mathcal{R}_1(v_i) \cup \mathcal{R}_2(v_i),\\ \mathcal{N}_b(v_i) &= \mathcal{R}_0(v_i) \cup \mathcal{R}_1(v_i) \cup \mathcal{R}_2(v_i) \cup \mathcal{R}_3(v_i).\end{aligned} \tag{3-2}$$

The number of elements of the regular neighborhood $\mathcal{N}_r$ defines the complexity of the optimization problem used to compute the anisotropic neighborhood. As in the point cloud case, if the number of elements of $\mathcal{N}_r$ is higher than a parameter k, we pick the k closest vertices in $\mathcal{N}_r$.

It is essential to define the average distance l_λ, because it is used to define several parameters of the algorithm. Thus, we define l_λ as the average edge length considering all the elements in $\mathcal{E}$. Then, we should define per-point areas, i.e. per-vertex areas. So, we adopt the barycentric cell-based area for each v_i, considering the triangulation $\mathcal{F}$. Finally, the *regular normal* for each v_i is computed by averaging the normals of the faces that v_i shares.

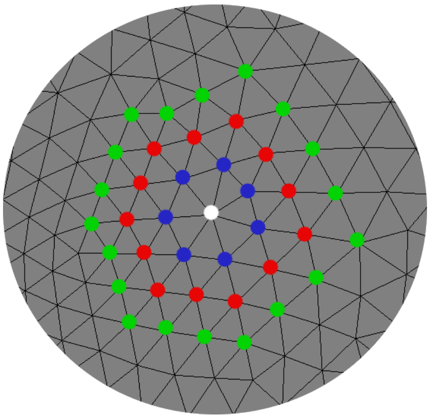


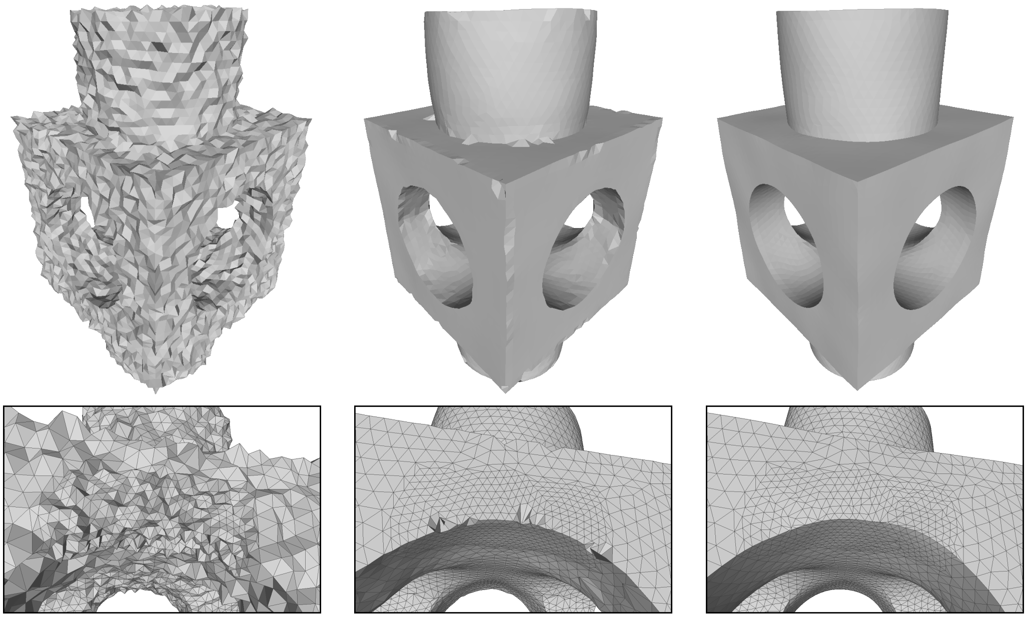
Figure 3.1: k-rings example on a triangular mesh. White: evaluated vertex (v_i) or 0-ring ($\mathcal{R}_0(v_i)$). Blue: 1-ring ($\mathcal{R}_1(v_i)$). Red: 2-ring ($\mathcal{R}_2(v_i)$). Green: 3-ring ($\mathcal{R}_3(v_i)$).

These structures and measurements are used in the next steps of the algorithm, where the anisotropic neighborhoods computation, normal filtering, and feature detection steps are the same presented in the point cloud denoising algorithm.

3.3.2 Mesh-based relaxation for vertex position update

The point updating step, included in the point cloud denoising algorithm, projects the point positions onto the underlying surface the estimated normal field represents. Since no connectivity information is considered, if we apply the same point updating step to the mesh vertices, we can generate several mesh artifacts. Figure 3.2 shows an example of some of the artifacts that are generated when applying the point cloud-based algorithm. For this reason, we introduce additional point updating operations that alleviate these problems.

In the case of flat point updating, we add a Laplacian-based relaxation operation which is applied after the point cloud-based updating operation. The intuition of this operation is to smooth vertex positions based on the mesh connectivity and to correct those vertices that are located at distant and unexpected positions regarding their immediate neighbors. The left subfigure in Figure 3.3 shows a typical case that requires correction, where the white point represents v_i, the blue points represent $\mathcal{R}_1(v_i)$, and all of them lie on a flat region. Note that the corresponding triangulation generates irregular triangles because v_i is far away from the centroid of $\mathcal{R}_1(v_i)$. Thus, in addition to the flat point updating operation introduced in the point cloud-based algorithm, we apply the following operation:



3.2(a): Noisy 3.2(b): Point cloud-based 3.2(c): Mesh-based

Figure 3.2: Results obtained on the Block.

$$\tilde{\mathbf{x}}_i = \begin{cases} \mathbf{x}_i + \upsilon_f(\mathbf{c}_i^v - \mathbf{x}_i) & r_i < 0.7 \\ \mathbf{x}_i + 0.025\upsilon_f(\mathbf{c}_i^v - \mathbf{x}_i) & \text{otherwise} \end{cases}, \tag{3-3}$$

where υ_f is the same step size used for the point cloud-based operation and $\mathbf{c}_i^v$ is a weighted centroid of $\mathcal{R}_1(v_i)$ defined as follows:

$$\mathbf{c}_i^v = \frac{1}{W_v(v_i)} \sum_{v_j \in \mathcal{R}_1(v_i)} w_{ij}\mathbf{x}_j, \tag{3-4}$$

where the weights w_{ij} and the normalization factor $W_v(v_i)$ are the same introduced in the point cloud-based operation. Thus, the weighted centroid computation tries to prioritize those points that correspond to the flat region v_i represents. The term r_i measures the ratio between the average distance from $\mathbf{c}_i^v$ to every $v_j \in \mathcal{R}_1(v_i)$ and the average distance from v_i to every $v_j \in \mathcal{R}_1(v_i)$, and is defined as follows:

$$r_i = \frac{\dfrac{1}{|\mathcal{R}_1(v_i)|} \displaystyle\sum_{v_j \in \mathcal{R}_1(v_i)} \|\mathbf{x}_j - \mathbf{c}_i^v\|}{\dfrac{1}{|\mathcal{R}_1(v_i)|} \displaystyle\sum_{v_j \in \mathcal{R}_1(v_i)} \|\mathbf{x}_j - \mathbf{x}_i\|}. \tag{3-5}$$

If this ratio is lower than 0.7, we assume that v_i is distant from the center of $\mathcal{R}_1(v_i)$ and possibly generating an undesired mesh artifact. When the latter occurs, we apply a high displacement amount to v_i, i.e. υ_f, in the direction to $\mathbf{c}_i^v$, which can be interpreted as a weighted Laplacian smoothing step. Otherwise, we apply a small displacement, i.e. $0.025\upsilon_f$, in the same direction.

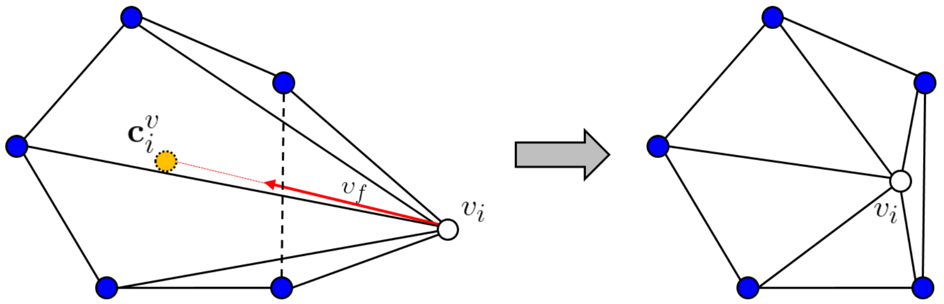


Figure 3.3: Flat point correction using larger steps for Laplacian relaxation.

This new operation for flat points is applied in the same iterative scheme of the point cloud-based operation and allows us to minimize mesh artifacts and obtain more regular triangulations.

In the case of edge point updating, we introduce a similar Laplacian smoothing operation but constraining the displacement to the corresponding edge direction. So, we add the following operation to be applied after the point cloud-based operation:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + 0.025 \upsilon_e (\mathbf{c}_i^\pi - \mathbf{x}_i) \tag{3-6}$$

where υ_e is the same step size used for the point cloud-based operation and $\mathbf{c}_i^\pi$ is the uniform centroid of $\mathcal{R}_1(v_i)$ projected on the corresponding edge line, computed as follows:

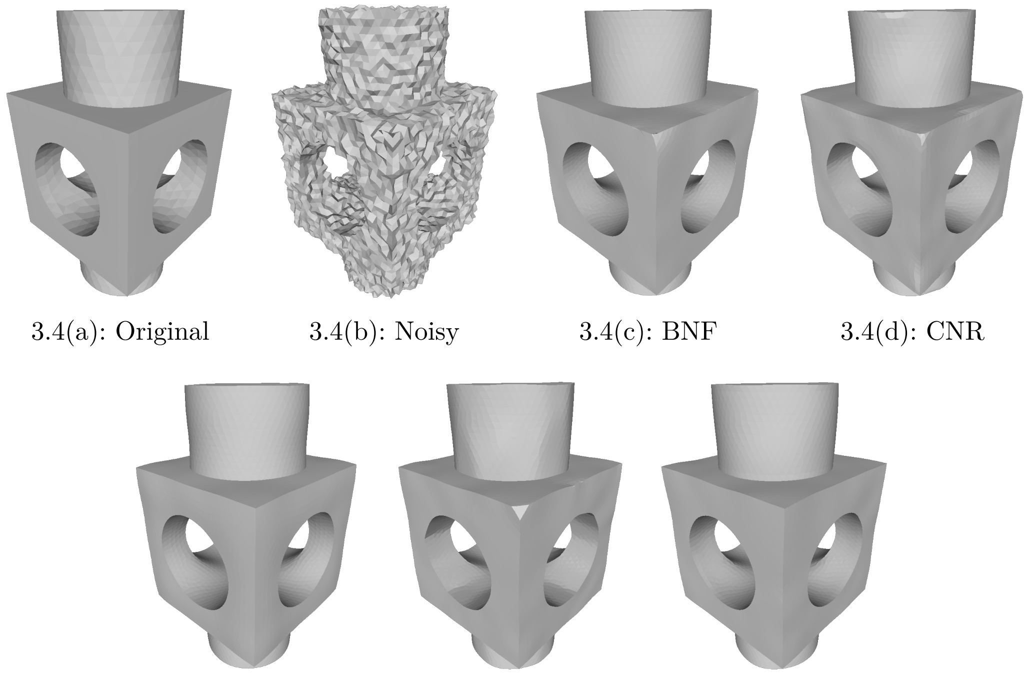
$$\mathbf{c}_i^\pi = \mathbf{x}_i + \left\langle \left(\left(\frac{1}{|\mathcal{R}_1(v_i)|} \sum_{v_j \in \mathcal{R}_1(v_i)} \mathbf{x}_j \right) - \mathbf{x}_i \right), \mathbf{e}_i \right\rangle \mathbf{e}_i, \tag{3-7}$$

where $\mathbf{e}_i$ is the edge direction computed in the sharp feature detection step and $|\mathcal{R}_1(v_i)|$ denotes the number of elements of $\mathcal{R}_1(v_i)$.

For corner points, we keep the same point cloud-based operation because these points are very sparse, and their surrounding neighbors can adapt the triangulation to them.

3.3.3 Mesh smoothing using the face-based bilateral normal filtering

Face normal filtering methods have shown important success on the mesh denoising problem because face normals can better represent the local geometry than vertex positions. For this reason, we introduce a face normal-based bilateral filtering operation, i.e. [81], in the denoising pipeline. This operation is applied at the beginning of each external iteration or at the beginning of the internal iteration when it is not the first one (See Algorithm 2). The idea is to combine the vertex-based denoising method that tends to



3.4(a): Original 3.4(b): Noisy 3.4(c): BNF 3.4(d): CNR

3.4(e): RHF 3.4(f): AP 3.4(g): Ours

Figure 3.4: Results obtained on the Block ($\sigma = 0.3 l_\lambda$).

better preserve sharp feature regions with a face-based denoising method that smooths the surface without blurring these regions.

The bilateral filtering method introduced in [81] consists of two steps. First, the normal field described by the face normals is filtered as follows:

$$\tilde{\mathbf{n}}_i^f = \frac{1}{W_n^f(f_i)} \sum_{f_j \in \mathscr{N}_{fv}(f_i)} a_j^f K_{ns}^f \left(\left\| \mathbf{c}_i^f - \mathbf{c}_j^f \right\| \right) K_{nn}^f \left(\left\| \mathbf{n}_i^f - \mathbf{n}_j^f \right\| \right) \mathbf{n}_j^f, \qquad (3\text{-}8)$$

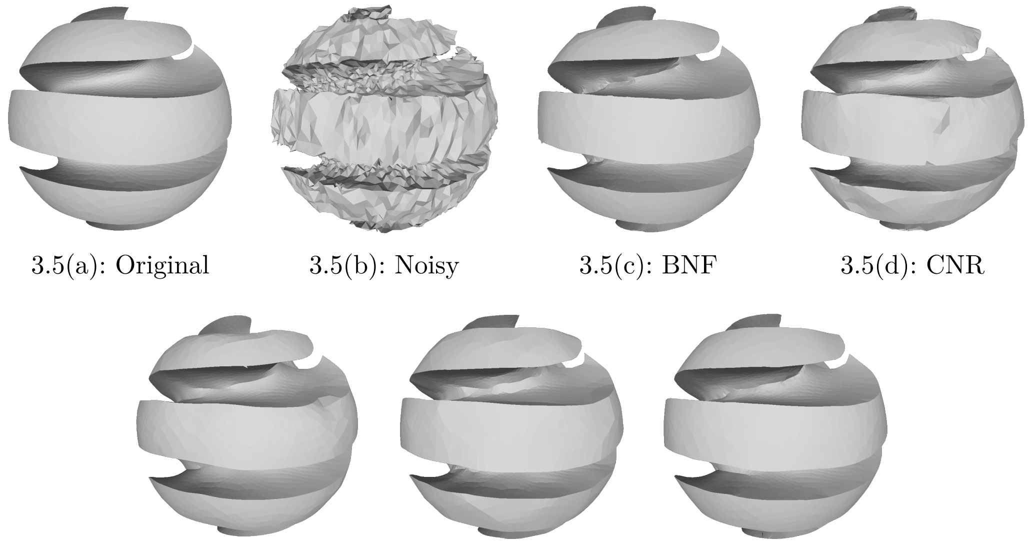
where $\tilde{\mathbf{n}}_i^f$ is the new face normal, $\mathscr{N}_{fv}(f_i)$ is the set of neighboring faces that share a vertex of f_i, K_{ns}^f and K_{nn}^f are Gaussian kernel functions with standard deviations σ_{ns}^f and σ_{nn}^f, respectively, and $W_n^f(f_i)$ is the corresponding normalization factor. This operation is applied iteratively, where n_{ns}^f defines the number of iterations.

Second, the vertex positions are updated to fit the filtered normals [78, 80] as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \frac{1}{|\mathscr{N}_f(v_i)|} \sum_{f_j \in \mathscr{N}_f(v_i)} \left\langle \left(\mathbf{c}_j^f - \mathbf{x}_i \right), \mathbf{n}_j^f \right\rangle \mathbf{n}_j^f, \qquad (3\text{-}9)$$

where $\tilde{\mathbf{x}}_i$ is the new position, $\mathscr{N}_f(v_i)$ denotes the set of faces shared by v_i, and $|\mathscr{N}_f(v_i)|$ denotes the number of elements in $\mathscr{N}_f(v_i)$. This operation is also applied iteratively, where n_{vu}^f defines the number of iterations.

We fix the parameters of the bilateral filtering as follows: $\sigma_{ns}^f = \sigma_{ns}$, $\sigma_{nn}^f = \sigma_{nn}$, and $n_{ns}^f = n_{vu}^f = 4$. Note that we use just a few iterations for both operations, since they are used as an additional smoothing procedure.



3.5(a): Original 3.5(b): Noisy 3.5(c): BNF 3.5(d): CNR

3.5(e): RHF 3.5(f): AP 3.5(g): Ours

Figure 3.5: Results obtained on the SharpSphere ($\sigma = 0.3l_\lambda$).

Figure 3.2 shows the difference between the point cloud-based algorithm and the mesh-based algorithm.

3.4 Results

We compare our mesh denoising method numerically and visually against state-of-the-art methods on a subset of the test dataset introduced in [91]. The selected subset consists of noise-free meshes with challenging sharp and smooth features, named Block, Carter, ChineseLion, Gargoyle, Joint, Merlion, Nicolo, Pyramid, SharpSphere, and SmoothFeature. We corrupt these models with Gaussian noise following the normal direction, using $\sigma = 0.3l_\lambda$ and $\sigma = 0.5l_\lambda$. The methods used for comparison are [81], [91], [86], and [55], named as BNF, CNR, RHF, and AP, respectively.

For the BNF and the RHF methods, the parameters were tuned following the recomendations of the authors or using the same values if the model was included in the corresponding work. Table 3.1 shows the selected parameters for these methods, using the formats (σ_e,σ_s,normal iterations,vertex iterations) and (σ_s,λ_I,p) for BNF and RHF, respectively. For the CNR method, we use the corresponding pre-trained model provided by the authors. For the AP method, as suggested by the authors, we pre-process the noisy models using the BNF method with (1.0, 0.35, 5, 5) and (1.0, 0.5, 5, 5) for $\sigma = 0.3l_\lambda$ and $\sigma = 0.5l_\lambda$, respectively. Then, based on the authors' experiments, we use the following parameters for all cases: ($\alpha = 1.0$, $\beta = 1.0$, $\gamma = 0.3$, $\delta = 30$, $n_{var} = 20$, $n_e = 3$,$n_p = 5$,$n_b = 2$,$n_p = 10$). For the proposed method, we fix $r_r = 2$,

Table 3.1: Mesh denoising parameters.

Model	**BNF**	**RHF**	**Ours**
Block ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(1.0,0.2,150)	(3,3,0.35,0.35)
Block ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(1.0,0.2,200)	(3,6,0.35,0.35)
Carter ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(0.55,0.2,150)	(3,3,0.35,0.35)
Carter ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(0.55,0.2,200)	(3,6,0.35,0.35)
ChineseLion ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,20,20)	(0.4,0.3,100)	(2,2,0.35,0.35)
ChineseLion ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,40,50)	(0.4,0.3,150)	(2,4,0.35,0.35)
Gargoyle ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,20,20)	(0.4,0.3,100)	(2,2,0.25,0.25)
Gargoyle ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,40,50)	(0.4,0.3,150)	(2,4,0.25,0.25)
Joint ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(1.0,0.2,150)	(3,3,0.35,0.35)
Joint ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(1.0,0.2,200)	(3,6,0.35,0.35)
Merlion ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,20,20)	(0.4,0.3,100)	(2,2,0.25,0.25)
Merlion ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,40,50)	(0.4,0.3,150)	(2,4,0.25,0.25)
Nicolo ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,20,20)	(0.4,0.3,100)	(2,2,0.35,0.35)
Nicolo ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,40,50)	(0.4,0.3,150)	(2,4,0.35,0.35)
Pyramid ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(0.55,0.2,150)	(3,3,0.35,0.35)
Pyramid ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(0.55,0.2,200)	(3,6,0.35,0.35)
SharpSphere ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(0.55,0.2,150)	(3,3,0.35,0.35)
SharpSphere ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(0.55,0.2,200)	(3,6,0.35,0.35)
SmoothFeature ($\sigma = 0.3l_\lambda$)	(l_λ,0.35,30,30)	(0.55,0.2,150)	(3,3,0.35,0.35)
SmoothFeature ($\sigma = 0.5l_\lambda$)	(l_λ,0.35,50,100)	(0.55,0.2,200)	(3,6,0.35,0.35)

$n_{ns} = 5$, ϵ_{mn}, $\delta_{cc} = 2l_\lambda$, and $n_{fp} = 1$. Table 3.1 shows the tuned parameters in the following format: (n_{ext}, n_{int}, τ_n, σ_n). For all cases, if the parameter is not specified, the default value is used. Please refer to the corresponding work for a better understanding of each parameter.

For numerical evaluation, we use two metrics: the L^2 vertex-based positional error, denoted as D_v, and the mean square angular error, denoted as $MSAE$. Both metrics are explained in [86]. Table 3.2 shows the numerical results obtained on all the test cases, where the minimum values are colored in blue and the second minimum values are colored in green.

CNR seems to be superlative when the noise is based on $\sigma = 0.3l_\lambda$; however, when we analyze the results visually, we can perceive some blurred and noisy regions, as shown in Figures 3.4 and 3.5. Furthermore, from Table 3.2, we can see that CNR suffers when the noise is based on $\sigma = 0.5l_\lambda$.

From the results on the Block model corrupted with noise based on $\sigma = 0.3l_\lambda$ (Figure 3.4), we can also see that AP generates over flattened regions. The latter is evident on the Carter model corrupted with noise based on $\sigma = 0.3l_\lambda$, which is shown in Figure 3.6. Also, in this example, AP generates several mesh artifacts close to the sharp regions.

From Figures 3.4, 3.5, and 3.6, we can see that our method generates good results when processing models that present sharp features and curved regions. The results of RHF in Figure 3.4 are quite similar to ours, however, it tends to shrink the surfaces as shown by the corresponding D_v values. Also, we can see that RHF deforms and blurs some edge regions in Figures 3.5 and 3.6.

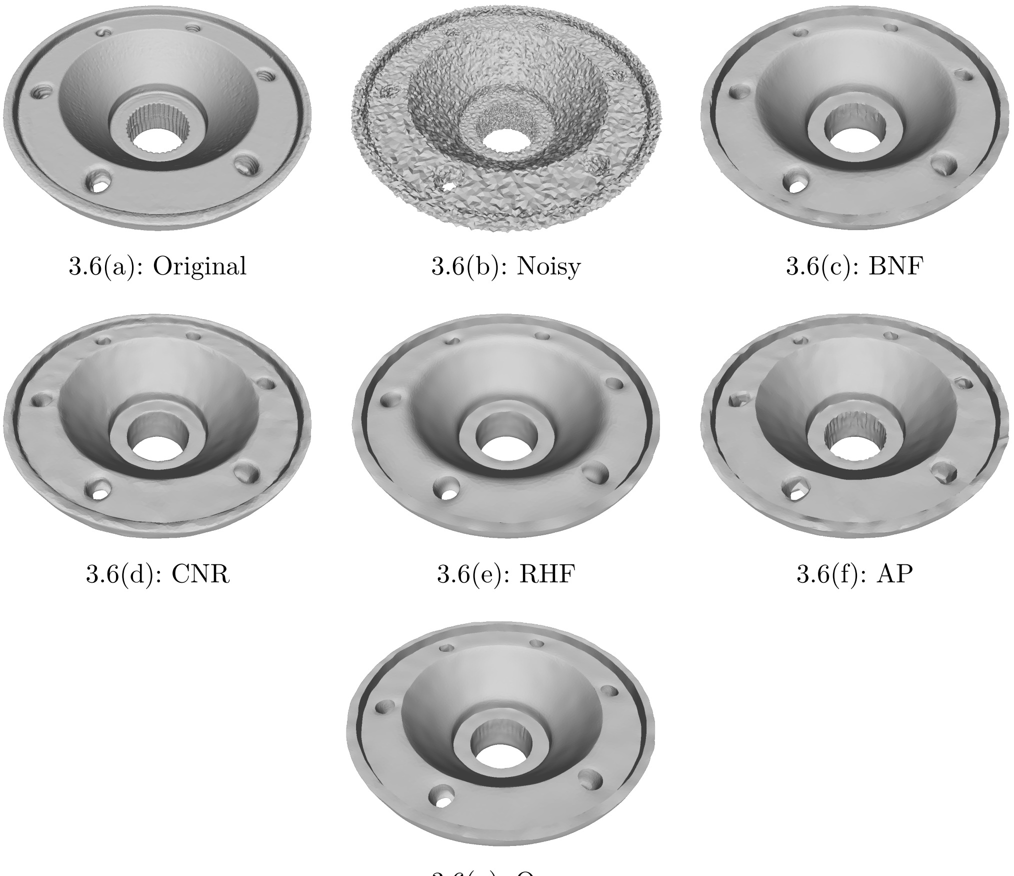
Our method is also capable of dealing with models that present smooth

Table 3.2: Mesh denoising results.

Model	Metric	BNF	CNR	RHF	AP	Ours
Block ($\sigma = 0.3l_\lambda$)	D_v	0.05186	0.04630	0.07042	0.04830	0.04928
	$MSAE$	3.36225	2.91999	3.62524	3.31965	2.72166
Block ($\sigma = 0.5l_\lambda$)	D_v	0.10806	0.10080	0.08810	0.07858	0.08241
	$MSAE$	9.03776	6.91090	4.57410	5.24198	4.26168
Carter ($\sigma = 0.3l_\lambda$)	D_v	0.12962	0.08525	0.11993	0.08766	0.09535
	$MSAE$	11.14230	8.86946	11.14550	10.25960	11.13020
Carter ($\sigma = 0.5l_\lambda$)	D_v	0.22638	0.15646	0.13849	0.12082	0.13109
	$MSAE$	17.17520	12.68230	12.19600	12.52380	13.26770
ChineseLion ($\sigma = 0.3l_\lambda$)	D_v	0.14904	0.10681	0.15304	0.12151	0.10964
	$MSAE$	11.2304	9.00213	12.17230	11.5156	10.4640
ChineseLion ($\sigma = 0.5l_\lambda$)	D_v	0.24985	0.18075	0.18140	0.16701	0.15417
	$MSAE$	18.76820	12.56110	13.74650	14.20370	14.17110
Gargoyle ($\sigma = 0.3l_\lambda$)	D_v	0.325205	0.13522	0.468576	0.11407	0.151817
	$MSAE$	11.8491	8.80091	13.2901	11.0058	11.9242
Gargoyle ($\sigma = 0.5l_\lambda$)	D_v	0.63490	0.23669	0.56575	0.14338	0.20111
	$MSAE$	20.12560	13.67430	15.48030	14.05610	16.70750
Joint ($\sigma = 0.3l_\lambda$)	D_v	0.00082	0.00100	0.00112	0.00107	0.00111
	$MSAE$	2.05617	2.23018	2.15223	2.46076	2.02759
Joint ($\sigma = 0.5l_\lambda$)	D_v	0.00155	0.00206	0.00150	0.00162	0.00186
	$MSAE$	3.71325	4.74553	2.69485	3.43932	2.92367
Merlion ($\sigma = 0.3l_\lambda$)	D_v	0.00033	0.00018	0.00035	0.00022	0.00022
	$MSAE$	6.77952	4.73488	6.93717	5.79941	6.55522
Merlion ($\sigma = 0.5l_\lambda$)	D_v	0.00054	0.00029	0.00040	0.00030	0.00030
	$MSAE$	12.05860	6.57421	7.66620	7.23915	9.50065
Nicolo ($\sigma = 0.3l_\lambda$)	D_v	0.00334	0.00267	0.00324	0.00304	0.00274
	$MSAE$	5.80729	4.9861	5.8307	6.12768	5.4733
Nicolo ($\sigma = 0.5l_\lambda$)	D_v	0.00584	0.00452	0.00412	0.00444	0.00382
	$MSAE$	8.91441	7.11122	6.56079	7.72683	7.08066
Pyramid ($\sigma = 0.3l_\lambda$)	D_v	0.00184	0.00189	0.00180	0.00179	0.00176
	$MSAE$	1.48455	1.91501	1.43766	1.54871	1.34818
Pyramid ($\sigma = 0.5l_\lambda$)	D_v	0.00366	0.00312	0.00252	0.00262	0.00247
	$MSAE$	3.27803	4.16106	2.04269	2.40448	2.15468
SharpSphere ($\sigma = 0.3l_\lambda$)	D_v	0.07738	0.05473	0.08415	0.04941	0.05283
	$MSAE$	13.4105	8.63913	9.10155	9.4108	12.0378
SharpSphere ($\sigma = 0.5l_\lambda$)	D_v	0.16014	0.11240	0.09433	0.07861	0.09462
	$MSAE$	23.36460	14.37160	11.52730	13.74580	17.34710
SmoothFeature ($\sigma = 0.3l_\lambda$)	D_v	0.00407	0.00396	0.00533	0.00426	0.00410
	$MSAE$	1.75039	1.60640	2.09626	2.01176	1.65717
SmoothFeature ($\sigma = 0.5l_\lambda$)	D_v	0.00995	0.00623	0.00644	0.00622	0.00597
	$MSAE$	6.35214	3.40875	2.66150	3.42862	3.09433

features, as shown in Figure 3.7. In this case, AP over-flattens the surface, while BNF and RHF blur some features. The CNR output presents higher fidelity to the original model than ours because we assume certain feature sharpness (See the left eye). However, the choice of one of these behaviors depends on the application. For example, to generate a simplified version of the Carter model shown in Figure 3.6, we might prefer a denoised surface with sharp edges instead of a denoised surface with chamfered edges.

By just increasing the number of internal iterations, our method is capable of processing meshes with a higher noise level. Figures 3.8 and 3.9 show the results on the Pyramid and the Joint models corrupted with noise based on $\sigma = 0.5l_\lambda$. Our method is robust against different levels of noise when considering the appropriate number of iterations. The CNR results present several noisy regions in these examples because the method is dependent on



3.6(a): Original

3.6(b): Noisy

3.6(c): BNF

3.6(d): CNR

3.6(e): RHF

3.6(f): AP

3.6(g): Ours

Figure 3.6: Results obtained on the Carter ($\sigma = 0.3l_\lambda$).

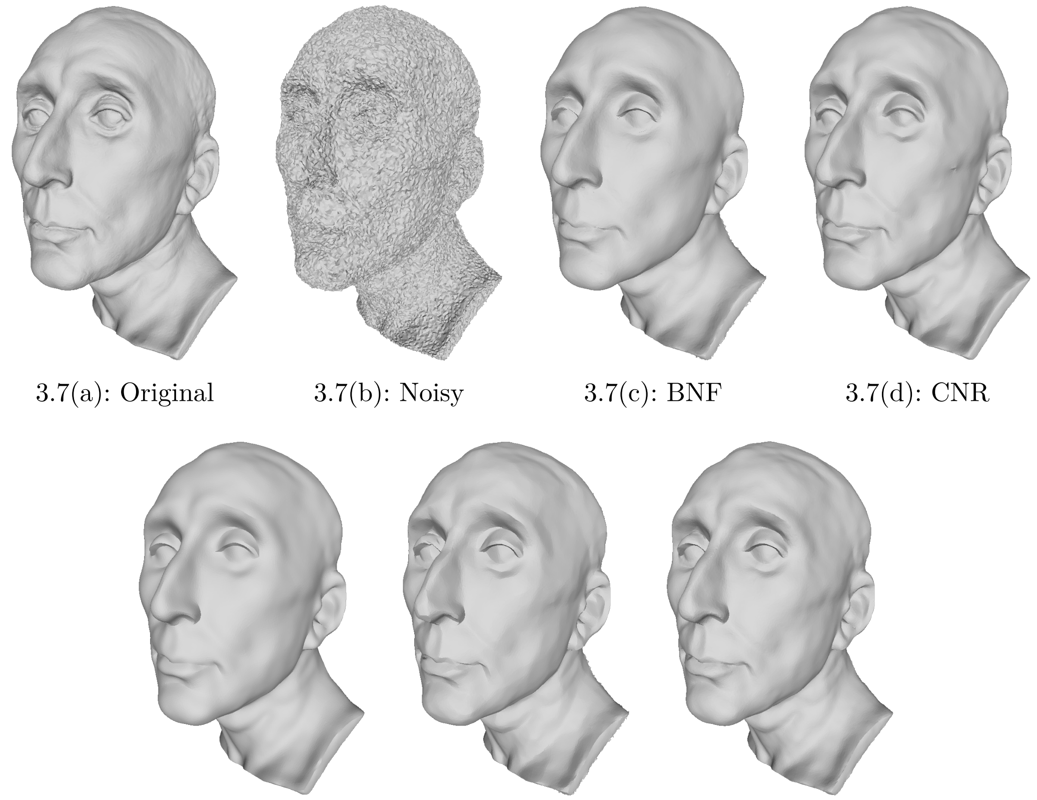
the training dataset.

Figure 3.10 shows the results on the Merlion model corrupted with Gaussian noise based on $\sigma = 0.5l_\lambda$. Note that the model presents several details which are blurred by the BNF and RHF methods. CNR preserves most of these details, but it introduces noise, similar to the results shown in Figures 3.8 and 3.9. In this case, our method and the AP can deal better with detail preservation while removing noise.

3.5 Conclusion and future work

We introduced a mesh denoising algorithm that focuses on the preservation of sharp features. This algorithm is an extension of the point cloud-based denoising algorithm presented in the previous chapter, where the proposed modifications take advantage of the explicit topology defined by a mesh. The numerical and visual results show that our algorithm is competitive when compared to the selected algorithms.

Although we introduced some operations to minimize mesh artifacts, our



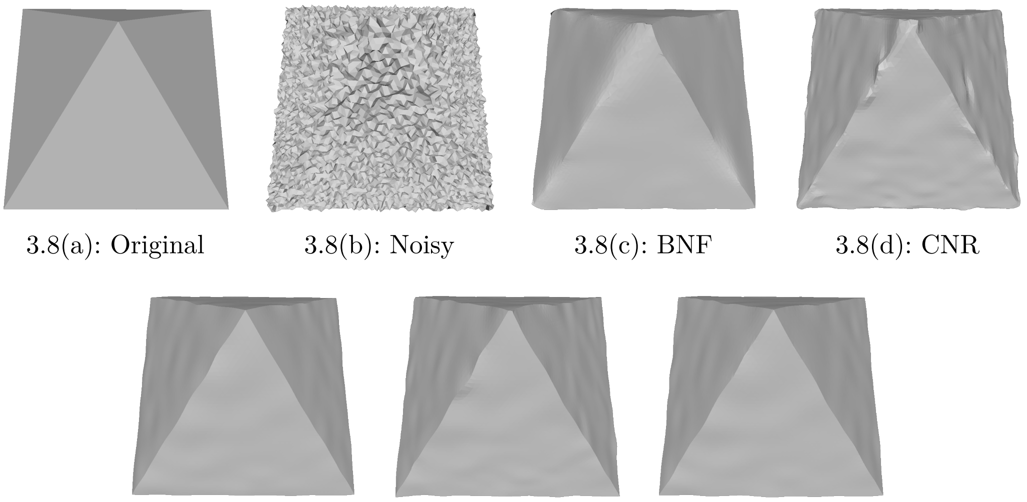
3.7(a): Original 3.7(b): Noisy 3.7(c): BNF 3.7(d): CNR

3.7(e): RHF 3.7(f): AP 3.7(g): Ours

Figure 3.7: Results obtained on the Nicolo ($\sigma = 0.3 l_\lambda$).

method is prone to them if the number of iterations is insufficient. If we apply too many iterations, we can over-smooth the surface. So, it is important to take care of the number of iterations in addition to the thresholding parameters.

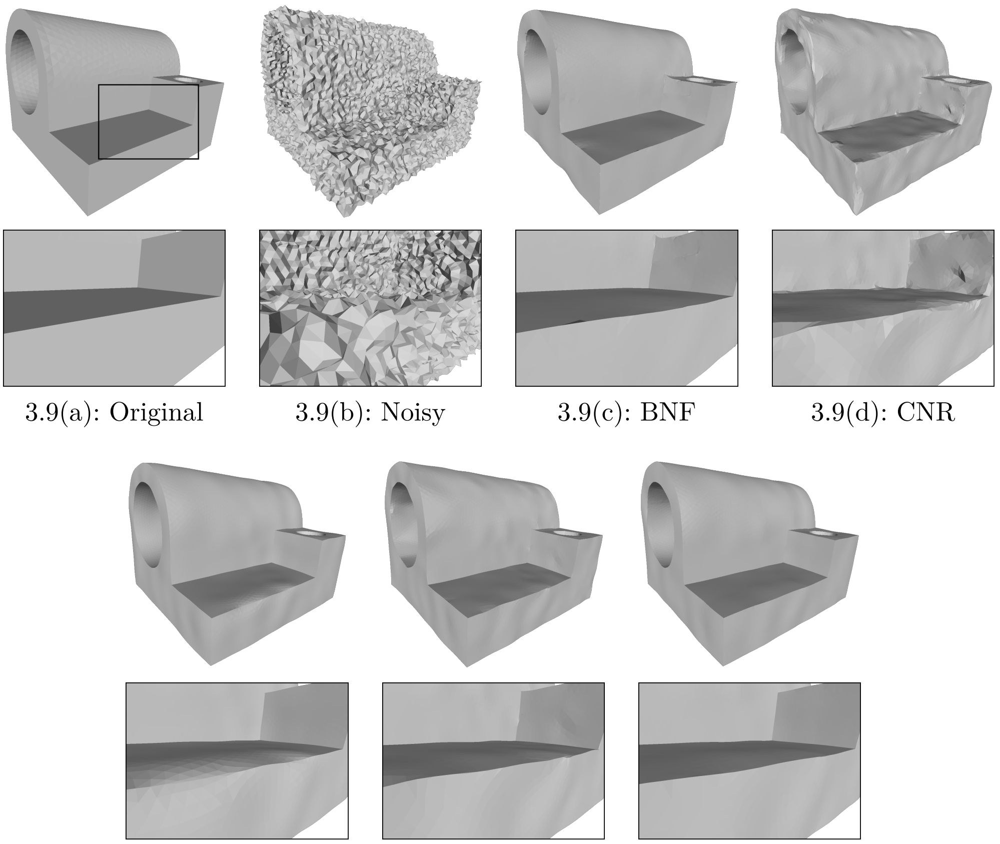
Most of the point cloud-based denoising pipeline remains the same. However, as future work, we can consider mesh-based operations for the normal filtering and sharp feature detection steps.



3.8(a): Original 3.8(b): Noisy 3.8(c): BNF 3.8(d): CNR

3.8(e): RHF 3.8(f): AP 3.8(g): Ours

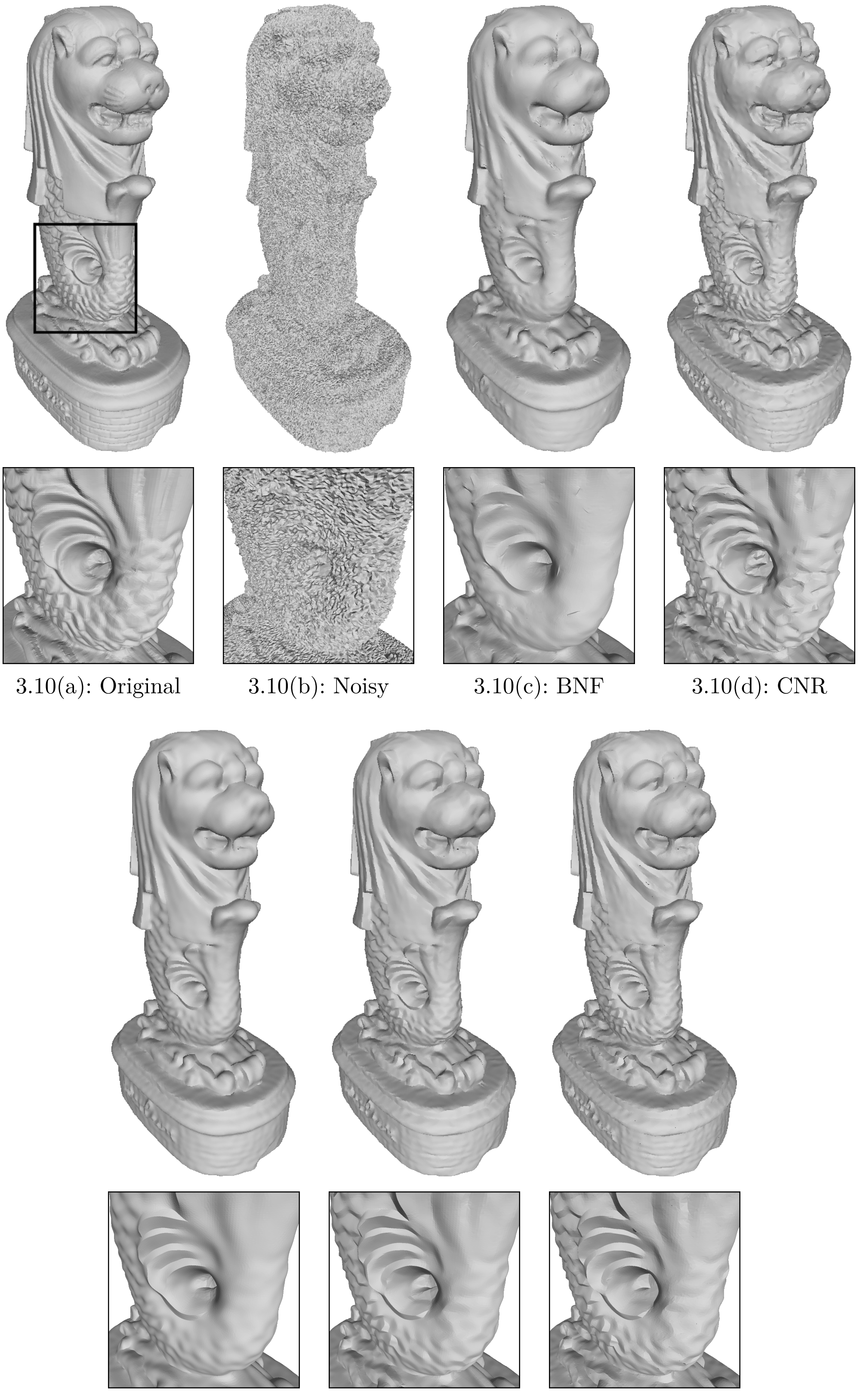
Figure 3.8: Results obtained on the Pyramid ($\sigma = 0.5l_\lambda$).



3.9(a): Original 3.9(b): Noisy 3.9(c): BNF 3.9(d): CNR

3.9(e): RHF 3.9(f): AP 3.9(g): Ours

Figure 3.9: Results obtained on the Joint ($\sigma = 0.5l_\lambda$).



3.10(a): Original 3.10(b): Noisy 3.10(c): BNF 3.10(d): CNR

3.10(e): RHF 3.10(f): AP 3.10(g): Ours

Figure 3.10: Results obtained on the Merlion ($\sigma = 0.5 l_\lambda$).

4
Enveloping CAD mesh models

This chapter is a slightly modified version of [98].

4.1
Introduction

CAD models are widely used in several industries for manufacturing, such as aerospace, automotive, architectural, electronics, entertainment, oil & gas, sports, and so on. These models are commonly used to document a product's design process involving a high-fidelity representation of the physical object(s). For this reason, these models are not optimized for real-time visualization, interaction, or simulation. In order to make them available for the latter tasks, a simplification process is required. The kind of simplification depends on how the model will be used in a specific application. For example, for a virtual reality (VR) application that allows interaction with the virtual objects, the simplification result should be a polygon mesh with the following properties: geometric fidelity w.r.t the CAD model, low number of polygons and texture coordinates [99].

In recent years, XR is being used in e-commerce applications to simulate product experience (e.g., Ikea Place[1]). Users can test if a product fits in the space they have and see how it looks like in a virtual environment. This kind of application usually works on mobile devices, and the product's representation must consider the hardware limitations. In this case, the representation must be optimal for visualization and interaction, allowing efficient implementation of simple interactions such as rigid transformations. Commonly, designers use a mesh editor to manually design a polygon mesh with a low number of polygons (low-poly) that approximates the original CAD model.

In order to automatize this procedure, several CAD systems are including automatic and semi-automatic defeaturing methods to reduce the complexity of the CAD model, and then export it as a mesh. Defeaturing is related to CAD model simplification by detecting and processing features, and is dependent on the CAD system model representation (e.g., constructive solid geometry, B-Rep [100, 101, 102], feature-based [103, 104, 105], etc.). Since there exists a vast

[1] https://highlights.ikea.com/2017/ikea-place/

amount of feature classes without a universal definition [106], feature detection is not a trivial task. Some works focus on the detection of basic features such as circle holes, chamfers, or blends. This phenomena and the model representation dependency, make these methods lack generality.

Most of CAD model representations allow the generation of a polygon mesh, called CAD mesh model. This mesh approximates the surface of the CAD model with high fidelity, representing all possible features. Typically, the resulting mesh is irregular [107] and may be non-manifold [108], which makes more difficult the mesh processing tasks. The direct simplification of this mesh can be considered as a more general solution to simplify the CAD model.

In this work, we propose a fully automatic method for the extreme simplification of CAD mesh models, trying to maintain the visualization quality while removing unnecessary features. We focus on XR applications that require visualization and interaction with the model. In that sense, the resulting representation must be as simple as possible to allow both tasks. Our method receives as input a triangle mesh generated from a CAD model and returns a low-poly 2-manifold triangle mesh that we call XR mesh. This mesh approximates the geometry of the outer shape of the CAD model, working as a new type of 3D polygonal impostor. Differently from classical impostors [109, 110, 111, 112, 113, 114, 115, 116], the proposed representation enables all sorts of interaction as a conventional CAD mesh model. Impostors usually encode the visual aspect of the model lacking the geometric properties that would enable a more complex level of interaction, i.e., collision and contact.

This chapter is structured as follows. In Section 4.2, we explain some previous work relevant to our proposal. In Section 4.3, we explain the proposed method in detail. In Section 4.4, we show our experiments and results. In Section 4.5, we discuss some properties of our method. Finally, in Section 4.6, we give our conclusions and future work.

4.2 Related work

There are many mathematical representations of objects in a CAD system, but almost all of them allow the generation of a polygon mesh, which is the usual representation used in computer graphics. Typically, these meshes are complex, irregular, contain sharp features, and maybe non-manifold (See Figure 4.1). In this section, we focus on methods that allow efficient visualization and interaction using a polygon mesh as input.

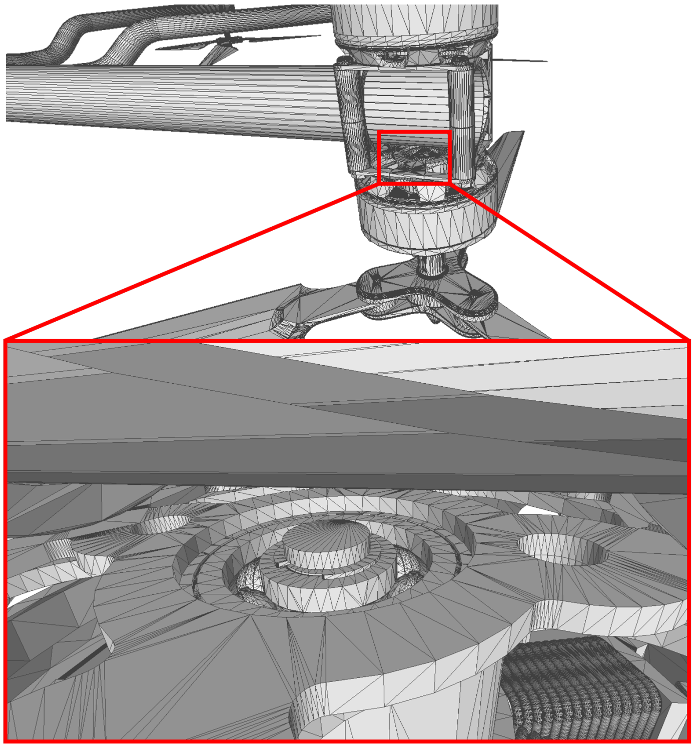


Figure 4.1: Irregular triangulation of the Drone model and zoomed view of small features that are visible because of the gap.

4.2.1 Impostors

Efficient visualization of complex 3D polygon meshes can be addressed by using impostors. In the context of image-based rendering, for example, an impostor is a simple 3D geometry that fools the viewer using textures. Early methods use a screen aligned textured quadrilateral (billboard) that updates its image texture using pre-rendered images [109, 112]. Instead of using a quadrilateral, Sillion et al. use a planar mesh to approximate a 3D image warp [110]. Décoret et al. introduce the usage of multiple layers to avoid some artifacts [111]. In [113], a billboard cloud computed using a geometry simplification error metric is proposed to represent a single object.

Using relief maps, Policarpo and Oliveira propose the usage of six impostors that define the bounding box of the object [114]. Similarly, Risser uses bounding billboards facing the viewer [115]. Andújaer et al. use overlapping heightfields from different views that improve visualization from any direction [116].

Point-based rendering is a different way to visualize complex models [117]. The idea is to use a vast number of points to sample the original surface and then fill the gaps between them. The main difficulty is how to choose the sampling density. Wimmer et al. used this representation to generate an image-based impostor [118].

This kind of technique is recommended when the rendered object is far enough from the viewer, the object is static, or the view does not change too much [119]. However, we are interested in XR applications that need close views in multiple directions and object manipulation.

4.2.2 Geometry simplification and approximation

Another way to improve the efficiency of visualization and interaction is to approximate the geometry of the high-resolution mesh with a mesh with fewer polygons. These methods are classified into four main categories: vertex clustering, incremental decimation, resampling, and mesh approximation [120].

Vertex clustering methods merge vertices contained in the same cell of a regular volumetric grid and optimize the position of the new vertex [121, 122]. These methods are simple and fast, but the resulting resolution depends on the size of the cells, which can be hard to determine when the input mesh is irregular.

Incremental decimation methods are widely used and extended in different ways because they are robust and easy to control. The decimation process is related to iteratively collapse vertices [123] or edges [124, 125, 126, 127, 128], minimizing a geometric error metric. In addition to this error, some works introduce a view dependent error metric to penalize visual similarity [129, 130, 131, 132, 133], visibility [134] or saliency [135].

Resampling methods estimate a point distribution over the surface that the input mesh represents, and compute a new mesh using it [136, 137, 138, 139]. The new mesh uses fewer polygons than the original one. For example, Yan et al. [136] proposed the usage of a centroidal Voronoi tesselation (CVT) on the surface to generate a regular triangle mesh. The resolution of the mesh depends on the number of Voronoi regions.

Mesh approximation is related to find piecewise constant regions of a mesh that can be represented by a simple primitive like a plane. Cohen-Steiner et al. proposed a method to cluster these regions in an iterative manner [140]. The method minimizes the squared orthogonal distance w.r.t. the cluster reference point. Then the vertices on the clusterization boundaries are used to generate polygons.

In addition to this type of method, volume-based simplification methods are used for the simplification of CAD mesh models. The main idea is to voxelize the mesh, simplify the volume representation, and then reconstruct a mesh from the processed volume [141, 142, 143, 144].

Geometry simplification methods tend to approximate the CAD model's surface without taking into account the exclusion of features and non-visible regions.

4.2.3 Defeaturing

Defeaturing was also applied to CAD mesh models. Dey et al. [145] focus on small feature removal by using edge collapse operations. Jang et al. [146] propose the detection and simplification of loop-based features on manifold meshes. Sunil and Pande [147] propose a heuristic feature detector on CAD mesh models of sheet metal parts. Gao et al. [107] define a region-based representation to classify each region as a feature or not. Then the features are suppressed following three main strategies that depend on the feature class. Feature detection and suppression steps are performed iteratively until the desired simplification is acquired. Using a geometry-based size field, Quadros and Owen [148] propose the detection and removal of irrelevant features by using edge collapse operations. Their method takes care of geometry and topology preservation. Huang and Wang [149] propose a volume-based mesh simplification algorithm that is capable of fulfilling concave features. They use a binary space partition tree, which provides a compact and robust representation of the volume.

Since defeaturing methods are based on feature recognition, they usually work for a specific subset of feature classes. Also, the processing of features is a challenging task in the mesh domain.

4.2.4 Hidden surface removal

In complex CAD models, several regions of the surface are hidden or severely occluded in any external view. Most of the methods presented above tend to simplify the original surface without taking into account if a region is visible or not. Hidden surface removal is an old problem whose solutions are mainly focused on rendering [150]. To remove the occluded surfaces, one can delete the polygons that are not rendered in a series of external views from different directions. One can also compute volumetric ambient occlusion and remove polygons with a value lower than a given threshold.

As mentioned, simplification algorithms can be guided by visibility. Lindstrom and Turk [129] propose an edge collapse mesh simplification method based on image dissimilarity, which indirectly tends to hard simplify occluded

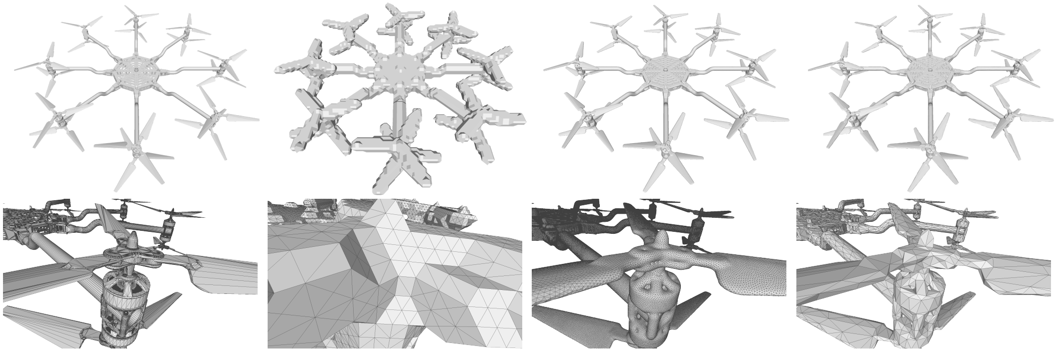


Figure 4.2: Partial results generated at each step of our method used on the Drone model. From left to right: CAD mesh model, GAC-based loose envelope (without remeshing), tight envelope, and decimated envelope (XR mesh).

regions. Following a similar idea, Zhang and Turk [134] introduce a visibility function whose values are combined with a quadric error metric.

However, in some cases, CAD models have grids and gaps that make visible most of the internal surface. Figure 4.1 shows small features that are visible because of the gap but can be ignored for visualization purposes.

4.2.5 Proposal

We propose an extreme simplification method that seeks to use a low number of triangles to represent the outer shape of the CAD model's surface. We compute a bounding mesh that works as an adaptive envelope that wraps the original CAD model. This mesh works as a 3D polygonal impostor and tends to simplify the geometry, remove and smooth unnecessary features, and ignore regions with low visibility. Our proposal is closely related to [151], where multiple 2D envelopes are computed on slice-based projections of the CAD model, and then merged to build the 3D envelope. Different from this work, we directly compute the 3D envelope without depending on a fine volume-based quantization (slices). Martineau et al. [152] propose the usage of an adaptive hexahedral grid to remesh and wrap a CAD mesh model, covering gaps and mesh holes. Their output is a watertight mesh which represents the outer shape of the CAD mesh model, and whose shape depends on the resolution of the adaptive grid. Since their focus is simulation and not visualization, their results show blurred features and unstable filling of semantic holes and gaps. Our proposal tends to preserve feature appearance and to simplify holes and gaps in a soft manner.

4.3 Method

We aim to compute an envelope mesh that wraps the CAD model to serve as the *geometric impostor*. This envelope must be adapted to the outer shape of the CAD model. Our method receives as input a CAD triangle mesh model and returns as output a low-resolution 2-manifold triangle mesh that can be used in XR applications (XR mesh). This mesh wraps the outer surface of the CAD mesh model and consists of three main steps. In the first step, a loose envelope mesh is generated by two different approaches: one is based on the computation of the Convex Hull (CH) mesh of the CAD mesh model, and the other is based on the evolution of an enclosing surface using the active contours method. In the second step, a tight envelope mesh is generated by deforming the loose envelope mesh in a more precise way, recovering the original outer details. The deformation process can generate several artifacts, and for this reason, we need a corrector step to reconstruct the mesh. These two processes: deform and reconstruct, are performed iteratively until the mesh reaches the desired adaptation. In the third step, the tight envelope mesh is decimated to reach the desired resolution. The output of this step is the XR mesh. Figure 4.2 shows the corresponding inputs/outputs of these steps, and the following subsections explain them in more detail.

4.3.1 Loose envelope generation

We can use different methods to generate a loose envelope. A simple bounding box, for example, encloses all the vertices, but it would require more deformation to reach a desired adaptation to the original mesh. We could also choose a more adaptive method such as alpha shapes [153], but this algorithm requires the definition of a tolerance radius that can be difficult to estimate automatically. Furthermore, alpha shapes can generate undesirable multiple connected components and include non-visible regions.

We propose two ways to generate the initial loose envelope. The first computes a CH mesh, and the second uses a volumetric approach. In both approaches, we produce a refined uniform mesh, which is then deformed to yield a tight envelope mesh. The choice of one of these two methods depends on the convexity of the outer shape of the CAD model.

4.3.1.1 Convex Hull-based loose envelope

Computing the CH of the input mesh is a viable alternative to solve the problem of generating the loose envelope. It is simple and has the following advantages: yields a single connected component, does not require any parameter definition, and generates a mesh with a reasonable level of adaptation to the input shape when it is not too complex.

Since the CH mesh is usually irregular, we remesh it using the algorithm proposed in [136], which is based on computing the CVT on the surface. The CVT generates regular Voronoi regions, i.e., with similar areas, resulting in a regular triangulation whose resolution depends on the number of seeds. The selection of the number of seeds depends on the surface area that we want to sample, more surface area will require more seeds. This regular mesh is the loose envelope mesh that is going to be adapted to the CAD mesh model.

The preliminary results using the CH approach produces promising results. In many cases, however, the CH adaptation is still far from the original input and requires too many deformation steps in the *tight envelope generation* step.

4.3.1.2 Geodesic Active Countours-based loose envelope

Active Contours is a well-known technique largely used for boundary detection in image processing and analysis. Active Contours can be defined using energy minimization or geometry flow theories. In [154, 155], Caselles et al. unified both approaches using a geodesic formulation where an energy minimization problem is mapped onto a problem of finding a geodesic curve/surface in a Riemannian space whose metric is defined by the image content.

One of the advantages of the Geodesic Active Contours (GAC) formulation for boundary detection is that it is independent of parameterization and deals appropriately with arbitrary topology by embedding the surface in a implicit formulation using level sets. Let $\mathcal{C}$ be the boundary to be detected in 3D and $\phi : [x_0, x_f] \times [y_0, y_f] \times [z_0, z_f] \times [t_0, t_f] \to \mathbb{R}$ be an embedding implicit function $\phi(x, y, z, t)$ with zero-level set $\Gamma(t) = \{(x, y, z) \in \mathbb{R}^3 : \phi(x, y, z, t) = 0\}$. It is possible to define ϕ such that its evolution is equivalent to the evolution of the surface, i.e, $\Gamma(t) \equiv \mathcal{C}(t)$. The level set corresponding to the detected boundary can be obtained by the state solution $\frac{\partial \phi}{\partial t} = 0$ of the equation below:

$$\frac{\partial \phi}{\partial t} = \alpha g(I)|\nabla \phi| + \beta \kappa g(I)|\nabla \phi| + \lambda \nabla g(I) \cdot \nabla \phi, \tag{4-1}$$

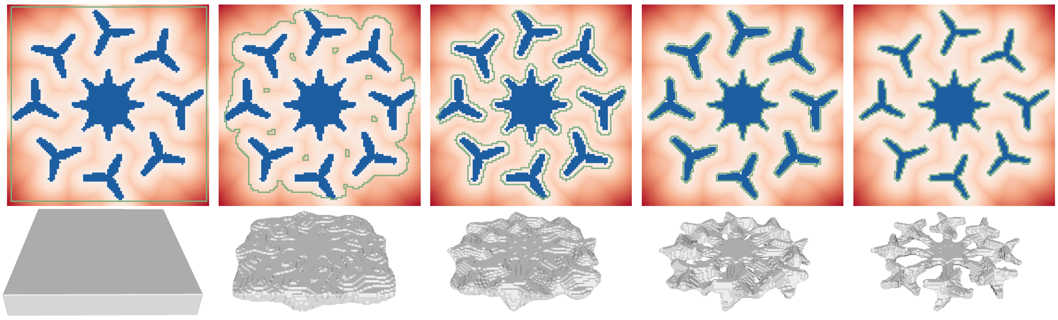


Figure 4.3: *Loose envelope generation* using the Drone model. Each column shows the initial surface and the partial results of the 60th, 120th, 180th, and 240th iterations. The first row shows a single slice of g, where blue means -1 and red means 1. The evolved surface is shown in green. The second row shows the reconstructed mesh from the evolved surface.

where $\phi_0 = \phi(x, y, z, t = 0)$ is the initial signed distance function with respect to $\Gamma_0 = \Gamma(t = 0)$, $g : [0, +\infty] \to \mathbb{R}$ is a positive strictly decreasing function and I is the image. The first two terms in the equation act as the inner forces and the last term acts as the external force, which depends on the gradient of a function g of the image. The first one drives the level set, and consequently, the corresponding surface, inwards ($\alpha < 0$) or outwards ($\alpha > 0$) with g-dependant velocity in the direction defined by the normal field of ϕ. It can be seen as defining a ballooning effect controlled by the parameter α. The second term, which depends on the curvature κ and the parameter β, is the heat flow term that minimizes the total curvature and also the geodesic area of the surface. The third term is responsible for moving the boundary surface towards the middle of the image edges and is controlled by the parameter λ. This term makes the method robust in cases where the image edges are not well defined, and the gradients vary along the boundary.

In our case, we adapt the method above as we do not have a volumetric image I, including an object whose boundary we must detect. Instead, we have a high-resolution non-uniform mesh $\mathcal{M}$ for which we want to compute the tightest envelope as possible, as the result of the evolution of the level sets of an embedding function ϕ. First, we define a bounding volume $\Omega \subseteq \mathbb{R}^3$ that approximates the region occupied by the bounding box of $\mathcal{M}$. Then we compute a distance field $\psi : \Omega \to \mathbb{R}$ that defines the point-to-mesh distance regarding $\mathcal{M}$. We use this function to define g as follows:

$$g(x, y, z) = \begin{cases} -1 & -\epsilon_l < \psi(x, y, z) < \epsilon_l \\ \frac{\psi(x,y,z)}{\max(\psi)} & otherwise \end{cases}, \qquad (4\text{-}2)$$

where $\max(\psi)$ is the maximum value of ψ (i.e., maximum distance), and ϵ_l is a bandwidth value that determines the region where the surface should not evolve. The latter is possible because negative values of g make the surface evolve in the opposite direction of the ballooning force. Finally, we define ϕ_0 as a signed distance field that indicates the signed distance to the surface of the bounding box of $\mathcal{M}$, where negative and positive values of ϕ_0 correspond to the interior and exterior regions, respectively.

We discretize the domain as a regular grid with a_{spa} spacing that covers the region occupied by Ω and includes a padding amount of a_{pad}. Then, we define g and ϕ on this grid and evolve the surface for a given time t, which can be defined by a number of iterations n_{it_l} and a time step equal to 1. The surface evolution depends on the parameters α, β, and λ, where α should be a negative value to acquire a contraction behavior.

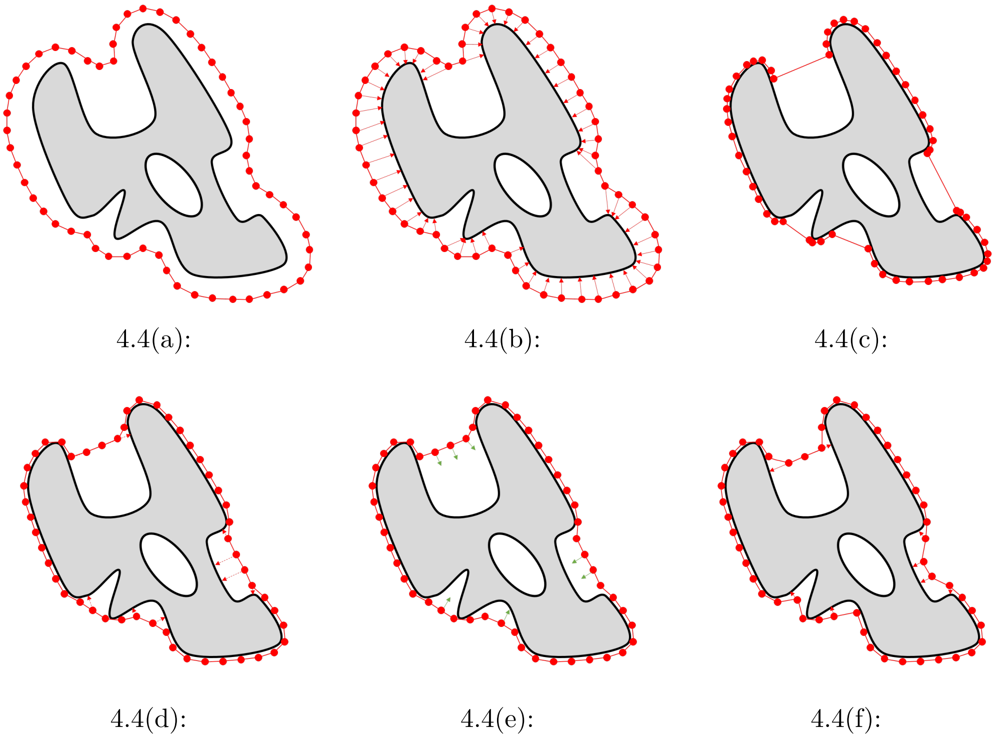
Once we have the final surface, we reconstruct the corresponding mesh by binarizing the interior region and applying the marching cubes algorithm [156]. This mesh is remeshed using [136] and defined as the loose envelope mesh, which is more adaptive than a CH mesh. Figure 4.3 shows how the surface evolves and how the resulting partial meshes look like through the iterations. The more iterations we perform the better will be the adaptation of the envelope to $\mathcal{M}$ and closer to the restricted region it will be.

4.3.2 Tight envelope generation

The goal of this step is to deform the loose envelope until it touches $\mathcal{M}$. Let us denote the envelope mesh, which is actually the loose envelope, as $\mathcal{M}_e = (\mathcal{V}_e, \mathcal{F}_e)$, where $\mathcal{V}_e = \{1, \ldots, n_e\}$ and $\mathcal{F}_e \subseteq \mathcal{V}_e \times \mathcal{V}_e \times \mathcal{V}_e$ are the vertices and faces, respectively. Also, consider $X_e = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_e}\}$ and $N_e = \{\mathbf{n}_1, \ldots, \mathbf{n}_{n_e}\}$ as the positions and normals of the vertices.

Inspired by the active contours models, we define a deformation process influenced by three main forces applied in a consecutive manner. The first one defines the attraction to $\mathcal{M}$, and simulates the projection of the vertices of $\mathcal{M}_e$ on $\mathcal{M}$. The second one, using the Laplacian operator, smooths and contracts $\mathcal{M}_e$ in regions that are not close enough to $\mathcal{M}$. The third one works as a ballooning force, where the smoothed vertices are pushed in directions opposite to their corresponding normals.

Different from an implicit surface evolution approach, deforming a mesh requires complex operations to solve topological changes, to avoid self-intersections, and to resample the surface. Also, it is complex to compute an attraction force regarding $\mathcal{M}$ at a given time t, without defining a regular grid. The gain of deforming a mesh arises from the fact that we are representing just the points on the surface instead of a volume that includes them, i.e., we avoid



4.4(a): 4.4(b): 4.4(c):

4.4(d): 4.4(e): 4.4(f):

Figure 4.4: 2D representation of the *tight envelope generation*. The gray shape represents a solid model. The red points and lines represent the envelope vertices and segments. The red arrows represent the displacement vectors δ_i. The green arrows represent the normal opposite directions. (a) Loose envelope. (b) Projection directions of the first iteration. (c) Projection result of the first iteration. (d) Possible projection directions of the second iteration (without applying deformation in the opposite normal direction). (e) Deformation directions following the opposite normal direction of the first iteration. (f) Deformation in the opposite normal direction and projection directions of the second iteration.

loss of information resulting from a volume-based quantization process. So, we can better sample the envelope surface by using a large enough resolution for $\mathcal{M}_e$ that allows us to preserve the outer shape of $\mathcal{M}$.

The *tight envelope generation* consists in applying the deformation forces in an iterative manner, where n_{it_t} denotes the number of iterations. After the attraction force, we reconstruct the mesh to uniformize vertex distribution and remove artifacts. The required deformation operations are explained in the following subsections. Figures 4.4 and 4.5 describe how the *tight envelope generation* step works in 2D and 3D, respectively.

4.3.2.1
Vertex projection and mesh reconstruction

The first deformation force is responsible for attracting $\mathcal{M}_e$ to $\mathcal{M}$. Since a discrete distance field w.r.t. $\mathcal{M}$ is not available in this formulation, we can not define an attraction direction for an arbitrary point in the domain. So, for a vertex $v_{i_e} \in \mathcal{V}_e$, we opted to use as attraction direction the corresponding direction that points towards the closest point $p \in S(\mathcal{M})$ (point-to-mesh distance), where $S(\mathcal{M})$ denotes the surface that $\mathcal{M}$ is representing. For a single time step, i.e., iteration, we define the displacement vector δ as follows:

$$\delta_i = \left(\underset{\mathbf{p} \in S(\mathcal{M})}{\arg\min}\, d(\mathbf{x}_{i_e}, \mathbf{p}) \right) - \mathbf{x}_{i_e}, \tag{4-3}$$

where $d(\cdot, \cdot)$ denotes the euclidean distance between two points. This displacement vector moves the corresponding vertex to the closest point in $S(\mathcal{M})$, working as a *vertex projection* on $\mathcal{M}$. We propose these abrupt displacements because computing n_e point-to-mesh distances is computationally expensive. Using a 2D representation, Figure 4.4(a) shows a regular sampled loose envelope and Figure 4.4(b) shows the corresponding vectors δ_i.

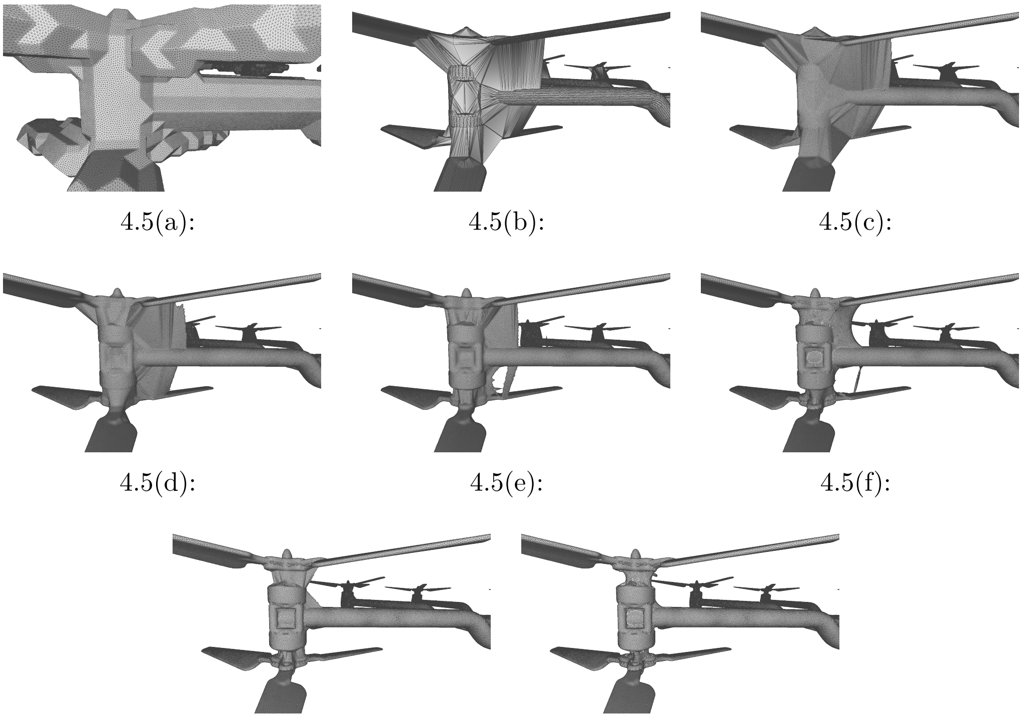
As in the *loose envelope generation*, we use a tight bandwidth ϵ_t to project the vertices on the continuous (ϵ_t)-level set of the continuous distance field w.r.t. $\mathcal{M}$. The *vertex projection* operation is defined as follows:

$$\mathbf{x}'_{i_e} = \mathbf{x}_{i_e} + \left(\delta_i - \epsilon_t \frac{\delta_i}{|\delta_i|} \right), \tag{4-4}$$

where $\mathbf{x}'_{i_e}$ is the new vertex position. The bandwidth ϵ_t helps the deformation process to avoid later mesh artifacts and to wrap $\mathcal{M}$ instead of overlap.

The *vertex projection* operation can generate several artifacts on $\mathcal{M}_e$ that we have to remove before the next operations. As we maintain the topology when we project vertices that are far from $\mathcal{M}$, neighboring triangles can be severely stretched. The initial regular mesh becomes very irregular and can include self-intersections and degenerated triangles. Since we have no vertices on the region of the stretched triangles, later deformation operations can not be applied there. Figure 4.4(c) shows in 2D some of these artifacts: more than one vertex are projected on the same salient feature, and stretched segments are generated by the vertices that are close to the gaps. Figure 4.5(b) shows a 3D example of how the triangles are stretched, starting from the loose envelope mesh shown in Figure 4.5(a).

For this reason, we resample the stretched region by creating new vertices and triangles, maintaining the topology of the surface that $\mathcal{M}_e$ represents. This resampling operation allows the projection process to be applied to these regions in the next iterations, adapting $\mathcal{M}_e$ to the features of $\mathcal{M}$. Just for illustration, Figure 4.4(d) shows in 2D the resample result applied to the configuration shown in Figure 4.4(c), and the corresponding consecutive projection vectors δ_i. As we can see, the next deformation step can better



4.5(a): 4.5(b): 4.5(c):

4.5(d): 4.5(e): 4.5(f):

4.5(g): 4.5(h):

Figure 4.5: *Tight envelope generation.* The smooth and sink operations are not applied in the first two iterations. (a) Loose envelope. (b) *Vertex projection* result of the first iteration. (c) *Mesh reconstruction* result of the first iteration. (d) *Mesh reconstruction* result of the second iteration. (e) *Mesh reconstruction* result of the third iteration. (f) *Localized mesh smoothing and sinking* result of the third iteration. (g) *Mesh reconstruction* result of the fourth iteration. (h) *Localized mesh smoothing and sinking* result of the fourth iteration.

approximate the gaps where the new sampled vertices have as closest point, a point that was not yet used in the *vertex projection* operation. Note that when the latter does not occur, this region does not suffer deformation in the next projections, generating a kind of membrane that covers the gap.

In the 3D case, a naive attempt to resample the mesh could be made by splitting the stretched triangles until the new triangles are regular. However, this process can introduce several new triangles because the stretched triangles can have areas close to zero, i.e., very irregular triangles (See Figure 4.5(b)). For this reason, we rely on a *remeshing* algorithm that generates a new regular mesh that preserves the shape and the surface topology, i.e., we use the algorithm proposed in [136], as in the *loose envelope generation* step. This algorithm generates a regular 2-manifold mesh but does not deal appropriately with mesh boundaries. Mesh boundaries require special treatment if we want to remesh just the stretched triangle regions. To avoid dealing with such special cases, we propose to remesh the entire mesh instead of just the problematic regions. That allows us to achieve a simple control of the resolution of the *remeshing* result.

The *remeshing* algorithm [136] projects the new vertices onto the input mesh and tries to preserve the topology, avoiding non-manifold simplices. This treatment can generate multiple connected components and self intersected triangles because new vertices can be created to reach manifoldness. Usually, this phenomenon occurs when the target region represents a very thin volume, as the membrane artifact shown in Figure 4.5(c). If this region is not an artifact and is representing a thin volume of the CAD model, we can define a large enough ϵ_t to minimize topological changes. Note that the larger the value of ϵ_t, the thicker the tight envelope will be.

To overcome the *remeshing* algorithm problems, we propose a *remeshing post-processing* operation that uses the following operations.

- *intersected face removal.* Removes all the faces that intersect another face.
- *small connected component removal.* Removes the connected components that have a surface area under a threshold value.
- *hole filling.* Detects and triangulates mesh holes using a loop split-based algorithm.
- *topology repair.* Consists of a set of sequential simpler operations. First, vertices that are very close to each other are merged (*duplicated vertex removal*). Second, faces that share the same vertices of another face are removed, including degenerated faces (*duplicated face removal*). Third, since the *duplicated face removal* operation can introduce incoherent face orientation, the orientation of an initial face is propagated all over the mesh (*coherent reorientation of faces*). Finally, to ensure manifoldness, non-manifold vertices are split.
- *face-based normal flipping.* Although the faces can have a coherent orientation, their normals can be pointing to the interior of the envelope instead of the exterior. That happens because the normal of the initial face used for propagation in the *coherent reorientation of faces* operation can be pointing in the wrong direction. This operation uses a voting scheme that considers surrounding views to flip the normals (i.e., to reverse the orientation) if they are pointing to the wrong side.

These are typical operations used in a mesh repair process [157]. The *remeshing post-processing* operation consists of the following sequential operations: (1) *intersected face removal*; (2) *small connected component removal* to remove hole islands and small connected components introduced in previous splitting

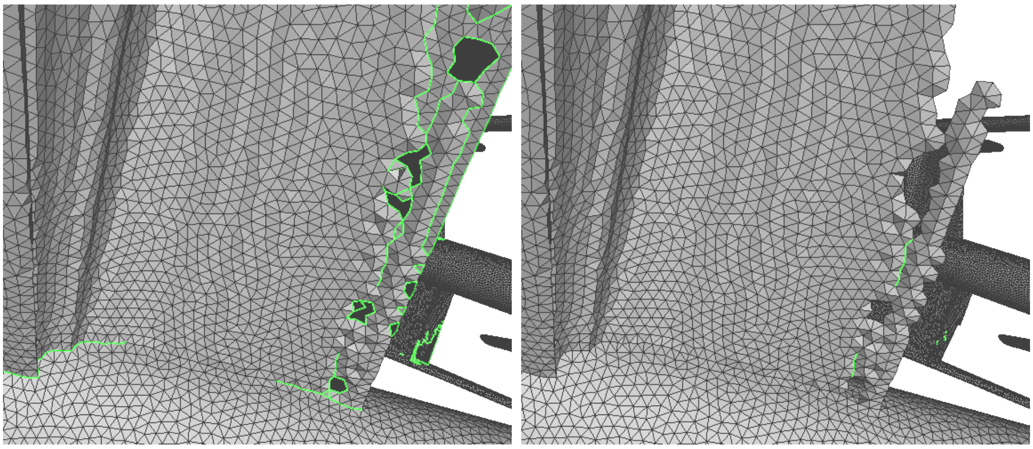


Figure 4.6: *Mesh reconstruction* partial results. Boundary edges are colored in green. Left: *remeshing* result. Right: *remeshing post-processing* result.

operations; (3) *hole filling* to fill the holes generated by the removed faces; (4) *topology repair* to solve topological issues introduced in previous operations; (6) *small connected component removal* to remove small connected components introduced by the previous *non-manifold vertex splitting*; and (7) *face-based normal flipping* to correct the direction of the face-based normals. We name the *remeshing* and *remeshing post-processing* operations as *mesh reconstruction*, whose resulting mesh allows us to apply the next deformation operations.

Figures 4.5(c), 4.5(d) and 4.5(e) show the *mesh reconstruction* results of the *vertex projection* operation output after the 1st, 2nd and 3rd iterations, respectively. We can see how $\mathcal{M}_e$ is adapted to $\mathcal{M}$ through the iterations. Also, it is important to remark that the *remeshing post-processing* operations drastically reduce the number of connected components because it tends to merge or remove them. For example, the *duplicated vertex removal* operation consists of merging vertices that are very close to each other; if the vertices of different connected components are merged, a larger connected component that includes both components will be generated. In addition, the *small connected component removal* operation removes several connected components that are generated by the splitting operations used in the *remeshing* algorithm. Figure 4.6 shows an example of the *remeshing* result, and the *remeshing post-processing* result, where the green edges define the mesh boundaries.

4.3.2.2 Localized mesh smoothing and contraction

The second force strongly smooths $\mathcal{M}_e$ on regions that are far from $\mathcal{M}$. This is done by using a constrained laplacian smoothing operation. As shown in [158], this type of smoothing contracts the mesh by minimizing the surface area. This operation is applied iteratively, where n_{it_s} defines the number of iterations. Let us denote the set of vertices $\mathcal{U}_e \subseteq \mathcal{V}_e$ that are far from $\mathcal{M}$ as follows:

$$\mathcal{U}_e = \left\{ v_{k_e} \in \mathcal{V}_e : \min_{\mathbf{p} \in S(\mathcal{M})} d(\mathbf{x}_{k_e}, \mathbf{p}) > \epsilon_d \right\}, \tag{4-5}$$

where ϵ_d is a tolerance amount that defines how far the vertices should be from $\mathcal{M}$. A single vertex updating step of this operation is defined by:

$$\mathbf{x}'_{i_e} = \begin{cases} \sum\limits_{v_{je} \in \mathcal{N}(v_{i_e})} \frac{\mathbf{x}_{je}}{|\mathcal{N}(v_{i_e})|} & v_{i_e} \in \mathcal{U}_e \\ \mathbf{x}_{i_e} & otherwise \end{cases}, tr \tag{4-6}$$

where $\mathbf{x}'_{i_e}$ is the new position, $\mathcal{N}(v_{i_e})$ is the set of neighboring vertices of v_{i_e}, and $|\mathcal{N}(v_{i_e})|$ is the number of neighboring vertices of v_{i_e}.

This deformation force is helpful in removing the remaining undesired membranes because the mesh is contracted in these regions. Figure 4.5(f) shows how the membranes of the mesh shown in Figure 4.5(e) are contracted, such that the next operations remove them, as shown in Figure 4.5(g). The removal of these membrane regions occurs for three different reasons: (1) in a consecutive *vertex projection* operation, the new points of projection can better adapt the vertices of the membrane to $\mathcal{M}$ (See the larger membrane shown in Figures 4.5(f) and 4.5(g)); (2) in a consecutive *mesh reconstrution* operation, the membrane area can be insufficient to be resampled by the *remeshing* algorithm; (3) in a consecutive *mesh reconstrution* operation, due to the membrane thickness, the *remeshing* algorithm can generate several connected components that can be removed in the *remeshing post-processing* operation.

4.3.2.3 Localized mesh sinking

The third force works as the ballooning force used in the active contours models. We sink the regions of $\mathcal{M}_e$ deformed in the previous operation, by moving the vertices in the opposite direction of their normals. The vertex normal is estimated as the average normal of the neighboring triangles, and the amount of displacement is defined by a parameter a_n. The updating step for this deformation operation is defined as follows:

$$\mathbf{x}'_{i_e} = \begin{cases} \mathbf{x}_{i_e} - a_n \mathbf{n}_{i_e} & v_{i_e} \in \mathcal{U}_e \\ \mathbf{x}_{i_e} & otherwise \end{cases}, \tag{4-7}$$

where $\mathbf{x}'_{i_e}$ is the new position. Since filling holes and gaps, in the volumetric sense, can be the desired behavior to simplify $\mathcal{M}$, we can fix $a_n = 0$ to avoid adaptation to them. Figure 4.4(e) shows in 2D the displacement vectors of this operation, and Figure 4.4(f) shows how the consecutive *vertex projection* vectors looks like. As we can see, in some cases, this is the only way to allow the envelope to evolve into the gaps. For visualization purposes, this deformation operation is important to enhance the details.

4.3.3 Mesh decimation

As a result of the *tight envelope generation* step, we pick the output of the *mesh reconstruction* operation of the last iteration. We select this mesh because it is more regular than the other outputs and has more flat regions that can be further simplified. In general, the result of the *tight envelope generation* is a high resolution dense regular 2-manifold mesh, possibly with boundaries, which is the desired input for mesh decimation algorithms. In order to obtain a low-resolution mesh, we decimate it using the Quadric Edge Collapse (QEC) method [124] until we reach the desired resolution. Since the edge collapse operation can introduce artifacts, we apply the following *decimation post-processing* pipeline: (1) *topology repair*; (2) *small connected component removal*; (3) *hole filling*; (4) *topology repair*; and (5) *face-based normal flipping*. The latter operations work as a mesh repair process and help to avoid non-manifold simplices. The definition of these operations is the same used for *mesh reconstruction*.

The resulting mesh (XR mesh) can be used directly in XR applications or at least used as a starting point for a low-poly design process. Furthermore, as it is a 2-manifold mesh with a simple topology, it is easy to introduce seams and generate a planar parameterization [159] for the generation of texture coordinates. Then, we can bake textures using $\mathcal{M}$, such as normal, color, ambient occlusion, and displacement maps.

4.3.4 Implementation details

Our implementation is based on the Insight Segmentation and Registration Toolkit v4.12 [160], the Geogram library v1.6.9 [161], the OpenMesh library v7.1 [162], and MeshLab v2016.12 [163].

To compute the distance field ψ used in the *loose envelope generation*, we just compute the point-to-mesh distance from the center of each voxel of the regular grid to the closest point contained in a triangle of $\mathcal{M}$. The initial surface is computed by first generating a binary 3D image that defines the interior of the bounding box and then using [164] to compute its corresponding signed

Algorithm 3 Mesh reconstruction

```
procedure MESH_RECONSTRUCTION($\mathcal{V}_e$,$\mathcal{F}_e$,$n_r$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ remeshing($\mathcal{V}_e, \mathcal{F}_e, n_r$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ intersected_face_removal($\mathcal{V}_e, \mathcal{F}_e$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ small_connected_component_removal($\mathcal{V}_e, \mathcal{F}_e$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ hole_filling($\mathcal{V}_e, \mathcal{F}_e$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ topology_repair($\mathcal{V}_e, \mathcal{F}_e$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ small_connected_component_removal($\mathcal{V}_e, \mathcal{F}_e$)
    $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ face-based_normals_flipping($\mathcal{V}_e, \mathcal{F}_e$)
    return $(\mathcal{V}_e, \mathcal{F}_e)$
```

distance field. The surfacer is evolved by defining the parameters α, β and λ, and the image that represents g.

As $\mathcal{M}$ is an irregular mesh, finding the closest point contained in any of its triangles, results in a complex operation. For this reason, we create a dense point cloud that samples the triangles of $\mathcal{M}$ and keeps the point-to-triangle correspondence. Given a desired number of random samples n_s, we sample $\left\lfloor \frac{\text{area}(f)}{\text{area}(\mathcal{M})} n_s \right\rfloor$ random points for each triangle f, where area(f) is the triangle area and area($\mathcal{M}$) is the area of all the triangles of $\mathcal{M}$. Also, we include in the point cloud, the vertices, and centroids of all the triangles of $\mathcal{M}$. Then we index the point cloud in a KD-Tree to perform spatial queries in optimal time. The *point sampling* and *KD-Tree indexation* operations are performed in a *pre-processing* step, because spatial queries are required for the *loose envelope generation* and the *tight envelope generation* steps.

To obtain the closest point within the mesh $\mathcal{M}$, we compute the k nearest points with their corresponding triangles, adopting an approximate nearest neighbors search. Then, from this subset of triangles, we compute the closest point by using a point-to-triangle distance.

The *mesh reconstruction* operation is summarized in Algorithm 3, where n_r defines the number of vertices of the *remeshing* output. The parameters for each sub-operation of the *mesh reconstruction* operation and the *decimation post-processing* (e.g., CVT optimization method, number of iterations, small component threshold area, etc.) are the default values used in the Geogram library.

Algorithm 4 summarizes the *tight envelope generation* step. The parameters of the algorithm are the original mesh $\mathcal{M}$, the envelope mesh vertices and faces $(\mathcal{V}_e, \mathcal{F}_e)$, the number of iterations (n_{it_t}), the number of smoothing iterations (n_{it_s}), the number of seeds for the *remeshing* operation (n_r), the tight envelope bandwidth (ϵ_t), the amount of tolerance to consider if a point is far enough (ϵ_d), and the amount of movement in the opposite normal direction (a_n). The algorithm returns a new set of vertices and faces.

Algorithm 4 Tight envelope generation

```
procedure ENVELOPE($\mathcal{V}_e$,$\mathcal{F}_e$,$\mathcal{M}$,$n_{it_t}$,$n_{it_s}$,$n_r$,$\epsilon_f$,$\epsilon_s$,$a_n$)
    for $it_t \leftarrow 1$ to $n_{it_t}$ do
        for each $v_i \in V_E$ do
            $\mathbf{p} \leftarrow$closestPoint($\mathcal{M}$,$\mathbf{x}_{i_e}$)
            $\delta_i \leftarrow \mathbf{p} - \mathbf{x}_{i_e}$
            $\mathbf{x}_{i_e} \leftarrow \mathbf{x}_{i_e} + \left(\delta_i - \epsilon_t \frac{\delta_i}{|\delta_i|}\right)$
        $(\mathcal{V}_e, \mathcal{F}_e) \leftarrow$ mesh_reconstruction($\mathcal{V}_e, \mathcal{F}_e, n_r$)
        if $it_t = n_{it_t}$ then
            return $(\mathcal{V}_e, \mathcal{F}_e)$
        $\mathcal{U}_e \leftarrow \{\}$
        for each $v_{i_e} \in \mathcal{V}_e$ do
            $\mathbf{p} \leftarrow$closestPoint($\mathcal{M}$,$\mathbf{x}_{i_e}$)
            $d \leftarrow |\mathbf{x}_i - \mathbf{p}|$
            if $d > \epsilon_d$ then
                $\mathcal{U}_e \leftarrow \{\mathcal{U}_e, v_{i_e}\}$
        for $it_s \leftarrow 1$ to $n_{it_s}$ do
            for each $v_{i_e} \in \mathcal{U}_e$ do (in parallel)
                $\mathbf{p} \leftarrow (0, 0, 0)$
                for each $v_{j_e} \in \mathcal{N}(v_{i_e})$ do
                    $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{x}_{j_e}$
                $\mathbf{x}_{i_e} \leftarrow \frac{\mathbf{p}}{|\mathcal{N}(v_{i_e})|}$
        for each $v_{i_e} \in \mathcal{U}_e$ do (in parallel)
            $\mathbf{x}_{i_e} \leftarrow \mathbf{x}_{i_e} - a_n * \mathbf{n}_{i_e}$
```

For the *mesh decimation* algorithm, we use the QEC implementation provided in MeshLab, using its default parameters and enabling topology preservation and planar simplification.

4.4 Experiments and results

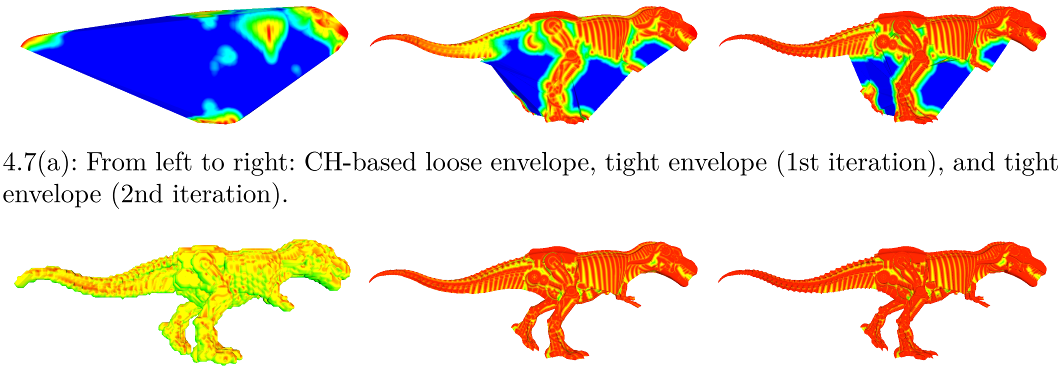
All the CAD models used in this work were obtained from the GrabCAD repository[2]. The meshes were exported using FreeCAD[3] and are shown in Figures 4.2 and 4.13. We named these models as follows: Arm, Cylinder, Drone, Engine, and T-Rex. The experiments were performed on an Intel (R) Core (TM) i5-8400 CPU @ 2.80GHz processor with 16,0 GB RAM and Windows 10 64-bit operating system. The parameters that we used for our experiments are summarized in Table 4.1, where l_d represents the bounding box diagonal length of $\mathcal{M}$. In the *tight envelope generation*, we do not apply the smoothing and sinking operations in the first two iterations to avoid over deformation in regions that are not yet close enough to $\mathcal{M}$. We adopted the same parameters

[2] https://grabcad.com/

[3] https://www.freecadweb.org/

Table 4.1: Parameters used for our experiments.

Parameter	Value	Parameter	Value
n_s	$50M$	ϵ_c	$1.5a_{spa}$
a_{spa}	$0.007l_d$	n_{it_t}	10
a_{pad}	$0.03l_d$	n_{it_s}	$1K$
n_{it_l}	2000	n_r	$700K$
α	-1.0	ϵ_f	$0.0005l_d$
β	0.1	ϵ_s	$1.1\epsilon_f$
λ	1.0	a_n	$0.9\epsilon_f$



4.7(a): From left to right: CH-based loose envelope, tight envelope (1st iteration), and tight envelope (2nd iteration).

4.7(b): From left to right: GAC-based loose envelope, tight envelope (1st iteration), and tight envelope (2nd iteration).

Figure 4.7: *Tight envelope generation* partial results through the iterations using CH-based loose envelope and the GAC-based loose envelope. The color maps the distance regarding the CAD mesh model, where red means minimum distance and blue means maximum distance.

for all the models in order to show that our method is not sensitive to them.

4.4.1
Convex Hull vs. Geodesic Active Contours

As a first experiment, we evaluate the difference between the usage of a CH-based loose envelope and a GAC-based loose envelope. Using the T-Rex CAD mesh model as $\mathcal{M}$, we generate the corresponding remeshed CH mesh with $n_r = 700K$. Then, we generate the loose envelope based on the GAC model, using the parameters shown in Table 4.1. The corresponding results are shown in the first column of Figures 4.7(a) and 4.7(b), where the color encodes the euclidean distance to the closest point in $\mathcal{M}$. It is clear that the GAC-based loose envelope is more adaptive than the CH-based. Finally, we use both results to compute the corresponding tight envelopes using the parameters shown in Table 4.1. The second and third column of Figures 4.7(a) and 4.7(b) show the partial results of the first and second iterations, where the color maps the euclidean distance w.r.t. $\mathcal{M}$. As shown in the figures, the CH-based *loose*

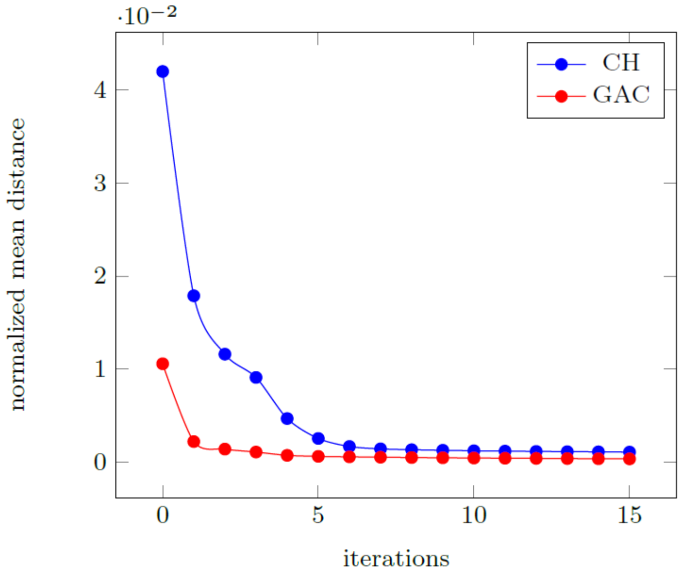


Figure 4.8: Normalized mean distance through the iterations of the *tight envelope generation* step, using the CH-based loose envelope and the GAC-based loose envelope.

envelope generation presents some difficulties to sink some regions that are far from $\mathcal{M}$, due to the concavity. Also, the GAC-based loose envelope can have an arbitrary genus while the CH-based loose envelope has always genus 0. Starting from a more adaptive mesh with a genus similar to the desired resultant mesh is helpful to avoid several artifacts on the *tight envelope generation* step.

To measure the geometric similarity between two meshes, i.e., between $\mathcal{M}_e$ and $\mathcal{M}$, we adopt the uniform mean distance between meshes. The mean distance between $\mathcal{M}_e$ and $\mathcal{M}$ is computed by randomly sampling 1M points on $\mathcal{M}_e$ and then computing the average point-to-mesh distance from these samples to $\mathcal{M}$. For both loose envelopes, Figure 4.8 shows the mean distance to $\mathcal{M}$ of the *tight envelope generation* partial results through the iterations. As we can see, the GAC-based loose envelope rapidly converges while the CH-based loose envelope requires more iterations. For this reason, in the following experiments, we use the GAC-based loose envelope as input for the *tight envelope generation* step. In the case that the desired shape is close to a convex shape, we recommend using the CH-based loose envelope to avoid parameter definition and to reduce the execution time.

4.4.2 Loose and tight envelope deformation through the iterations

Using all the models, Figure 4.9 shows the mean distance to $\mathcal{M}$ of the partial results of the *loose envelope generation* through the iterations. All

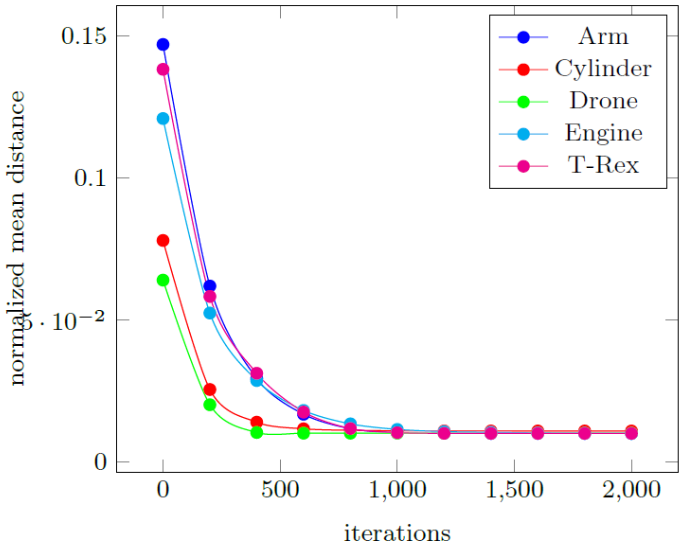


Figure 4.9: *Loose envelope generation* normalized mean distance through the iterations.

of them start to converge close to 1000 iterations. We decided to use 2000 iterations for the next experiments because more details can be captured, and the execution time is not significant compared to the *tight envelope generation*, as shown in a later experiment.

Then, Figure 4.10 shows the mean distance to $\mathcal{M}$ of the partial results of the *tight envelope generation* through the iterations. In this case, the behavior is different and strongly depends on the shape of the CAD model. For example, the Cylinder tight envelope can continue evolving to retrieve more details because it has several salient thin features and multiple gaps, while the Drone tight envelope rapidly converges due to its simpler shape that was well approximated by the loose envelope.

4.4.3 Comparison with geometric simplification

Clustering and iterative edge collapse methods are commonly used to simplify CAD mesh models [165]. In the case of clustering methods, it is hard to reach a desired number of triangles because the simplification usually depends on the definition of a regular grid. Iterative edge collapse methods are preferred because they are easy to control and better preserve the geometry and appearance. In this experiment, we compare our method with the QEC method [124] implemented in MeshLab by using the default parameters and enabling planar simplification. Different from the configuration used in our method, allowing topological changes, improves the direct decimation of the CAD mesh model because the algorithm can merge not connected vertices.

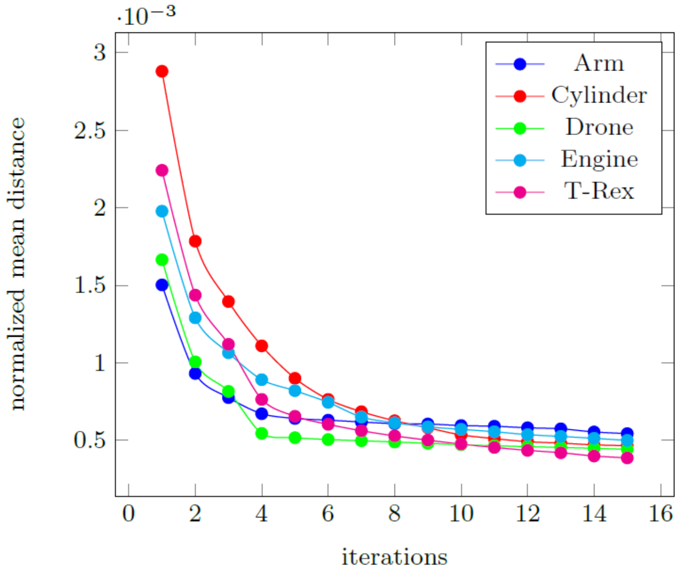


Figure 4.10: *Tight envelope generation* normalized mean distance through the iterations.

Our method and the QEC decimation were executed considering three different resolutions: 60K, 30K, and 10K triangles. We evaluate the results, considering topological, geometric, and visualization measures. The topological measures include the number of connected components (comp), the genus, the number of holes (holes), the number of non-manifold vertices (nm vert), and the number of non-manifold edges (nm edg). The geometric measures include the sum of all edge lengths (length), the total mesh surface area (area), and the mean distance regarding $\mathcal{M}$. The mean distance is computed as in the previous experiments.

To evaluate the visualization quality, we adopt the following methodology. For each model, we use 100 random surrounding views to render the CAD mesh model and the corresponding result. Using all the rendered images for each mesh, we compute the average Structural Similarity Index (SSIM) [166]. The rendering was performed using Matlab with the following configuration: cam light source on the top of the camera, flat shading, and dull material. The rendered images have a resolution of 1024 × 1024.

Table 4.2 summarizes the measurements for the CAD mesh models, the results of the QEC decimation, and the tight (full) and decimated envelopes generated by our method. All the CAD mesh models are non-manifold and have several connected components. The tight envelopes generated for all the models are always 2-manifold meshes and are conformed by a single connected component. Note that these meshes considerably reduce the CAD mesh model surface area without disturbing the visualization. For example, in the case of the Engine model, the tight envelope's area is 0.14 times the CAD mesh model's area, and the corresponding average SSIM is 0.94. In some cases, the tight envelope has a higher number of triangles than the CAD mesh model because it depends on the parameters. Also, we can generate meshes with a complex topology such as the T-Rex tight envelope, which has a genus 85 and 35 holes. Usually, the holes are generated to avoid non-manifold simplices.

The QEC decimation tends to generate results with a lower mean distance compared to our results. The latter occurs because the envelope fills void regions avoiding the representation of internal surfaces while the QEC decimation tries to preserve all the CAD mesh model's geometry. As the envelopes are *mesh decimation* results of the tight envelope, the surface area is considerably lower compared to the QEC decimation results. Also, the sum of edge lengths is lower because the envelope represents fewer features. The topological measures show that our method generates a simpler topology, including manifoldness. Figure 4.11 shows the 30K results of our method and the QEC decimation. We can see that the QEC decimation generates several artifacts that affect the visualization and geometric fidelity. The envelope better represents the CAD mesh model's shape without introducing these annoying mesh artifacts.

Since the tight envelope has a simple topology, we can apply the topology-preserving decimation to reach a low number of triangles without problems. All our decimated envelopes present a similar average SSIM regarding the tight envelope. The latter does not occur when we directly apply the QEC decimation on the CAD mesh model. Even this method suffers when it reaches the limits of covering all the CAD model's surface. Figure 4.12 shows the 10K results of the QEC decimation and our method. We can see how the QEC decimation collapses edges that it should not, removing surface regions relevant for visualization. This is evident in the average SSIM values of most of the 10K results, which are strongly affected by this phenomenon.

Table 4.2: Topological, geometric and visualization measurements to evaluate QEC decimation and our method. CAD: CAD mesh model. Ours (full): tight envelope. QEC (60K): QEC decimation 60K result. Ours (60K): our 60K result. QEC (30K): QEC decimation 30K result. Ours (30K): our 30K result. QEC (10K): QEC decimation 10K result. Ours (10K): our 10K result. faces: number of triangles. comp: number of connected components. holes: number of holes. nmv: number of non-manifold vertices. nme: number of non-manifold edges. length: sum of all edge lengths. area: mesh surface area. dist: normalized mean distance regarding the CAD mesh model. SSIM: average SSIM regarding the CAD mesh model.

Model	Method	faces	comp	genus	holes	nmv	nme	length	area	dist	SSIM
Arm	CAD	829K	82	ND	ND	8	63K	3M	2.86M	-	-
	Ours (full)	1.4M	1	10	3	0	0	4M	2.17M	0.000595	0.9831
	QEC (60K)	60K	66	ND	ND	5	2.51K	1.32M	2.87M	0.000011	0.9988
	Ours (60K)	60K	1	10	0	0	0	0.86M	2.17M	0.000572	0.9823
	QEC (30K)	30K	38	ND	ND	18	1.38K	0.9M	2.87M	0.000032	0.9964
	Ours (30K)	30K	1	10	0	0	0	0.62M	2.17M	0.000563	0.9817
	QEC (10K)	10K	30	ND	ND	14	526	0.48M	2.84M	0.000147	0.9835
	Ours (10K)	10K	1	7	0	0	0	0.38M	2.18M	0.000533	0.9808
Cylinder	CAD	21.34M	902	ND	ND	589	17.58K	24.22K	6.45	-	-
	Ours (full)	1.39M	1	0	0	0	0	3.41K	1.57	0.000535	0.9629
	QEC (60K)	60K	585	ND	ND	302	2.82K	2.02K	5.92	0.000495	0.9048
	Ours (60K)	60K	1	0	0	0	0	0.76K	1.54	0.000542	0.9585
	QEC (30K)	30K	434	ND	ND	219	2.48K	1.37K	5.74	0.000703	0.9075
	Ours (30K)	30K	1	0	0	0	0	0.54K	1.52	0.000548	0.9544
	QEC (10K)	10K	193	ND	ND	113	1.09K	0.68K	4.93	0.001229	0.8841
	Ours (10K)	10K	1	0	0	0	0	0.31K	1.46	0.000661	0.9400
Drone	CAD	8.05M	1215	ND	ND	240	418	8.84M	1.85M	-	-
	Ours (full)	1.39M	1	0	0	0	0	2.94M	1.17M	0.000471	0.9710
	QEC (60K)	60K	765	ND	ND	316	3.83K	1.21M	1.77M	0.000034	0.9958
	Ours (60K)	60K	1	0	0	0	0	0.68M	1.17M	0.000423	0.9726
	QEC (30K)	30K	647	ND	ND	236	2.79K	0.93M	1.68M	0.000065	0.9926
	Ours (30K)	30K	1	0	0	0	0	0.49M	1.17M	0.000410	0.9711
	QEC (10K)	10K	259	ND	ND	47	843	0.62M	1.47M	0.000142	0.9570
	Ours (10K)	10K	1	0	0	0	0	0.3M	1.17M	0.000376	0.9641
Engine	CAD	3.16M	489	ND	ND	1.16K	1.64M	5.19M	10.17M	-	-
	Ours (full)	1.4M	1	6	2	0	0	3.2M	1.38M	0.000571	0.9449
	QEC (60K)	60K	324	ND	ND	131	26.98K	0.86M	7.71M	0.000237	0.9271
	Ours (60K)	60K	1	6	0	0	0	0.74M	1.39M	0.000648	0.9420
	QEC (30K)	30K	265	ND	ND	112	13.53K	0.57M	6.95M	0.000437	0.9033
	Ours (30K)	30K	2	6	1	0	0	0.54M	1.39M	0.000636	0.9389
	QEC (10K)	10K	149	ND	ND	59	4.54K	0.3M	5.4M	0.000960	0.8613
	Ours (10K)	10K	1	6	0	0	0	0.32M	1.37M	0.000678	0.9253
T-Rex	CAD	1.23M	3368	ND	ND	333	2	35.22M	160.8M	-	-
	Ours (full)	1.39M	1	85	35	0	0	24.56M	81.53M	0.000475	0.9604
	QEC (60K)	60K	1247	ND	ND	1200	709	9.94M	142.9M	0.000074	0.9244
	Ours (60K)	60K	1	79	0	0	0	5.56M	81.58M	0.000698	0.9616
	QEC (30K)	30K	843	ND	ND	724	695	6.57M	129.5M	0.000140	0.9121
	Ours (30K)	30K	1	77	1	0	0	3.97M	81.1M	0.000691	0.9610
	QEC (10K)	10K	447	ND	ND	307	457	3.26M	102.3M	0.000296	0.9063
	Ours (10K)	10K	6	67	10	0	0	2.27M	76.58M	0.000752	0.9505

Table 4.3: Execution time in seconds per step.

Model	loose	tight	deci	other	total
Arm	135.77	846.11	29.09	33.77	1044.74
Cylinder	131.46	964.88	23.63	91.72	1211.68
Drone	94.65	909.42	26.67	58.90	1092.64
Engine	182.19	954.97	27.16	72.66	1236.98
T-Rex	94.79	1154.51	29.29	32.79	1311.38

Table 4.4: Average execution time of the *vertex projection* operation, the *remeshing* operation, the *remeshing post-processing* operations, and the *mesh smoothing* and *mesh sinking* operations.

Model	proj	remesh	post	smooth & sink
Arm	2.80	46.72	29.18	5.90
Cylinder	5.70	45.61	34.61	10.56
Drone	2.86	49.63	33.12	5.34
Engine	2.61	47.40	35.18	10.30
T-Rex	5.02	50.80	40.64	18.99

4.4.4 Execution time

In order to show the details of the execution time over the test cases, Table 4.3 shows the timing for each step of our proposal: *loose envelope generation*, *tight envelope generation*, and *mesh decimation*. Also, we show *pre-processing* (*point sampling* and *KD-Tree indexation*) and data transfer execution time.

We can see that the most time-consuming step is the *tight envelope generation*, followed by the *loose envelope generation*, which depends on the regular grid resolution. The *mesh decimation* step is similar in all cases because the tight envelopes are generated using the same parameters.

Because the *tight envelope generation* is the step that consumes more time, Table 4.4 shows the average execution time of the *vertex projection* operation, the *mesh reconstruction* operation (divided in *remeshing* and *remeshing post-processing*), and the *smoothing* and *sinking* operations (including the selection of vertices that are far from $\mathcal{M}$). As we can see, the *mesh reconstruction* operation is the bottleneck of our proposal.

4.5 Discussion

Our proposal simplifies the original mesh in different ways at the same time. Here, we discuss these ways and some advantages and disadvantages of our method.

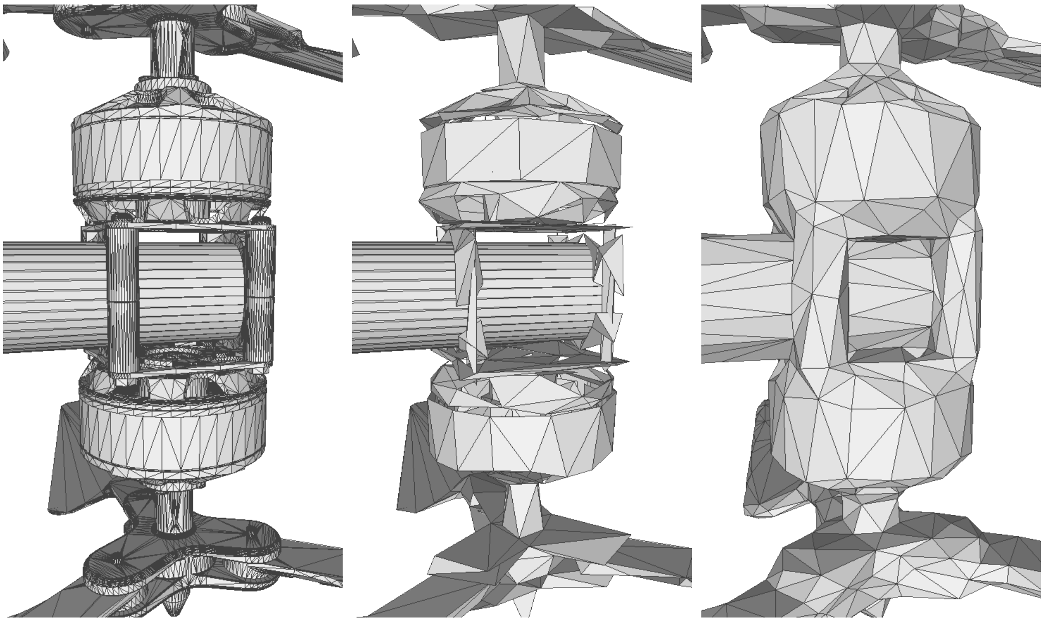


Figure 4.11: Comparison of the 30K results using the Drone model. The QEC decimation presents several artifacts while our method not. Left: CAD. Middle: QEC. Right: Ours.

4.5.1 All in one

An object can be made up of a set of components that can be independently designed and then assembled. All the CAD mesh models used in our experiments are conformed by several connected components, as shown in Table 4.2. Our method represents all the connected components by a single connected component, as shown in the tight envelope results. The latter occurs because the gaps between them have a certain proximity. Moreover, our method is capable of working on dense polygon soups or defective CAD mesh models (e.g., with boundaries).

In case the model has two separated pieces, for example, the *loose envelope generation* can split the initial contour into two connected components without any problem. It is an advantage of using a level set formulation. In the case the loose envelope remains as a single component, the membrane between both pieces can be removed by the deformation operations of the *tight envelope generation*.

4.5.2 Surface approximation

Since we use a fixed target number of vertices in the *remeshing* operation and the tight envelope tends to represent more surface through the iterations, salient features are smoothed or roughly approximated.

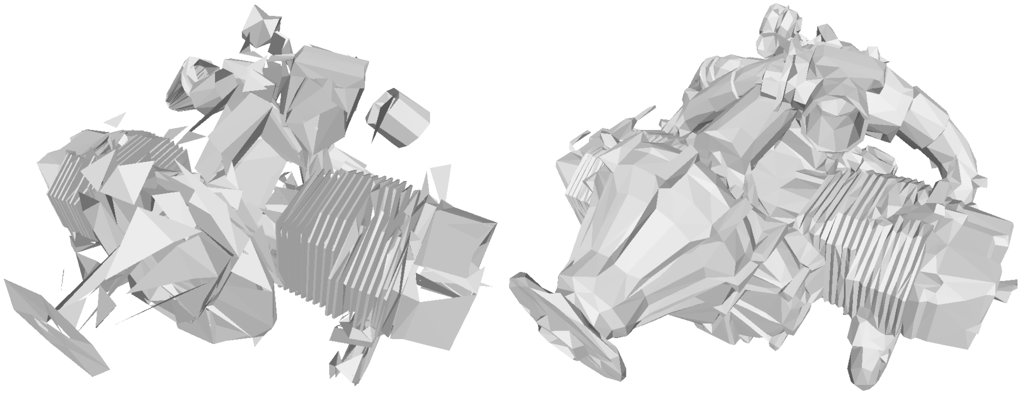


Figure 4.12: Comparison of the 10K results using the Engine model. The QEC decimation starts to collapse edges that should not, splitting the mesh in multiple connected components. Our result remains as a single connected component. Left: QEC. Right: Ours.

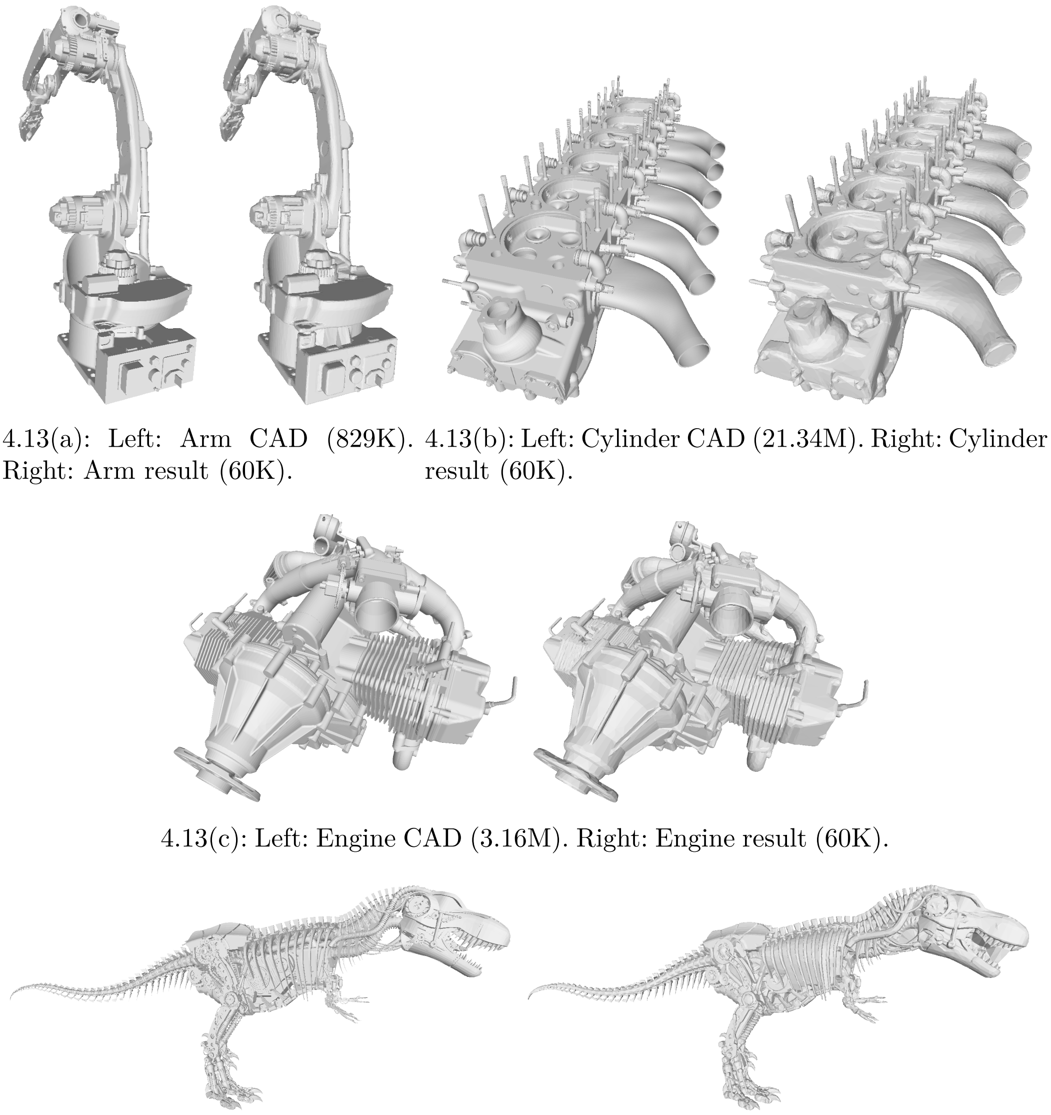
The inclusion of the bandwidth ϵ_t makes the vertices of $\mathcal{M}_e$ to be projected on a simpler surface than $\mathcal{M}$, omitting some features. Figure 4.11 shows an example of how the envelope's surface is simpler than the CAD model's surface.

Due that the decimation algorithm is based on a quadric error metric when the algorithm finishes the decimation of planar regions, it starts to collapse feature edges, which introduce a lower quadric error. For this reason, in addition to reducing the number of triangles, the algorithm tends to simplify the surface of the tight envelope, approximating the features.

4.5.3 Hidden surface

Several components of the original mesh can be hidden or partially hidden, considering multiple external points of view. The Cylinder model, for example, has internal parts that are occluded and not necessary for visualization and interaction. The envelope ignores all these components because the deformation can not go inside the model. The Arm model has some screws that are partially occluded. Their heads are visible while their bodies are occluded. The envelope approximates only their visible parts.

CAD models of sheet metal parts are widely used in manufacturing. Their corresponding triangular mesh representation considers the side that is not facing a possible external viewer. Our method approximates only the outer region.



4.13(a): Left: Arm CAD (829K). Right: Arm result (60K).

4.13(b): Left: Cylinder CAD (21.34M). Right: Cylinder result (60K).

4.13(c): Left: Engine CAD (3.16M). Right: Engine result (60K).

4.13(d): Left: T-Rex CAD (1.23M). Right: T-Rex result (60K).

Figure 4.13: CAD mesh models and decimated envelope meshes using our method. All envelopes contain 60K faces.

4.5.4 Gaps and holes

One of the operations of mesh defeaturing is the removal of through and blind holes. This kind of holes generate more surface area and make visible the internal surfaces. Our method can create a membrane to fill these holes or approximate them. The deformation of the membrane depends on how far it is from the bottom of a blind hole and the parameter a_n. If it is a through-hole, the surface of the membrane can be intersected, and as a consequence, the hole can be created, modifying the genus of the tight envelope.

Several CAD models have solid grids whose gaps also make visible the inner objects. Our method can generate a rough representation of a grid covering all the gaps. The ribs of the T-Rex model generate a kind of grid that our method uses to avoid the representation of internal components that can be ignored in the visualization.

4.5.5 Manifoldness

The *mesh reconstruction* and *decimation post-processing* operations return a 2-manifold mesh. This property is a requirement for several mesh processing algorithms, such that an automatic planar parameterization to generate texture coordinates can be efficiently applied. Also, this kind of mesh is easy to manipulate in a mesh editor. As shown in our experiments, in most cases, our results are watertight meshes, i.e., 2-manifold meshes without holes. Watertight meshes are widely used in shape analysis tasks that require mesh voxelization. Our method can be used to obtain watertight meshes from arbitrary CAD mesh models.

4.5.6 Comparison with related work

Our method works on CAD mesh models that can be generated from multiple CAD system representations. For example, there exist old models whose CAD system representation is not supported anymore, but its mesh model is available. Also, the CAD mesh model can have defects acquired during design or meshing processes.

We consider our envelope mesh as an impostor because it is not truly a simplification of the CAD mesh model's geometry. It is a wrapping geometry adapted to the outer shape of the model that tends to visually fool the viewer without disturbing a basic interaction required in XR applications.

Differently from volume quantization-based approaches [121, 122, 141, 142, 143, 144, 149, 152], our method uses a 3D grid just for the *loose envelope generation*, whose resolution can be low. For the *tight envelope generation*, the deformation of the envelope is purely based on a surface quantization that allows us to represent more surface details using less memory space.

Defeaturing methods [145, 146, 147, 107, 148] usually work on a specific domain of features and specific operations are proposed to suppress them. Our method fills gaps and approximates details on the surface adaptively without taking into account the feature class. Also, most of the defeaturing methods focus on finite element analysis, where features can be totally removed. We focus on visualization and interaction, so we prefer to keep the appearance of the features instead of complete suppression of them.

Vertex clustering [121, 122] and edge collapse decimation methods [124, 129, 134, 125, 130, 131, 132, 135, 126, 127, 133, 128] are commonly used for CAD mesh model simplification [165]. The vertex clustering methods require volume quantization and introduce several non-manifold simplices. Edge collapse methods suffers the phenomena shown in Figure 4.12 when they reach the limits of covering all the surface. Moreover, the method proposed in [134], suffers the latter when applied to models like the T-Rex, where the visibility is high for most of the surface.

In addition to the geometry simplification, our method drastically simplifies the topology of the CAD mesh model, as shown in Table 4.2. Decimation methods, such as the QEC decimation [124], are not capable of simplifying the topology in this way.

4.5.7 Limitations

Our method is sensitive to noise. If the original mesh has some floating triangles, some envelope vertices can be projected on a point within these triangles.

The bottleneck operation of our method is the *mesh reconstruction*. The *remeshing* algorithm generates regular meshes, but finding the CVT and computing its respective mesh requires high computational time. Also, since it is an isotropic algorithm, it can start removing sharp features if the parameter ϵ_t is not large enough. Closing mesh holes can be a costly task because when removing all the intersected triangles, several holes can be introduced.

We have to control the parameters n_{it_t}, n_{it_s}, and a_n, because they have major influence in the deformation process. High values of them can unnecessarily shrink some regions of the mesh.

4.6 Conclusion and future work

We propose an extreme mesh simplification method that approximates the geometry of the outer shape of the input, removing unnecessary features and non-visible regions. Our method focuses on XR applications that require a small model for real-time visualization and interaction. The resulting mesh can be easily post-processed since it is a 2-manifold mesh with low resolution.

The simplified mesh can be considered as a 3D polygonal impostor because it tries to fool the viewer by using less number of polygons and by approximating partial geometry of the input.

Our method is capable of simplifying meshes with complex geometry and topology, such as the meshes generated from CAD models, even if they are defective.

Our experiments show that when more iterations are used for the *tight envelope generation*, we obtain a better approximation of the original mesh. The target number of triangles used in our experiments (60K, 30K, and 10K) allows for efficient interaction even in a web environment. We evaluate the behavior of the method using geometric and visualization metrics, obtaining results that seem satisfactory for the viewer.

Our method took 1179 seconds on average to compute the decimated envelopes for our test cases. Although it can be considered a high computation time, the method is fully automated.

The generation of simplified models from CAD data has a strong demand for XR applications, especially when working on mobile devices. Our method is useful to automate this procedure or at least to generate an initial model for a low-poly design process. The method's computation time is not critical because it can be used in a pre-computation process.

As for future work, to reduce the computational time used to resample the surface, we can use a local remeshing algorithm. Furthermore, we can dynamically decimate the tight envelope during the deformation process to obtain an adaptive triangulation, allowing higher levels of refinement.

Also, we can test our method in other applications, such as the pre-processing of CAD mesh models used in data-driven shape analysis (e.g., [167]).

5
General conclusion

This thesis proposes a set of geometry processing algorithms to build 3D digital models from point clouds and previously constructed CAD models. These algorithms focus on two main tasks: denoising and simplification. Our experiments show that the proposed algorithms are competitive against state-of-the-art algorithms. We focus on digital models for interactive applications, such as XR applications.

The denoising algorithms generate piecewise smooth surfaces, minimizing the number of triangles required to approximate the entire object or environment. The simplification algorithm yields the outer surface of the target object, ignoring internal details that are irrelevant for most XR applications. Classic decimation algorithms, e.g., QEC, can easily control the mesh resolution.

The proposed denoising algorithms do not cover the entire construction pipeline. As future work, we aim to tackle other problems, such as outlier removal, mesh generation, and mesh decimation, to then integrate them into a single solution. Further, we can include data-driven algorithms for point cloud segmentation, object detection, and surface fitting.

The computational cost of the proposed algorithms is high, as recorded in the timing experiments. We aim to develop faster versions by replacing the most costly steps, such as anisotropic neighborhoods computation and remeshing. Also, since we introduced several parameters, we can design a set of hyperparameters that allow more straightforward and more intuitive interaction with the algorithms.

Bibliography

[1] TANG, Y. M.; HO, H. L.. **3d modeling and computer graphics in virtual reality**. In: MIXED REALITY AND THREE-DIMENSIONAL COMPUTER GRAPHICS. IntechOpen, 2020.

[2] DANESHMAND, M.; HELMI, A.; AVOTS, E.; NOROOZI, F.; ALISINANOGLU, F.; ARSLAN, H. S.; GORBOVA, J.; HAAMER, R. E.; OZCINAR, C. ; ANBARJAFARI, G.. **3d scanning: A comprehensive survey**. arXiv preprint arXiv:1801.08863, 2018.

[3] SCHÜTZ, M.; KRÖSL, K. ; WIMMER, M.. **Real-time continuous level of detail rendering of point clouds**. In: 2019 IEEE CONFERENCE ON VIRTUAL REALITY AND 3D USER INTERFACES (VR), p. 103–110. IEEE, 2019.

[4] BERGER, M.; TAGLIASACCHI, A.; SEVERSKY, L. M.; ALLIEZ, P.; GUENNEBAUD, G.; LEVINE, J. A.; SHARF, A. ; SILVA, C. T.. **A survey of surface reconstruction from point clouds**. In: COMPUTER GRAPHICS FORUM, volumen 36, p. 301–329. Wiley Online Library, 2017.

[5] MAGLO, A.; LAVOUÉ, G.; DUPONT, F. ; HUDELOT, C.. **3d mesh compression: Survey, comparisons, and emerging trends**. ACM Computing Surveys (CSUR), 47(3):1–41, 2015.

[6] MEDEROS, B.; VELHO, L. ; DE FIGUEIREDO, L. H.. **Robust smoothing of noisy point clouds**. In: PROC. SIAM CONFERENCE ON GEOMETRIC DESIGN AND COMPUTING, volumen 2004, p. 2. Citeseer, 2003.

[7] FLEISHMAN, S.; COHEN-OR, D. ; SILVA, C. T.. **Robust moving least-squares fitting with sharp features**. ACM transactions on graphics (TOG), 24(3):544–552, 2005.

[8] GUENNEBAUD, G.; GROSS, M.. **Algebraic point set surfaces**. In: ACM SIGGRAPH 2007 PAPERS, p. 23–es. 2007.

[9] GUENNEBAUD, G.; GERMANN, M. ; GROSS, M.. **Dynamic sampling and rendering of algebraic point set surfaces**. In: COMPUTER GRAPHICS FORUM, volumen 27, p. 653–662. Wiley Online Library, 2008.

[10] ÖZTIRELI, A. C.; GUENNEBAUD, G. ; GROSS, M.. **Feature preserving point set surfaces based on non-linear kernel regression**. In: COMPUTER GRAPHICS FORUM, volumen 28, p. 493–501. Wiley Online Library, 2009.

[11] WEBER, C.; HAHMANN, S.; HAGEN, H. ; BONNEAU, G.-P.. **Sharp feature preserving mls surface reconstruction based on local feature line approximations**. Graphical Models, 74(6):335–345, 2012.

[12] AVRON, H.; SHARF, A.; GREIF, C. ; COHEN-OR, D.. **ℓ_1-sparse reconstruction of sharp point set surfaces**. ACM Transactions on Graphics (TOG), 29(5):1–12, 2010.

[13] LEAL, E.; SANCHEZ-TORRES, G. ; BRANCH, J. W.. **Sparse regularization-based approach for point cloud denoising and sharp features enhancement**. Sensors, 20(11):3206, 2020.

[14] SUN, Y.; SCHAEFER, S. ; WANG, W.. **Denoising point sets via l0 minimization**. Computer Aided Geometric Design, 35:2–15, 2015.

[15] MATTEI, E.; CASTRODAD, A.. **Point cloud denoising via moving rpca**. In: COMPUTER GRAPHICS FORUM, volumen 36, p. 123–137. Wiley Online Library, 2017.

[16] CHEN, H.; WEI, M.; SUN, Y.; XIE, X. ; WANG, J.. **Multi-patch collaborative point cloud denoising via low-rank recovery with graph constraint**. IEEE transactions on visualization and computer graphics, 26(11):3255–3270, 2019.

[17] BUADES, A.; COLL, B. ; MOREL, J.-M.. **A non-local algorithm for image denoising**. In: 2005 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'05), volumen 2, p. 60–65. IEEE, 2005.

[18] DABOV, K.; FOI, A.; KATKOVNIK, V. ; EGIAZARIAN, K.. **Image denoising by sparse 3-d transform-domain collaborative filtering**. IEEE Transactions on image processing, 16(8):2080–2095, 2007.

[19] DESCHAUD, J.-E.; GOULETTE, F.. **Point cloud non local denoising using local surface descriptor similarity**. 2010.

[20] DIGNE, J.. **Similarity based filtering of point clouds**. In: 2012 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION WORKSHOPS, p. 73–79. IEEE, 2012.

[21] GUILLEMOT, T.; ALMANSA, A. ; BOUBEKEUR, T.. **Non local point set surfaces.** In: 2012 SECOND INTERNATIONAL CONFERENCE ON 3D IMAGING, MODELING, PROCESSING, VISUALIZATION & TRANSMISSION, p. 324–331. IEEE, 2012.

[22] SARKAR, K.; BERNARD, F.; VARANASI, K.; THEOBALT, C. ; STRICKER, D.. **Structured low-rank matrix factorization for point-cloud denoising.** In: 2018 INTERNATIONAL CONFERENCE ON 3D VISION (3DV), p. 444–453. IEEE, 2018.

[23] DIGNE, J.; VALETTE, S. ; CHAINE, R.. **Sparse geometric representation through local shape probing.** IEEE Transactions on Visualization and Computer Graphics, 24(7):2238–2250, 2017.

[24] LU, X.; SCHAEFER, S.; LUO, J.; MA, L. ; HE, Y.. **Low rank matrix approximation for 3d geometry filtering.** IEEE Transactions on Visualization and Computer Graphics, 2020.

[25] ROSMAN, G.; DUBROVINA, A. ; KIMMEL, R.. **Patch-collaborative spectral point-cloud denoising.** In: COMPUTER GRAPHICS FORUM, volumen 32, p. 1–12. Wiley Online Library, 2013.

[26] SCHOENENBERGER, Y.; PARATTE, J. ; VANDERGHEYNST, P.. **Graph-based denoising for time-varying point clouds.** In: 2015 3DTV-CONFERENCE: THE TRUE VISION-CAPTURE, TRANSMISSION AND DISPLAY OF 3D VIDEO (3DTV-CON), p. 1–4. IEEE, 2015.

[27] DINESH, C.; CHEUNG, G.; BAJIĆ, I. V. ; YANG, C.. **Local 3d point cloud denoising via bipartite graph approximation & total variation.** In: 2018 IEEE 20TH INTERNATIONAL WORKSHOP ON MULTIMEDIA SIGNAL PROCESSING (MMSP), p. 1–6. IEEE, 2018.

[28] DINESH, C.; CHEUNG, G. ; BAJIĆ, I. V.. **Point cloud denoising via feature graph laplacian regularization.** IEEE Transactions on Image Processing, 29:4143–4158, 2020.

[29] ZENG, J.; CHEUNG, G.; NG, M.; PANG, J. ; YANG, C.. **3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model.** IEEE Transactions on Image Processing, 29:3474–3489, 2019.

[30] HU, W.; GAO, X.; CHEUNG, G. ; GUO, Z.. **Feature graph learning for 3d point cloud denoising.** IEEE Transactions on Signal Processing, 68:2841–2856, 2020.

[31] HU, G.; PENG, Q. ; FORREST, A. R.. **Mean shift denoising of point-sampled surfaces**. The Visual Computer, 22(3):147–157, 2006.

[32] WANG, J.; XU, K.; LIU, L.; CAO, J.; LIU, S.; YU, Z. ; GU, X. D.. **Consolidation of low-quality point clouds from outdoor scenes**. In: COMPUTER GRAPHICS FORUM, volumen 32, p. 207–216. Wiley Online Library, 2013.

[33] ZHENG, Y.; LI, G.; WU, S.; LIU, Y. ; GAO, Y.. **Guided point cloud denoising via sharp feature skeletons**. The Visual Computer, 33(6-8):857–867, 2017.

[34] ZHENG, Y.; LI, G.; XU, X.; WU, S. ; NIE, Y.. **Rolling normal filtering for point clouds**. Computer Aided Geometric Design, 62:16–28, 2018.

[35] YADAV, S. K.; REITEBUCH, U.; SKRODZKI, M.; ZIMMERMANN, E. ; POLTHIER, K.. **Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics**. Computers & Graphics, 74:234–243, 2018.

[36] LIU, Z.; XIAO, X.; ZHONG, S.; WANG, W.; LI, Y.; ZHANG, L. ; XIE, Z.. **A feature-preserving framework for point cloud denoising**. Computer-Aided Design, p. 102857, 2020.

[37] BÉARZI, Y.; DIGNE, J. ; CHAINE, R.. **Wavejets: A local frequency framework for shape details amplification**. In: COMPUTER GRAPHICS FORUM, volumen 37, p. 13–24. Wiley Online Library, 2018.

[38] LIPMAN, Y.; COHEN-OR, D.; LEVIN, D. ; TAL-EZER, H.. **Parameterization-free projection for geometry reconstruction**. ACM Transactions on Graphics (TOG), 26(3):22–es, 2007.

[39] HUANG, H.; LI, D.; ZHANG, H.; ASCHER, U. ; COHEN-OR, D.. **Consolidation of unorganized point clouds for surface reconstruction**. ACM transactions on graphics (TOG), 28(5):1–7, 2009.

[40] HUANG, H.; WU, S.; GONG, M.; COHEN-OR, D.; ASCHER, U. ; ZHANG, H.. **Edge-aware point set resampling**. ACM transactions on graphics (TOG), 32(1):1–12, 2013.

[41] PREINER, R.; MATTAUSCH, O.; ARIKAN, M.; PAJAROLA, R. ; WIMMER, M.. **Continuous projection for fast l1 reconstruction**. ACM Trans. Graph., 33(4):47–1, 2014.

[42] WU, S.; HUANG, H.; GONG, M.; ZWICKER, M. ; COHEN-OR, D.. **Deep points consolidation**. ACM Transactions on Graphics (ToG), 34(6):1–13, 2015.

[43] LU, X.; WU, S.; CHEN, H.; YEUNG, S.-K.; CHEN, W. ; ZWICKER, M.. **Gpf: Gmm-inspired feature-preserving point set filtering**. IEEE transactions on visualization and computer graphics, 24(8):2315–2326, 2017.

[44] BOULCH, A.; MARLET, R.. **Deep learning for robust normal estimation in unstructured point clouds**. In: COMPUTER GRAPHICS FORUM, volumen 35, p. 281–290. Wiley Online Library, 2016.

[45] GUERRERO, P.; KLEIMAN, Y.; OVSJANIKOV, M. ; MITRA, N. J.. **Pcpnet learning local shape properties from raw point clouds**. In: COMPUTER GRAPHICS FORUM, volumen 37, p. 75–85. Wiley Online Library, 2018.

[46] ROVERI, R.; ÖZTIRELI, A. C.; PANDELE, I. ; GROSS, M.. **Pointpronets: Consolidation of point clouds with convolutional neural networks**. In: COMPUTER GRAPHICS FORUM, volumen 37, p. 87–99. Wiley Online Library, 2018.

[47] BEN-SHABAT, Y.; LINDENBAUM, M. ; FISCHER, A.. **Nesti-net: Normal estimation for unstructured 3d point clouds using convolutional neural networks**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 10112–10120, 2019.

[48] DUAN, C.; CHEN, S. ; KOVACEVIC, J.. **3d point cloud denoising via deep neural network based local surface estimation**. In: ICASSP 2019-2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), p. 8553–8557. IEEE, 2019.

[49] HERMOSILLA, P.; RITSCHEL, T. ; ROPINSKI, T.. **Total denoising: Unsupervised learning of 3d point cloud cleaning**. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, p. 52–60, 2019.

[50] RAKOTOSAONA, M.-J.; LA BARBERA, V.; GUERRERO, P.; MITRA, N. J. ; OVSJANIKOV, M.. **Pointcleannet: Learning to denoise**

and remove outliers from dense point clouds. In: COMPUTER GRAPHICS FORUM, volumen 39, p. 185–203. Wiley Online Library, 2020.

[51] LUO, S.; HU, W.. **Differentiable manifold reconstruction for point cloud denoising**. In: PROCEEDINGS OF THE 28TH ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, p. 1330–1338, 2020.

[52] YU, L.; LI, X.; FU, C.-W.; COHEN-OR, D. ; HENG, P.-A.. **Ec-net: an edge-aware point set consolidation network**. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), p. 386–402, 2018.

[53] LU, D.; LU, X.; SUN, Y. ; WANG, J.. **Deep feature-preserving normal estimation for point cloud filtering**. Computer-Aided Design, p. 102860, 2020.

[54] WEI, M.; CHEN, H.; ZHANG, Y.; XIE, H.; GUO, Y. ; WANG, J.. **Geodualcnn: Geometry-supporting dual convolutional neural network for noisy point clouds**. IEEE Transactions on Visualization and Computer Graphics, 2021.

[55] HURTADO, J.; GATTASS, M.; RAPOSO, A. ; COELHO, J.. **Adaptive patches for mesh denoising**. In: 2018 31ST SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES (SIBGRAPI), p. 1–8, 2018.

[56] HURTADO, J.. **Detail-preserving mesh denoising using adaptive patches**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Brazil, 2018.

[57] ZHANG, J.; CAO, J.; LIU, X.; CHEN, H.; LI, B. ; LIU, L.. **Multi-normal estimation via pair consistency voting**. IEEE transactions on visualization and computer graphics, 25(4):1693–1706, 2018.

[58] THOMPSON, E. M.; BIASOTTI, S.; GIACHETTI, A.; TORTORICI, C.; WERGHI, N.; OBEID, A. S.; BERRETTI, S.; NGUYEN-DINH, H.-P.; LE, M.-Q.; NGUYEN, H.-D. ; OTHERS. **Shrec 2020: Retrieval of digital surfaces with similar geometric reliefs**. Computers & Graphics, 91:199–218, 2020.

[59] CIGNONI, P.; CALLIERI, M.; CORSINI, M.; DELLEPIANE, M.; GANOVELLI, F. ; RANZUGLIA, G.. **MeshLab: an Open-Source Mesh Processing Tool**. In: Scarano, V.; Chiara, R. D. ; Erra, U., editors, EUROGRAPHICS ITALIAN CHAPTER CONFERENCE. The Eurographics Association, 2008.

[60] HOPPE, H.; DEROSE, T.; DUCHAMP, T.; MCDONALD, J. ; STUETZLE, W.. **Surface reconstruction from unorganized points**. In: PROCEEDINGS OF THE 19TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 71–78, 1992.

[61] CAZALS, F.; POUGET, M.. **Estimating differential quantities using polynomial fitting of osculating jets**. Computer Aided Geometric Design, 22(2):121–146, 2005.

[62] MÉRIGOT, Q.; OVSJANIKOV, M. ; GUIBAS, L. J.. **Voronoi-based curvature and feature estimation from point clouds**. IEEE Transactions on Visualization and Computer Graphics, 17(6):743–756, 2010.

[63] THE CGAL PROJECT. **CGAL User and Reference Manual**. CGAL Editorial Board, 5.2.1 edition, 2021.

[64] BAZAZIAN, D.; CASAS, J. R. ; RUIZ-HIDALGO, J.. **Fast and robust edge extraction in unorganized point clouds**. In: 2015 INTERNATIONAL CONFERENCE ON DIGITAL IMAGE COMPUTING: TECHNIQUES AND APPLICATIONS (DICTA), p. 1–8. IEEE, 2015.

[65] CPLEX IBM ILOG. **ILOG CPLEX Optimization Studio 20.1: User's manual for CPLEX**, 2020.

[66] BERNARDINI, F.; MITTLEMAN, J.; RUSHMEIER, H.; SILVA, C. ; TAUBIN, G.. **The ball-pivoting algorithm for surface reconstruction**. IEEE transactions on visualization and computer graphics, 5(4):349–359, 1999.

[67] MEDEROS, B.; AMENTA, N.; VELHO, L. ; DE FIGUEIREDO, L. H.. **Surface reconstruction for noisy point clouds**. In: SYMPOSIUM ON GEOMETRY PROCESSING, p. 53–62. Citeseer, 2005.

[68] OHTAKE, Y.; BELYAEV, A. G. ; BOGAEVSKI, I. A.. **Polyhedral surface smoothing with simultaneous mesh regularization**. In: GEOMETRIC MODELING AND PROCESSING 2000. THEORY AND APPLICATIONS. PROCEEDINGS, p. 229–237. IEEE, 2000.

[69] DESBRUN, M.; MEYER, M.; SCHRÖDER, P. ; BARR, A. H.. **Anisotropic feature-preserving denoising of height fields and bivariate data**. In: GRAPHICS INTERFACE, volumen 11, p. 145–152. Citeseer, 2000.

[70] CLARENZ, U.; DIEWALD, U. ; RUMPF, M.. **Anisotropic geometric diffusion in surface processing**. In: PROCEEDINGS OF THE CONFERENCE ON VISUALIZATION'00, p. 397–405. IEEE Computer Society Press, 2000.

[71] BAJAJ, C. L.; XU, G.. **Anisotropic diffusion of surfaces and functions on surfaces**. ACM Transactions on Graphics (TOG), 22(1):4–32, 2003.

[72] EL OUAFDI, A. F.; ZIOU, D.. **A global physical method for manifold smoothing**. In: 2008 IEEE INTERNATIONAL CONFERENCE ON SHAPE MODELING AND APPLICATIONS, 2008.

[73] HILDEBRANDT, K.; POLTHIER, K.. **Anisotropic filtering of non-linear surface features**. In: COMPUTER GRAPHICS FORUM, volumen 23, p. 391–400. Wiley Online Library, 2004.

[74] HE, L.; SCHAEFER, S.. **Mesh denoising via l 0 minimization**. ACM Transactions on Graphics (TOG), 32(4):64, 2013.

[75] FLEISHMAN, S.; DRORI, I. ; COHEN-OR, D.. **Bilateral mesh denoising**. In: ACM TRANSACTIONS ON GRAPHICS (TOG), volumen 22, p. 950–953. ACM, 2003.

[76] JONES, T. R.; DURAND, F. ; DESBRUN, M.. **Non-iterative, feature-preserving mesh smoothing**. In: ACM TRANSACTIONS ON GRAPHICS (TOG), volumen 22, p. 943–949. ACM, 2003.

[77] SOLOMON, J.; CRANE, K.; BUTSCHER, A. ; WOJTAN, C.. **A general framework for bilateral and mean shift filtering**. CoRR, abs/1405.4734, 2014.

[78] TAUBIN, G.. **Linear anisotropic mesh filtering**. Res. Rep. RC2213 IBM, 1(4), 2001.

[79] SHEN, Y.; BARNER, K. E.. **Fuzzy vector median-based surface smoothing**. IEEE Transactions on Visualization and Computer Graphics, 10(3):252–265, May 2004.

[80] SUN, X.; ROSIN, P.; MARTIN, R. ; LANGBEIN, F.. **Fast and effective feature-preserving mesh denoising**. IEEE transactions on visualization and computer graphics, 13(5):925–938, 2007.

[81] ZHENG, Y.; FU, H.; AU, O. K.-C. ; TAI, C.-L.. **Bilateral normal filtering for mesh denoising**. IEEE Transactions on Visualization and Computer Graphics, 17(10):1521–1530, 2011.

[82] WEI, M.; YU, J.; PANG, W.-M.; WANG, J.; QIN, J.; LIU, L. ; HENG, P.-A.. **Bi-normal filtering for mesh denoising**. IEEE transactions on visualization and computer graphics, 21(1):43–55, 2015.

[83] ZHANG, W.; DENG, B.; ZHANG, J.; BOUAZIZ, S. ; LIU, L.. **Guided mesh normal filtering**. In: COMPUTER GRAPHICS FORUM, volumen 34, p. 23–34. Wiley Online Library, 2015.

[84] LI, T.; WANG, J.; LIU, H. ; LIU, L.-G.. **Efficient mesh denoising via robust normal filtering and alternate vertex updating**. Frontiers of Information Technology & Electronic Engineering, 18(11):1828–1842, 2017.

[85] YADAV, S. K.; REITEBUCH, U. ; POLTHIER, K.. **Mesh denoising based on normal voting tensor and binary optimization**. IEEE Transactions on Visualization & Computer Graphics, (8):2366–2379, 2018.

[86] YADAV, S. K.; REITEBUCH, U. ; POLTHIER, K.. **Robust and high fidelity mesh denoising**. IEEE transactions on visualization and computer graphics, 25(6):2304–2310, 2018.

[87] WEI, M.; LIANG, L.; PANG, W.-M.; WANG, J.; LI, W. ; WU, H.. **Tensor voting guided mesh denoising**. IEEE Transactions on Automation Science and Engineering, 14(2):931–945, 2017.

[88] GUO, M.; SONG, Z.; HAN, C.; ZHONG, S.; LV, R. ; LIU, Z.. **Mesh denoising via adaptive consistent neighborhood**. Sensors, 21(2):412, 2021.

[89] ZHONG, S.; SONG, Z.; LIU, Z.; XIE, Z.; CHEN, J.; LIU, L. ; CHEN, R.. **Shape-aware mesh normal filtering**. Computer-Aided Design, 140:103088, 2021.

[90] WANG, Y.; YANG, Y. ; LIU, Q.. **Feature-aware trilateral filter with energy minimization for 3d mesh denoising**. IEEE Access, 8:52232–52244, 2020.

[91] WANG, P.-S.; LIU, Y. ; TONG, X.. **Mesh denoising via cascaded normal regression**. ACM Trans. Graph., 35(6):232–1, 2016.

[92] WANG, J.; HUANG, J.; WANG, F. L.; WEI, M.; XIE, H. ; QIN, J.. **Data-driven geometry-recovering mesh denoising**. Computer-Aided Design, 114:133–142, 2019.

[93] LI, X.; LI, R.; ZHU, L.; FU, C.-W. ; HENG, P.-A.. **Dnf-net: A deep normal filtering network for mesh denoising**. IEEE Transactions on Visualization and Computer Graphics, 2020.

[94] ARVANITIS, G.; LALOS, A. ; MOUSTAKAS, K.. **Feature-aware and content-wise denoising of 3d static and dynamic meshes using deep autoencoders**. In: 2019 IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME), p. 97–102. IEEE, 2019.

[95] ZHAO, W.; LIU, X.; ZHAO, Y.; FAN, X. ; ZHAO, D.. **Normalnet: Learning-based normal filtering for mesh denoising**. arXiv preprint arXiv:1903.04015, 2019.

[96] ARVANITIS, G.; LALOS, A. S. ; MOUSTAKAS, K.. **Image-based 3d mesh denoising through a block matching 3d convolutional neural network filtering approach**. In: 2020 IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME), p. 1–6. IEEE, 2020.

[97] NOUSIAS, S.; ARVANITIS, G.; LALOS, A. S. ; MOUSTAKAS, K.. **Fast mesh denoising with data driven normal filtering using deep variational autoencoders**. IEEE Transactions on Industrial Informatics, 17(2):980–990, 2020.

[98] HURTADO, J.; MONTENEGRO, A.; GATTASS, M.; CARVALHO, F. ; RAPOSO, A.. **Enveloping cad models for visualization and interaction in xr applications**. Engineering with Computers, p. 1–19, 2020.

[99] RAPOSO, A.; CORSEUIL, E. T.; WAGNER, G. N.; DOS SANTOS, I. H. ; GATTASS, M.. **Towards the use of cad models in vr applications**. In: PROCEEDINGS OF THE 2006 ACM INTERNATIONAL CONFERENCE ON VIRTUAL REALITY CONTINUUM AND ITS APPLICATIONS, p. 67–74. ACM, 2006.

[100] KIM, S.; LEE, K.; HONG, T.; KIM, M.; JUNG, M. ; SONG, Y.. **An integrated approach to realize multi-resolution of b-rep model**. In: PROCEEDINGS OF THE 2005 ACM SYMPOSIUM ON SOLID AND PHYSICAL MODELING, p. 153–162. ACM, 2005.

[101] KIM, B. C.; MUN, D.. **Stepwise volume decomposition for the modification of b-rep models**. The International Journal of Advanced Manufacturing Technology, 75(9-12):1393–1403, 2014.

[102] CHEN, J.; CAO, B.; ZHENG, Y.; XIE, L.; LI, C. ; XIAO, Z.. **Automatic surface repairing, defeaturing and meshing algorithms based on an extended b-rep**. Advances in Engineering Software, 86:55–69, 2015.

[103] YOON, Y.; KIM, B. C.. **Cad model simplification using feature simplifications**. Journal of Advanced Mechanical Design, Systems, and Manufacturing, 10(8):JAMDSM0099–JAMDSM0099, 2016.

[104] MUN, D.; KIM, B. C.. **Extended progressive simplification of feature-based cad models**. The International Journal of Advanced Manufacturing Technology, 93(1-4):915–932, 2017.

[105] KWON, S.; MUN, D.; KIM, B. C. ; HAN, S.. **Feature shape complexity: a new criterion for the simplification of feature-based 3d cad models**. The International Journal of Advanced Manufacturing Technology, 88(5-8):1831–1843, 2017.

[106] CHOW, P.; KUBOTA, T. ; GEORGESCU, S.. **Automatic detection of geometric features in cad models by characteristics**. Computer-Aided Design and Applications, 12(6):784–793, 2015.

[107] GAO, S.; ZHAO, W.; LIN, H.; YANG, F. ; CHEN, X.. **Feature suppression based cad mesh model simplification**. Computer-Aided Design, 42(12):1178–1188, 2010.

[108] GONZÁLEZ-LLUCH, C.; COMPANY, P.; CONTERO, M.; CAMBA, J. D. ; PLUMED, R.. **A survey on 3d cad model quality assurance and testing tools**. Computer-Aided Design, 83:64–79, 2017.

[109] MACIEL, P. W.; SHIRLEY, P.. **Visual navigation of large environments using textured clusters**. In: PROCEEDINGS OF THE 1995 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 95–ff. ACM, 1995.

[110] SILLION, F.; DRETTAKIS, G. ; BODELET, B.. **Efficient impostor manipulation for real-time visualization of urban scenery**. In: COMPUTER GRAPHICS FORUM, volumen 16, p. C207–C218. Wiley Online Library, 1997.

[111] DÉCORET, X.; SILLION, F.; SCHAUFLER, G. ; DORSEY, J.. **Multi-layered impostors for accelerated rendering**. In: COMPUTER GRAPHICS FORUM, volumen 18, p. 61–73. Wiley Online Library, 1999.

[112] FORSYTH, T.. **Impostors: Adding clutter**. In: DeLoura, M., editor, GAME PROGRAMMING GEMS 2, Charles River Media programming. Charles River Media, 2001.

[113] DÉCORET, X.; DURAND, F.; SILLION, F. X. ; DORSEY, J.. **Billboard clouds for extreme model simplification**. ACM Trans. Graph., 22(3):689–696, July 2003.

[114] POLICARPO, F.; OLIVEIRA, M. M.. **Relief mapping of non-height-field surface details**. In: PROCEEDINGS OF THE 2006 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, p. 55–62. ACM, 2006.

[115] RISSER, E.. **True imposters**. In: SIGGRAPH RESEARCH POSTERS, p. 58, 2006.

[116] ANDÚJAR, C.; BOO, J.; BRUNET, P.; FAIRÉN, M.; NAVAZO, I.; VAZQUEZ, P. ; VINACUA, A.. **Omni-directional relief impostors**. In: COMPUTER GRAPHICS FORUM, volumen 26, p. 553–560. Wiley Online Library, 2007.

[117] KOBBELT, L.; BOTSCH, M.. **A survey of point-based techniques in computer graphics**. Computers & Graphics, 28(6):801–814, 2004.

[118] WIMMER, M.; WONKA, P. ; SILLION, F.. **Point-based impostors for real-time visualization**. In: RENDERING TECHNIQUES 2001, p. 163–176. Springer, 2001.

[119] AKENINE-MOLLER, T.; HAINES, E. ; HOFFMAN, N.. **Real-time rendering**. AK Peters/CRC Press, 2018.

[120] BOTSCH, M.; KOBBELT, L.; PAULY, M.; ALLIEZ, P. ; LÉVY, B.. **Polygon mesh processing**. AK Peters/CRC Press, 2010.

[121] ROSSIGNAC, J.; BORREL, P.. **Multi-resolution 3d approximations for rendering complex scenes**. In: MODELING IN COMPUTER GRAPHICS, p. 455–465. Springer, 1993.

[122] LOW, K.-L.; TAN, T.-S.. **Model simplification using vertex-clustering**. In: PROCEEDINGS OF THE 1997 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, p. 75–ff. ACM, 1997.

[123] SCHROEDER, W. J.; ZARGE, J. A. ; LORENSEN, W. E.. **Decimation of triangle meshes**. SIGGRAPH Comput. Graph., 26(2):65–70, July 1992.

[124] GARLAND, M.; HECKBERT, P. S.. **Surface simplification using quadric error metrics**. In: PROCEEDINGS OF THE 24TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[125] DYER, R.; ZHANG, H. ; MÖLLER, T.. **Delaunay mesh construction**. In: PROCEEDINGS OF THE FIFTH EUROGRAPHICS SYMPOSIUM ON GEOMETRY PROCESSING, SGP '07, p. 273–282, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[126] LIU, Y.-J.; XU, C.-X.; FAN, D. ; HE, Y.. **Efficient construction and simplification of delaunay meshes**. ACM Transactions on Graphics (TOG), 34(6):174, 2015.

[127] SALINAS, D.; LAFARGE, F. ; ALLIEZ, P.. **Structure-aware mesh decimation**. Comput. Graph. Forum, 34(6):211–227, Sept. 2015.

[128] YI, R.; LIU, Y.-J. ; HE, Y.. **Delaunay mesh simplification with differential evolution**. In: SIGGRAPH ASIA 2018 TECHNICAL PAPERS, p. 263. ACM, 2018.

[129] LINDSTROM, P.; TURK, G.. **Image-driven simplification**. ACM Transactions on Graphics (ToG), 19(3):204–241, 2000.

[130] QU, L.; MEYER, G. W.. **Perceptually guided polygon reduction**. IEEE Transactions on Visualization and Computer Graphics, 14(5):1015–1029, 2008.

[131] CASTELLÓ, P.; SBERT, M.; CHOVER, M. ; FEIXAS, M.. **Viewpoint-based simplification using f-divergences**. Information Sciences, 178(11):2375–2388, 2008.

[132] CASTELLÓ, P.; SBERT, M.; CHOVER, M. ; FEIXAS, M.. **Viewpoint-driven simplification using mutual information**. Computers & Graphics, 32(4):451–463, 2008.

[133] NADER, G.; WANG, K.; HÉTROY-WHEELER, F. ; DUPONT, F.. **Visual contrast sensitivity and discrimination for 3d meshes and their applications**. Comput. Graph. Forum, 35(7):497–506, Oct. 2016.

[134] ZHANG, E.; TURK, G.. **Visibility-guided simplification**. In: PROCEEDINGS OF THE CONFERENCE ON VISUALIZATION'02, p. 267–274. IEEE Computer Society, 2002.

[135] CASTELLÓ, P.; CHOVER, M.; SBERT, M. ; FEIXAS, M.. **Reducing complexity in polygonal meshes with view-based saliency**. Computer Aided Geometric Design, 31(6):279–293, 2014.

[136] YAN, D.-M.; LÉVY, B.; LIU, Y.; SUN, F. ; WANG, W.. **Isotropic remeshing with fast and exact computation of restricted voronoi diagram**. In: PROCEEDINGS OF THE SYMPOSIUM ON GEOMETRY PROCESSING, SGP '09, p. 1445–1454, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[137] GUO, J.; YAN, D.-M.; JIA, X. ; ZHANG, X.. **Efficient maximal poisson-disk sampling and remeshing on surfaces**. Computers & Graphics, 46:72–79, 2015.

[138] HU, K.; YAN, D.-M.; BOMMES, D.; ALLIEZ, P. ; BENES, B.. **Error-bounded and feature preserving surface remeshing with minimal angle improvement**. IEEE transactions on visualization and computer graphics, 23(12):2560–2573, 2016.

[139] ABDELKADER, A.; MAHMOUD, A. H.; RUSHDI, A. A.; MITCHELL, S. A.; OWENS, J. D. ; EBEIDA, M. S.. **A constrained resampling strategy for mesh improvement**. Comput. Graph. Forum, 36(5):189–201, Aug. 2017.

[140] COHEN-STEINER, D.; ALLIEZ, P. ; DESBRUN, M.. **Variational shape approximation**. ACM Trans. Graph., 23(3):905–914, Aug. 2004.

[141] HE, T.; HONG, L.; KAUFMAN, A.; VARSHNEY, A. ; WANG, S.. **Voxel based object simplification**. In: PROCEEDINGS OF THE 6TH CONFERENCE ON VISUALIZATION'95, p. 296. IEEE Computer Society, 1995.

[142] ANDÚJAR, C.; BRUNET, P. ; AYALA, D.. **Topology-reducing surface simplification using a discrete solid representation**. ACM Transactions on Graphics (TOG), 21(2):88–105, 2002.

[143] NOORUDDIN, F. S.; TURK, G.. **Simplification and repair of polygonal models using volumetric techniques**. IEEE Transactions on Visualization and Computer Graphics, 9(2):191–205, 2003.

[144] LEE, S. H.. **A cad–cae integration approach using feature-based multi-resolution and multi-abstraction modelling techniques**. Computer-Aided Design, 37(9):941–955, 2005.

[145] DEY, S.; SHEPHARD, M. S. ; GEORGES, M. K.. **Elimination of the adverse effects of small model features by the local modification of automatically generated meshes.** Engineering with Computers, 13(3):134–152, 1997.

[146] JANG, J.; WONKA, P.; RIBARSKY, W. ; SHAW, C. D.. **Punctuated simplification of man-made objects.** The Visual Computer, 22(2):136–145, 2006.

[147] SUNIL, V.; PANDE, S.. **Automatic recognition of features from freeform surface cad models.** Computer-Aided Design, 40(4):502–517, 2008.

[148] QUADROS, W. R.; OWEN, S. J.. **Defeaturing cad models using a geometry-based size field and facet-based reduction operators.** In: PROCEEDINGS OF THE 18TH INTERNATIONAL MESHING ROUNDTABLE, p. 301–318. Springer, 2009.

[149] HUANG, P.; WANG, C. C.. **Volume and complexity bounded simplification of solid model represented by binary space partition.** In: PROCEEDINGS OF THE 14TH ACM SYMPOSIUM ON SOLID AND PHYSICAL MODELING, p. 177–182. ACM, 2010.

[150] BITTNER, J.; WONKA, P.. **Visibility in computer graphics.** Environment and Planning B: Planning and Design, 30(5):729–755, 2003.

[151] ZHUANG, Y.; GOLDBERG, K. ; PICKETT, M.. **Simplifying complex cad geometry with conservative bounding contours.** In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, volumen 3, p. 2503–2508. IEEE, 1997.

[152] MARTINEAU1, D. G.; GOULD1, J. D. ; PAPPER, J.. **An integrated framework for wrapping and mesh generation of complex geometries.** In: PROCEEDINGS OF THE VII EUROPEAN CONGRESS ON COMPUTATIONAL METHODS IN APPLIED SCIENCES AND ENGINEERING, 2016.

[153] EDELSBRUNNER, H.; MÜCKE, E. P.. **Three-dimensional alpha shapes.** ACM Transactions on Graphics (TOG), 13(1):43–72, 1994.

[154] CASELLES, V.; KIMMEL, R. ; SAPIRO, G.. **Geodesic active contours.** International journal of computer vision, 22(1):61–79, 1997.

[155] CASELLES, V.; KIMMEL, R.; SAPIRO, G. ; SBERT, C.. **Minimal surfaces: A geometric three dimensional segmentation approach**. Numerische Mathematik, 77(4):423–451, 1997.

[156] LORENSEN, W. E.; CLINE, H. E.. **Marching cubes: A high resolution 3d surface construction algorithm**. SIGGRAPH Comput. Graph., 21(4):163–169, Aug. 1987.

[157] ATTENE, M.; CAMPEN, M. ; KOBBELT, L.. **Polygon mesh repairing: An application perspective**. ACM Computing Surveys (CSUR), 45(2):15, 2013.

[158] AU, O. K.-C.; TAI, C.-L.; CHU, H.-K.; COHEN-OR, D. ; LEE, T.-Y.. **Skeleton extraction by mesh contraction**. ACM Trans. Graph., 27(3):44:1–44:10, Aug. 2008.

[159] LI, X.; IYENGAR, S.. **On computing mapping of 3d objects: A survey**. ACM Computing Surveys (CSUR), 47(2):34, 2015.

[160] JOHNSON, H. J.; MCCORMICK, M. M. ; IBANEZ, L.. **The ITK Software Guide Book 1: Introduction and Development Guidelines-Volume 1**, 2015.

[161] INRIA. **Geogram: a programming library of geometric algorithms**. `http://alice.loria.fr/software/geogram/doc/html/index.html`. Accessed: 2019-10-11.

[162] BOTSCH, M.; STEINBERG, S.; BISCHOFF, S. ; KOBBELT, L.. **Openmesh-a generic and efficient polygon mesh data structure**, 2002.

[163] CIGNONI, P.; CALLIERI, M.; CORSINI, M.; DELLEPIANE, M.; GANOVELLI, F. ; RANZUGLIA, G.. **Meshlab: an open-source mesh processing tool.** In: EUROGRAPHICS ITALIAN CHAPTER CONFERENCE, volumen 2008, p. 129–136, 2008.

[164] MAURER, C. R.; QI, R. ; RAGHAVAN, V.. **A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(2):265–270, 2003.

[165] LUEBKE, D. P.. **A developer's survey of polygonal simplification algorithms**. IEEE Computer Graphics and Applications, 21(3):24–35, 2001.

[166] WANG, Z.; BOVIK, A. C.; SHEIKH, H. R.; SIMONCELLI, E. P. ; OTHERS. **Image quality assessment: from error visibility to structural similarity**. IEEE transactions on image processing, 13(4):600–612, 2004.

[167] STUTZ, D.; GEIGER, A.. **Learning 3d shape completion under weak supervision**. International Journal of Computer Vision, p. 1–20, 2018.