Pontifícia Universidade Católica
do Rio de Janeiro

Jan Jose Hurtado Jauregui

Detail-preserving mesh denoising using adaptive patches

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marcelo Gattass

Rio de Janeiro
March 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Jan Jose Hurtado Jauregui

Detail-preserving mesh denoising using adaptive patches

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the undersigned Examination Committee.

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

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

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

Prof. Marcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, March 8th, 2018

Jan Jose Hurtado Jauregui

Graduated in computer science by the Universidad Nacional de San Agustin (Arequipa, Perú).

Bibliographic data

Hurtado Jauregui, Jan Jose

Detail-preserving mesh denoising using adaptive patches / Jan Jose Hurtado Jauregui; advisor: Marcelo Gattass. – Rio de janeiro: PUC-Rio , Departamento de Informática, 2018.

v., 84 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Procesamento Geométrico;. 3. Remoção de ruído de malha;. 4. Vizinhança adaptativa.. I. Gattass, Marcelo. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to first thank my parents Rocio Jauregui and Rusbel Hurtado, and my brother Cristian Hurtado for their continuous support. To my advisor Marcelo Gattass for his patience, motivation, and guidance. To my jury conformed by professors that I admire. To my graduation professor Cristian Lopez del Alamo because he guided and motivated me to start in my research area. To the professor Ernesto Cuadros because he explained me what is computer science. To the professor Alex Bronstein for his inmense knowledge and motivation, in fact, most of my work was inspired in his ideas. Finally, to the CAPES, PUC-Rio and Tecgraf Institute to support this work.

Abstract

Hurtado Jauregui, Jan Jose; Gattass, Marcelo (Advisor). **Detail-preserving mesh denoising using adaptive patches**. Rio de Janeiro, 2018. 84p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The acquisition of triangular meshes typically introduces undesired noise. Mesh denoising is a geometry processing task to remove this kind of distortion. To preserve the geometric fidelity of the desired mesh, a mesh denoising algorithm must preserve true details while removing artificial high-frequencies from the surface. Several algorithms were proposed to address this problem using a bilateral filtering scheme. In this work, we propose a two-step algorithm which uses adaptive patches and bilateral filtering to denoise the normal field, and then update vertex positions fitting the faces to the denoised normals. The computation of the adaptive patches is our main contribution. We formulate this computation as local quadratic optimization problems that we can control to obtain a desired behavior of the patch. We compared our proposal with several algorithms proposed in the literature using synthetic and real data.

Keywords

Geometry Processing; Mesh Denoising; Adaptive Patches.

Resumo

Hurtado Jauregui, Jan Jose; Gattass, Marcelo. **Remoção de ruído de malha com preservação de detalhe usando vizinhanças adaptativas**. Rio de Janeiro, 2018. 84p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A aquisição de malhas triangulares normalmente introduz ruídos indesejados. A remoção de ruído de malhas é uma tarefa da área de processamento geométrico que serve para remover esse tipo de distorção. Para preservar a fidelidade em relação à malha desejada, um algoritmo de remoção de ruído de malha deve preservar detalhes enquanto remove altas frequências indesejadas sobre a superfície. Vários algoritmos foram propostos para resolver este problema usando um esquema de filtragem bilateral. Neste trabalho, propomos um algoritmo de dois passos que usa vizinhança adaptativa e filtragem bilateral para remover ruído do campo normal e, em seguida, atualizar as posições dos vértices ajustando os triângulos às novas normais. A nossa contribuição principal é a computação da vizinhança adaptativa. Essa computação é formulada como problemas locais de otimização quadrática que podem ser controlados para obter o comportamento desejado da vizinhança. A proposta é comparada visual e quantitativamente com vários algoritmos propostos na literatura, usando dados sintéticos e reais.

Palavras-chave

Procesamento Geométrico; Remoção de ruído de malha; Vizinhança adaptativa.

Table of contents

List of figures

List of tables

1
Introduction

Nowadays 3D surface models are used in several fields and industries such as medicine, engineering, entertainment, geo-exploration, architecture, cultural heritage and so on. These models can be acquired from a variety of sources like 3D scanning, 3D imaging, multi-view stereo reconstruction, CAD modeling, etc. The data generated by these techniques should be processed to be available for production or any task where it can be used (visualization, simulation, animation, interaction, etc.). This processing step is called digital geometry processing which is a field of computer science that uses mathematical models and algorithms [1].

3D surface models obtained from real-world data usually present undesired noise, that can result in problematic effects on later applications. For example, depth-sensing cameras reconstruct noisy surfaces due to the physical limitations of the sensors. In Figure 1.1 we show some noisy models generated by this type of acquisition. As another example, in the case of medical data, a volume representing the anatomy of the patient can be obtained using several techniques (e.g., X-ray radiography, medical ultrasound, Magnetic Resonance Imaging (MRI), etc.). Using a subset of this volume, a 3D model can be reconstructed to represent the isosurface of the target region. The selection of a subset is carried out by a segmentation process which can be manual, automatic or semi-automatic. Due to these intermediate steps, the final model presents different kind of noise. Bade *et al.* explain some common noise sources in [3]: (1) Technical and physical circumstances when acquiring the image (e.g., speckle noise in medical ultrasound). (2) Image discretization. (3) Image segmentation. (4) Surface reconstruction. In Figure 1.2 we show some models generated from medical data.

This kind of problems is treated using denoising techniques, which are related to noise removal while preserving high-frequency features (details). Due that it is difficult to distinguish these features from noise, denoising is challenging problem so far. The denoising step is important in a typical geometry processing pipeline [1].

In a discrete setting, 3D surface models are commonly represented as triangle meshes due to its simplicity and easy processing. The denoising task

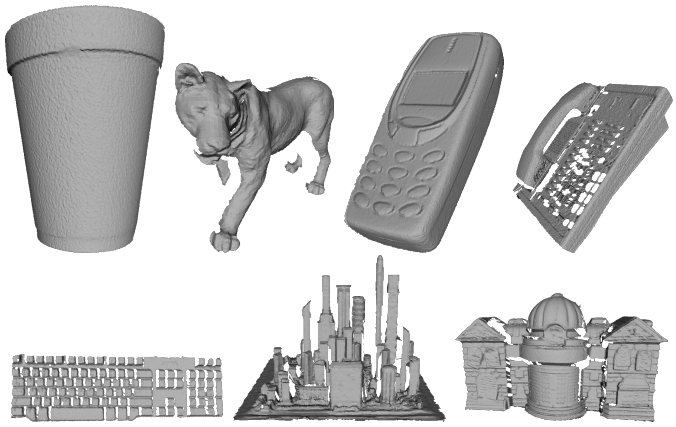


Figure 1.1: Range scans. Data obtained from [49].

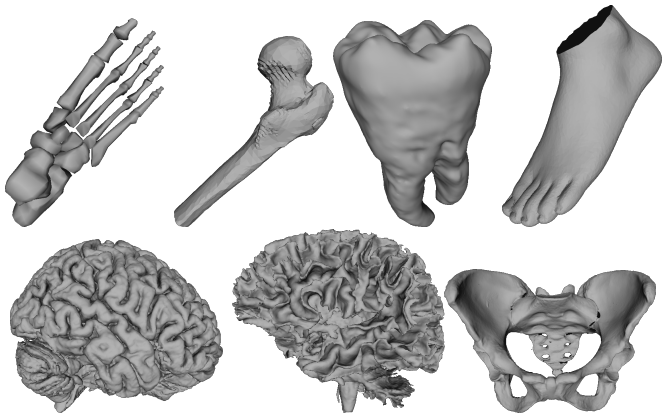


Figure 1.2: Meshes generated from medical data. Data obtained from the AIM@SHAPE Shape Repository [2]

over meshes is called *mesh denoising*, and it is related to the modification of the geometric properties of the mesh (e.g., vertex positions). There are some important considerations in mesh denoising algorithms such as detail-preserving, low normal variation, volume preservation, etc. Depending on the application these features determine robustness.

In this work, we propose a new algorithm for detail-preserving mesh denoising following a two-step scheme. First filtering the normal field and then updating vertex positions to adapt them to the filtered normals. Our main contribution is focused on the normal field filtering, which is performed using adaptive patches computed on the mesh. We formulate local quadratic optimization problems to compute these patches, such that the behavior of the patch is influenced by multiple features. The final result of our algorithm strongly depends on the computation of the adaptive patches.

We performed several experiments to describe our proposal and to compare it with other denoising algorithms, using synthetic and real data. The results that we obtained show that our proposal successfully removes the noise

while preserving-details, and in most test cases was better than the others. We evaluated it visually and used several mesh denoising metrics.

This document is structured as follows. In Chapter 2 we present some previous work relevant to our problem. In Chapter 3 we describe some important definitions to understand a step of our proposal and a comparison to a patch-based algorithm. In Chapter 4 we explain how we compute the adaptive patches, and how we discretize and implement them. In Chapter 5 we explain how our denoising algorithm works. To compare our proposal with others, in Chapter 6 we show some strategies to evaluate denoising algorithms. In Chapter 7 we describe our experiments and results. Finally, in Chapter 8 we present our conclusion and future work.

2
Previous Work

Early smoothing methods tried to minimize functionals such as total curvature or membrane energy [4, 5, 6] introducing computationally expensive algorithms. Laplacian smoothing is one of the most used methods due to its fastest and simplest scheme [7, 8, 9, 10], it moves each vertex iteratively to a relative position regarding its respective neighboring vertices. Unfortunately, Laplacian smoothing produces mesh shrinkage which is an important problem when working with medical data. To avoid this problem, Taubin proposed a two-step Laplacian smoothing, expanding the mesh on each iteration [11, 12]. The behavior of the algorithm depends on the discretization of the Laplace operator. Desbrun *et al.* introduced diffusion flow for irregular mesh smoothing, using cotangent weights in the Laplace operator [13]. Also, they introduced an implicit version of the smoothing process. Vollmer *et al.* extended the Laplacian smoothing in the same way as Taubin but using different expanding directions [14].

All the methods above are called isotropic filters, which means that they are independent of surface geometry. As a counterpart there exists another kind of filters called anisotropic which usually better preserve geometric features while removing noise [15]. Based on a diffusion process, numerous anisotropic filters were proposed [16, 17, 18, 19, 20] 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 [21]. He and Schaefer proposed a method for sharp features preservation [22] using L_0 minimization.

The bilateral filter for images was an important inspiration for a lot of anisotropic mesh filters. The adaptation of this filter was introduced by Fleishman et al. [23] and Jones et al. [24], and then generalized by Solomon et al. [25]. Two step methods, consisting in normal field filtering followed by vertex updating, were proposed adopting an anisotropic behavior [26, 27, 28]. Using a bilateral filter for normal field filtering, Zheng et al. proposed an iterative and global scheme for mesh denoising [29]. Wei et al. introduced a bilateral normal filtering using face normals and vertex normals to reach more robustness [30]. Using a guidance signal generated by computing an average normal from consistent patches, Zhang et al. proposed an extension of the

joint bilateral filter [31]. Later, Li *et al.* tried to improve the consistent patch definition proposing a new metric [32].

Recently, using binary optimizations, Yadav *et al.* proposed a normal voting tensor to denoise the normal field and then update vertices [33]. Then, in [34], the same authors proposed an edge-weighted Laplace operator to avoid face normal flip and to be more robust to high-intensity noise. 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 in [35].

Our adaptive patch computation follows the idea of computing consistent patches as shown in [31, 32, 35], performing a new optimization procedure proposed here. Our denoising algorithm uses these patches to filter the normal field in an iterative manner, including an optional step that performs a bilateral filtering [29] over the new normals (to obtain smoother results).

3
Anisotropic denoising

3.1
Bilateral filtering for images

The bilateral filtering is an image denoising method which preserves edges. We can represent an image as a signal $g(p)$, where p is the minimal structure, i.e. a pixel. Following this notation the bilateral filtering is defined as follows:

$$g'(p) = \frac{1}{W(p)} \sum_{q \in N(p)} K_c(||p-q||) K_s(||g(q)-g(p)||) g(q) \tag{3-1}$$

where $g'(p)$ is the new signal value at pixel p, $W(p)$ is the normalization factor, $N(p)$ represents the neighboring pixels of p, $K_c(x)$ is the kernel function which penalize spatial distance between pixels, and $K_s(x)$ is the kernel function which penalize distance between signal values. The normalization term is defined as the sum of the weights of all entries included in a sum, such that for a sum $\sum \omega x$, we can represent it as $W(x) = \sum \omega$. Kernel functions mentioned in this work are defined as Gaussian functions with the following form: $K_i(x) = e^{-x^2/2\sigma_i^2}$.

3.2
Joint bilateral filtering for images

The joint bilateral filtering is an extension of bilateral filtering which uses a guidance image signal $\tilde{g}(p)$ to denoise the original signal $g(p)$. The guidance signal should be defined in the same domain and determines the behavior of the filter. A joint bilateral filter can be defined as follows:

$$g'(p) = \frac{1}{W(p)} \sum_{q \in N(p)} K_c(||p-q||) K_s(||\tilde{g}(q)-\tilde{g}(p)||) g(q) \tag{3-2}$$

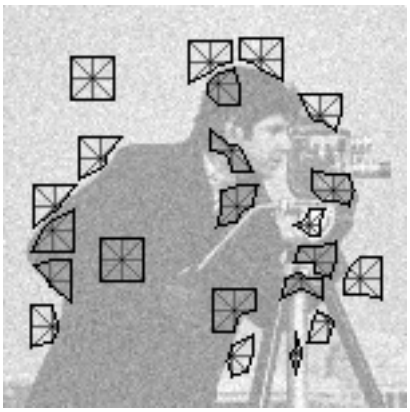


Figure 3.1: Adaptive patches (image adapted from [36]).

3.3 Adaptive patches for images

In the image processing field, image denoising was addressed using shape adaptive patches or anisotropic neighborhoods. This idea is focused on the preservation of features such as edges or corners (detail preservation). For example, in [36] the authors proposed an image filtering algorithm using patches generated by consistent 1D signals. These signals are described by the intensity values of a segment defined on the image. The segments they use for a sampled pixel are related two the 8 pixel directions: left, right, up, down and diagonals. The length is selected considering the consistency of the corresponding segment. Finally, the polygon described by the endpoints of these segments defines the patch for a given pixel. In Figure 3.1 we show an example of these adaptive patches.

3.4 Bilateral filtering for meshes

Based on the bilateral filtering for images, Fleishman et al. [23] proposed an extension for mesh denoising. Their idea is based on computing an approximated surface for each vertex (i.e. a plane) and use the distance of the nearest vertices to this surface as a signal distance. In triangular meshes the normal of the tangent plane of a vertex can be obtained averaging the normals of the faces containing it. So, the plane can be represented by the normal $\mathbf{n}_i$ and the corresponding point $\mathbf{x}_i$ at vertex v_i. The distance between a point $\mathbf{x}_j$ and this plane can be computed by $\langle \mathbf{n}_i, \mathbf{x}_i - \mathbf{x}_j \rangle$. Using euclidean distance between vertices and the mentioned signal distance, the new vertex position is computed

as follows:

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{n}_i \frac{1}{W(v_i)} \sum_{v_j \in N_v(v_i)} K_c(||\mathbf{x}_i - \mathbf{x}_j||) K_s(h_{i,j}) h_{ij} \tag{3-3}$$

where $\mathbf{x}'_i$ is the new vertex position, $N_v(v_i)$ the set of neighboring vertices of vertex v_i and $h_{ij} = \langle \mathbf{n}_i, \mathbf{x}_i - \mathbf{x}_j \rangle$. The normal determines the direction of the movement and the bilateral filtering the step size.

3.5 Bilateral normal filtering for meshes

Instead of filtering vertex positions other authors proposed normal field filtering, considering normals over the gauss sphere. Zheng et al. [29] introduced a new way to filter normals considering their positions in the global space. Their scheme defines a bilateral filter over face normals, using the normals as signal and the corresponding face centroids as positions. After normal filtering they use a vertex updating step explained before. The new normals are computed in the following way:

$$\mathbf{n}'_i = \frac{1}{W(f_i)} \sum_{f_j \in N_f(f_i)} A_j K_c(||\mathbf{c}_i - \mathbf{c}_j||) K_s(||\mathbf{n}_j - \mathbf{n}_i||) \mathbf{n}_j \tag{3-4}$$

where $\mathbf{n}'_i$ is the new normal, $N_f(f_i)$ represents the set of neighboring faces of face f_i, $\mathbf{c}_k$ is the corresponding centroid of f_k, and A_k is the area of the face f_k. This area works as a weight for each triangle, larger areas should be more influential. After normal filtering a vertex updating step is performed to adapt the vertices to the new normals.

3.6 Guided bilateral normal filtering for meshes

This algorithm is based on the joint bilateral filter, which uses a guidance signal (external signal) in a bilateral filtering scheme. The effectiveness of filtering depends on how similar is the guidance signal to the desired signal. Zhang et al. proposed normal field filtering [31] in the same way as in [29], but adopting a joint bilateral filtering scheme:

$$\mathbf{n}'_i = \frac{1}{W(f_i)} \sum_{f_j \in N_f(f_i)} A_j K_c(||\mathbf{c}_i - \mathbf{c}_j||) K_s(||\tilde{\mathbf{n}}_j - \tilde{\mathbf{n}}_i||) \mathbf{n}_j \tag{3-5}$$

where $\tilde{\mathbf{n}}_k$ is the corresponding guidance signal value of face f_k. The guidance signal is defined as the average normal of a consistent patch centered at the corresponding face.

To obtain a consistent patch, the authors define a set of regular patches as candidates, and then select the most consistent. A regular patch P defined for a face f is conformed by the faces which share any of the vertices of f, including f. For a given face f, the set of candidate patches is conformed by all regular patches which share f. The consistency of a patch is defined by the following error metric:

$$\mathcal{H}(\mathcal{P}) = \Phi(\mathcal{P})\mathcal{R}(\mathcal{P}), \tag{3-6}$$

where $\Phi(\mathcal{P})$ measures the maximum difference between two face normals of the patch:

$$\Phi(\mathcal{P}) = \max_{f_i, f_k \in \mathcal{P}} ||\mathbf{n}_j - \mathbf{n}_k||, \tag{3-7}$$

and $\mathcal{R}(\mathcal{P})$ measures the saliency of the patch in the following manner:

$$\mathcal{R}(\mathcal{P}) = \frac{\max_{e_j \in \mathcal{P}} \phi(e_j)}{\varepsilon + \sum_{e_j \in \mathcal{P}} \phi(e_j)}, \tag{3-8}$$

where $\phi(e_j)$ is the length of the normal difference of two faces conforming the edge e_j and ε is a small positive value to avoid division by 0. The patch with lowest error is considered the most consistent.

As in bilateral normal filtering, a vertex updating step is needed to obtain new vertex positions.

4
Adaptive Patches

In this chapter, we will describe the core of our mesh denoising algorithm, which is the computation of adaptive patches. As in the image denoising case where adaptive patches are computed to preserve edges (e.g., [36]), we opted to compute adaptive patches over meshes and use them to perform a denoising operation preserving details. These patches have to be robust to noise and configurable to reach a desired behavior for the denoising algorithm. These properties define the effectiveness of our denoising algorithm. We will explain this computation in a continuous setting and then introduce its discretization and implementation.

4.1
Continuous setting

Let X be a 2-manifold embedded in $\mathbb{R}^3$, and X' a subset of X representing a patch (neighborhood) for a reference point $x' \in X$. We aim to find an optimal subset X' considering some properties desired in our denoising algorithm. The first one and the most important is that the patch should be adapted to the desired shape. Adaptive patches will be helpful to preserve sharp features while denoising flat regions. As we shown in a previous section this kind of patches are computed over images using the pixel intensity as the main signal. In our case, we opted to use the normal field as shape descriptor, such that flat regions will have low normal variation, and sharp features or curved regions will have higher normal variation. The patch we want to compute should be piecewise constant regarding the normal field or with minimal normal variation. In other words, the normal difference between two points contained in the patch should be minimum.

4.1.1
Error term

Finding the solution X' allows us to formulate a quadratic optimization problem penalizing the error between two points $x_i \in X'$ and $x_j \in X'$ as

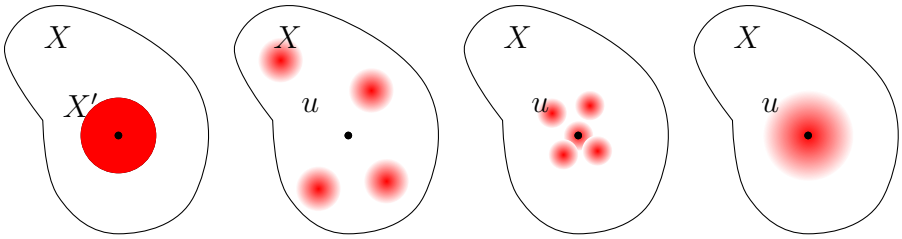


Figure 4.1: Reference point in black. Left: crisp subset X' representing the patch. Center-Left: a non regular solution u for optimization problem without using the distance term. Center-Right: a non regular solution u for optimization problem using the distance term. Right: a regular solution u for optimization problem

$q_{ij} = ||n_i - n_j||$. We can write this problem as follows:

$$\min_{X' \subseteq X} \int_{x_i \in X'} \int_{x_j \in X'} q_{ij} da. \tag{4-1}$$

Unfortunatelly, in a discrete setting, finding a crisp subset X' results in a NP-Hard combinatorial complexity problem. As suggested in other applications we can relax the problem defining a fuzzy membership function $u : X \to [0, 1]$ over all the domain X (fuzzy set theory). This function will define which is the degree of inclusion of a point $x_i \in X$ to the patch X'. Instead of finding the subset X', now we want to find the function u (see Figure 4.1). We can rewrite the problem:

$$\min_{u} \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada \quad s.t. \quad u \in [0, 1], \tag{4-2}$$

where $u_i = u(x_i)$ and $u_j = u(x_j)$. Now the optimization will find the membership function u and control the upperbound and lowerbound constraints.

If we try to solve the latter problem it is obvious that the solution can be $u = 0$ which leads in a patch without area. To obtain a solution with area we added a constraint for the solution u, such that the sum of the area of X weighted by the membership function u should be equal to a fixed value a_0. Rewriting the problem we have:

$$\min_{u} \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada \quad s.t. \quad u \in [0, 1] \quad \wedge \quad \int_{x_i \in X} uda = a_0. \tag{4-3}$$

4.1.2 Distance to the reference point term

We are computing the solution over all the domain, for this reason the patch can be defined over a far region regarding the reference point. The normal

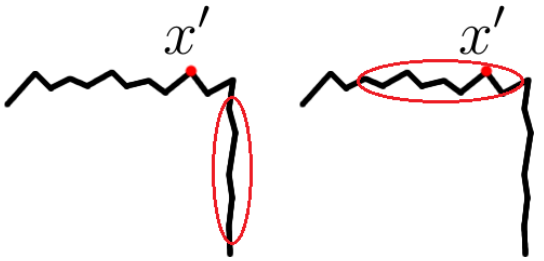


Figure 4.2: Patch coherence. Left: possible patch with reference point x' lying on the wrong side of the shape. Right: possible patch with reference point x' lying on the right side of the shape.

field lies on the unit sphere and we do not have any term to control the spatial information. The patch must be "centered" on the reference point or close to it. To address this problem we added a term to penalize the distance between any point of X to the reference point x' considering the weights of function u:

$$\begin{aligned} \min_u \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada + \int_{x_i \in X} ||x' - x_i|| u_i da \\ s.t. \quad u \in [0,1] \quad \wedge \quad \int_{x_i \in X} uda = a_0, \end{aligned} \tag{4-4}$$

where x' is the reference point (see Figure 4.1). Here we are using Euclidean distances but a more exact result can be obtained using geodesic distances.

4.1.3 Regularization term

One problem with the possible solution in the latter formulation is that it can be too irregular (see Figure 4.1). To avoid this problem, we follow the idea of the Mumford-Shah functional trying to minimize the boundary length of the solution. Because we have a continuous function u instead of a crisp set X', we opted to use the squared gradient norm of u as a penalization term.

$$\begin{aligned} \min_u \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada + \int_{x_i \in X} ||x' - x_i|| u_i da \\ + \int_X ||\nabla u||^2 da \quad s.t. \quad u \in [0,1] \quad \wedge \quad \int_{x_i \in X} uda = a_0, \end{aligned} \tag{4-5}$$

4.1.4 Normal coherence term

Depending on the influence of each of the previous terms we can obtain a solution including the reference point or not. If not it is possible to obtain a

solution which has no coherence with the desired reference point normal (see Figure 4.1). For example, if we have a noisy cube and we are computing a patch with a reference point close to an edge, the solution can lie in the wrong face because probably this region is flatter.

$$\begin{aligned} \min_u \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada + \int_{x_i \in X} ||x' - x_i|| u_i da \\ + \int_X ||\nabla u||^2 da + \int_{x_i \in X} ||n' - n_i|| u_i da \\ s.t. \quad u \in [0,1] \quad \wedge \quad \int_{x_i \in X} uda = a_0, \end{aligned} \tag{4-6}$$

where n' is the normal of the reference point, and n_i is the normal of the point x_i. Solving this optimization problem will give us the adaptive patch described by a membership function u.

4.1.5 Parameters

To control the behavior of the optimization problem we introduced four parameters unfluencing each term of the optimization:

$$\begin{aligned} \min_u \alpha \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada + \beta \int_{x_i \in X} ||x' - x_i|| u_i da \\ + \gamma \int_X ||\nabla u||^2 da + \delta \int_{x_i \in X} ||n' - n_i|| u_i da \\ s.t. \quad u \in [0,1] \quad \wedge \quad \int_{x_i \in X} uda = a_0, \end{aligned} \tag{4-7}$$

where α controls the normal error difference, β controls the distance to the reference point, γ controls the regularity of u and δ the normal coherence regarding the reference point.

4.2 Discretization

Due that our denoising algorithm works over triangular meshes, we have two candidates to use as sampled points to represent the manifold: vertex positions and face centroids. We opted to use face centroids because our algorithm uses the normal field generated by face normals. A triangular mesh M can be represented as a set of m vertices $V = \{v_1, \ldots, v_m\}$ and a set of n faces $F = \{f_1, \ldots, f_n\}$. Each face (triangle) is described by the three indices of the vertices that conform it. The position of the mesh vertices can be represented as $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ where $\mathbf{x}_i = \mathbf{x}(v_i) = (x(v_i), y(v_i), z(v_i))^T$.

Face centroids can be represented as $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ where $\mathbf{c}_i = \mathbf{c}(f_i) = (x(f_i), y(f_i), z(f_i))^T$. In a similar way, we can represent face normals as $N = \{\mathbf{n}_1, \ldots, \mathbf{n}_n\}$ where $\mathbf{n}_i = \mathbf{n}(f_i) = (nx(f_i), ny(f_i), nz(f_i))^T$. Given a face f conformed by the vertices v_1, v_2 and v_3, its corresponding normal can be obtained by $\mathbf{n} = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)$. The direction of the normal will depend on the order of the vertices (clockwise or counterclockwise).

We will define a patch over the face domain which is represented by the face centroids. If we define the patch as a crisp subset, we can represent it as a subset of faces $F' \subseteq F$. In our formulation we define a membership function over this domain (F), so we can represent it as a vector $\mathbf{u} = \{u_1, \ldots, u_n\}^T$ where u_i represents the membership value of face f_i. Due that our sampled points are all face centroids and we want to integrate it over all the domain we need the area that they represent. We assume that the area for each centroid is the area of the corresponding face. We can represent all areas by a vector $\mathbf{a} = \{a_1, \ldots, a_n\}^T$ containing all face areas as entries. Also we can represent them using a diagonal matrix $\mathbf{A}$ such that $(a_{ii}) = a_i$.

The first term of the optimization problem has a quadratic form and in a discrete setting can be rewritten as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} u_i a_i u_j a_j, \tag{4-8}$$

using a matrix form we have:

$$\mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u}, \tag{4-9}$$

where $\mathbf{A}$ is the diagonal matrix containing face areas and $\mathbf{Q}$ is an error matrix containing all normal differences between two faces (or centroids). Each entry of $\mathbf{Q}$ is defined by $q_{ij} = ||\mathbf{n}_i - \mathbf{n}_j||$.

The lowerbound and upperbound constraints can be represented by the vectors $\mathbf{0}$ and $\mathbf{1}$, which are n-dimensional vectors containing in all their entries 0s and 1s respectively. The area constraint results in a single linear constraint $\sum_i^n a_i u_i = a_0$, whose matrix representation is: $\mathbf{a}^T \mathbf{u} = a_0$. Now, considering the error term and the constraints we have:

$$\min_{\mathbf{u}} \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} \quad s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0. \tag{4-10}$$

In practice, we restrict the domain of the optimization problem to a regular neighborhood limited by a given radius. In Figure 4.3 we show an example of how a membership function looks like. It is important to say that the example we show in this figure uses all terms defined in the optimization

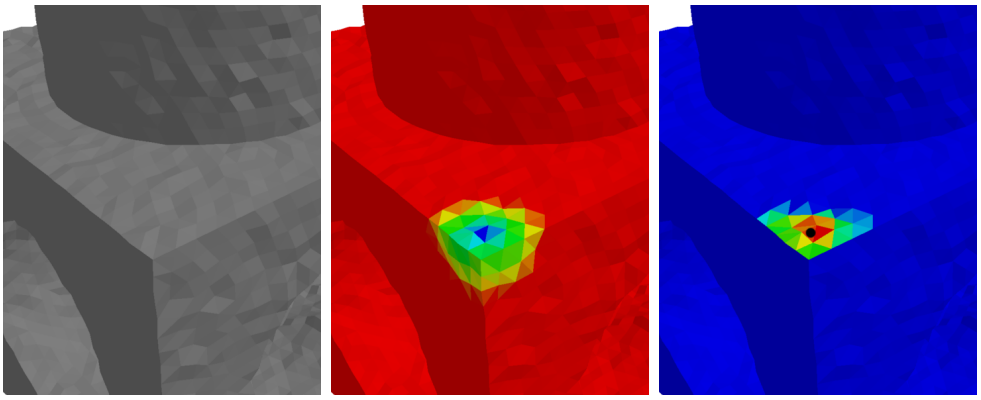


Figure 4.3: Left: noisy mesh. Center: color mapping of the distance function, from minimum distance (blue) to maximum distance (red). Right: color mapping of membership function u, from 0 (blue) to 1 (red). The domain of the optimization problem is restricted to the non fully red area in the color mapping of distance function.

problem and it is restricted to the domain generated by a fixed radius.

According to our formulation, we need a term that penalizes the distance between any point inside the patch to the reference point. As we mentioned before, we are working on face centroids, so the distance between two points is defined by the euclidean distance between the corresponding face centroids. We can consider a $n \times n$ matrix $\mathbf{D}$ containing all distances between two pair of points: $d_{ij} = ||\mathbf{c}_i - \mathbf{c}_j||$. The distance term is linear, so if the reference point index is k we can use the kth column of $\mathbf{D}$ in the optimization problem. For convenience we will call this vector as $\mathbf{d}$. Also, we have to integrate this distance penalization over the involved area. Let us denote the area of the reference point as a' (area of the reference face). This term can be described by $a' \sum_i^n d_i a_i u_i$, and in a matrix form by $\mathbf{d}^T a' \mathbf{A}\mathbf{u}$. Rewriting the optimization problem we have:

$$\min_{\mathbf{u}} \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} + \mathbf{d}^T a' \mathbf{A}\mathbf{u} \quad s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0. \tag{4-11}$$

In Figure 4.4 we show an example of a solution using the distance to the reference point and without using it.

Now, to regularize the solution we have to introduce the squared gradient norm term. So we have to define a gradient norm operator and its respective matrix form. To simplify this definition, we approximate it with the following formulation. We assume that u is constant over all the face (triangle), so the gradient norm is 0 within it. For this reason, we only need to integrate the gradient norm over the edges. Adopting the mentioned scheme, for each point over an edge that shares faces f_i and f_j, the gradient should be orthogonal

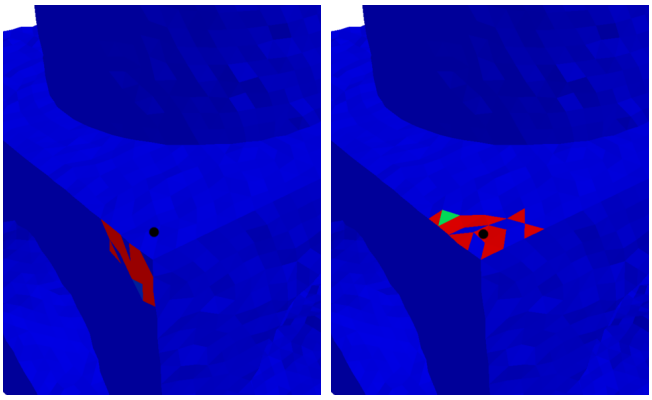


Figure 4.4: Membership function behavior when including penalization for distance to the reference point. Reference point in black. Left: membership function u obtained without using the distance to reference point penalization term. Right: membership function u obtained using the distance to reference point penalization term. Both results do not use a regularization (gradient) or normal coherence term.

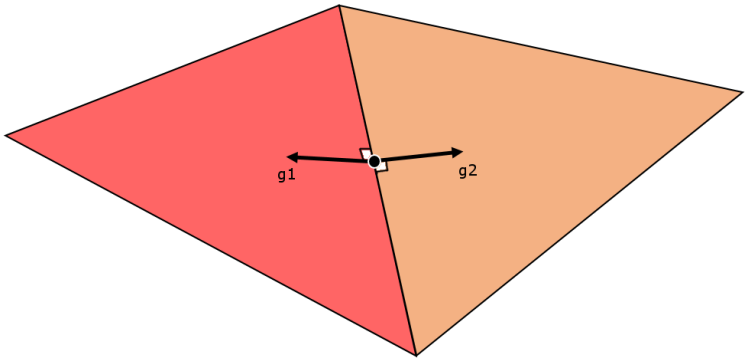


Figure 4.5: Gradient possible directions: g1 and g2.

to the edge and has only two possible directions depending on u values (see Figure 4.5). Due that we have only two possible gradient directions we can think about it as a 1D gradient, such that the norm of the gradient over an edge point is equal to $|u_i - u_j|$. Following this idea, integrating the gradient norm over all points of the edge results in the following expression:

$$l|u_i - u_j|, \tag{4-12}$$

where l is the edge length.

With this formulation we can define the gradient norm operator as the following matrix:

$$\mathbf{G} = (g_{ij}) = \begin{cases} \sum_{f_k \in N_f(f_i)} l_{ik} & i = j \\ -l_{ij} & e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{4-13}$$

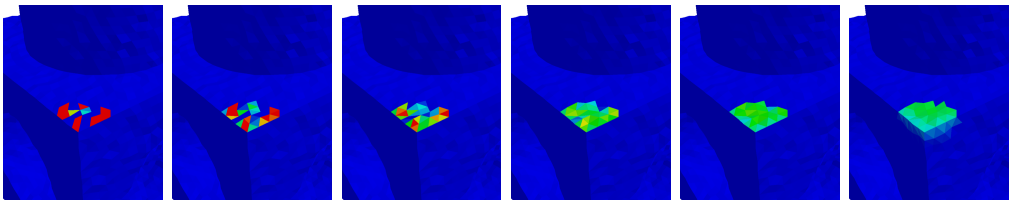


Figure 4.6: Membership function behavior when increasing the influence of regularization term in the optimization problem. From left to right: less influence to high influence ($\gamma = 0.05, 0.1, 0.2, 0.5, 10, 100$).

where $N_f(f_i)$ is a set containing edge based neighboring faces of f_i, E is the set of all edges of the mesh, e_{ij} is an edge sharing faces f_i and f_j, and l_{ij} is the lenght of the edge e_{ij}. So, the squared gradient norm of $\mathbf{u}$ can be calculated as follows:

$$||\nabla u||^2 \approx (\mathbf{Gu})^2 = \mathbf{u}^T\mathbf{G}^T\mathbf{Gu}. \tag{4-14}$$

Rewritting the optimization problem we have:

$$\min_{\mathbf{u}} \mathbf{u}^T\mathbf{A}^T\mathbf{QAu} + \mathbf{d}^T a'\mathbf{Au} + \mathbf{u}^T\mathbf{G}^T\mathbf{Gu} \quad s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T\mathbf{u} = a_0 \tag{4-15}$$

When giving more influence to this term, we will obtain more regular solutions, i.e., with a lower variation of u. In Figure 4.6 we can see how the solution behaves when increasing this influence. It is important to balance the influence of this term to obtain regular solutions and be careful about too smooth solutions which are not helpful for our denoising algorithm (solutions not adapted to the shape).

The coherence term which penalizes the difference between the normal of a point and the normal of the reference point is linear. So we can discretize it in the same manner as in the distance to the reference point discretization. Let us denote the area of the reference point as a' (area of the reference face) and the normal of the reference point as $\mathbf{n}'$. We can write this term as $a' \sum_i^n ||\mathbf{n}_i - \mathbf{n}'||a_i u_i$, and in a matrix form as $\mathbf{f}^T a'\mathbf{Au}$, where $\mathbf{f}$ is a n dimensional vector containing in ith position the normal difference between the reference face and the face f_i. Rewritting the optimization problem we have:

$$\min_{\mathbf{u}} \mathbf{u}^T\mathbf{A}^T\mathbf{QAu} + \mathbf{d}^T a'\mathbf{Au} + \mathbf{u}^T\mathbf{G}^T\mathbf{Gu} + \mathbf{f}^T a'\mathbf{Au} \quad s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T\mathbf{u} = a_0 \tag{4-16}$$

In Figure 4.7 we show an example of a solution using the coherence term and a solution without using it.

Considering the parameters that control the optimization behavior we

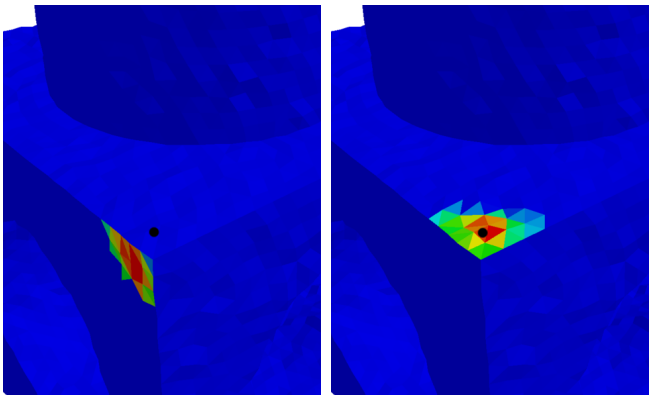


Figure 4.7: Coherence term. Membership function behavior when including penalization for difference between the normal of each point and the normal of the reference point. Reference point in black. Left: membership function u obtained without using the coherence term . Right: membership function u obtained using the coherence term.

have the following quadratic optimization problem for each face centroid of the mesh:

$$\begin{aligned} \min_{\mathbf{u}} \alpha \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} + \beta \mathbf{d}^T a' \mathbf{A} \mathbf{u} + \gamma \mathbf{u}^T \mathbf{G}^T \mathbf{G} \mathbf{u} + \delta \mathbf{f}^T a' \mathbf{A} \mathbf{u} \\ s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0. \end{aligned} \tag{4-17}$$

Factorizing, it leads to a typical quadratic optimization problem with a quadratic term, a linear term, upper bound constraints, lower bound constraints, and a single linear equality constraint. In Figure 4.8 we show some examples of solutions for the noisy block model.

4.3 Implementation

Due that our proposal was developed for triangular meshes, we use a half-edge based data structure to represent them. This data structure allows us to iterate over the mesh elements in linear time following the topology. We precompute some relevant information for the adaptive patch computation: face areas, face centroids, face normals and all distances between the centroids of two pair of faces. To the first three items, we can perform the computation shown in Algorithm 1, where *mesh* is the input mesh with F as corresponding faces, *areas* is an array containing face areas, *centroids* is an array containing face centroids, *normals* is an array containing face normals, and $N_v(f)$ is the set of vertices which define the face f.

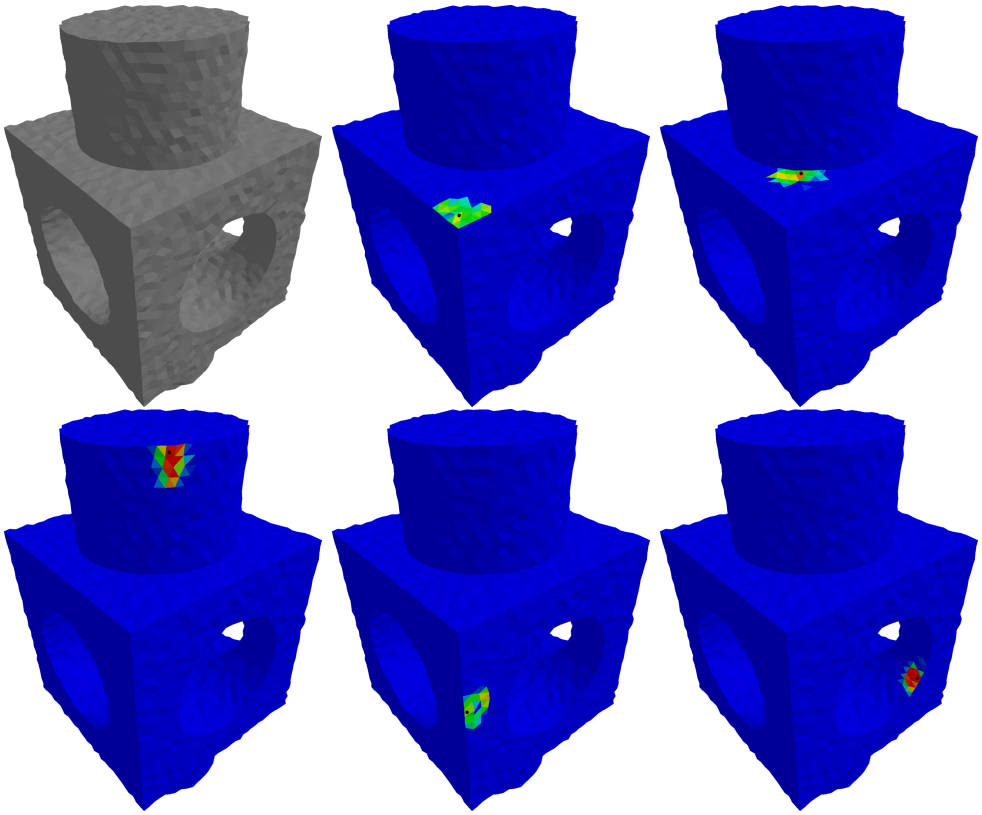


Figure 4.8: Some examples of solutions for adaptive patches. Reference point in black. First image: noisy block model.

Algorithm 1: Compute mesh face areas, centroids and normals

Input : *mesh*

Output: *areas*, *centroids*, *normals*

foreach $f \in F$ **do**
  $p_1 \leftarrow \text{empty}()$
  $p_2 \leftarrow \text{empty}()$
  $p_3 \leftarrow \text{empty}()$
  foreach $v \in N_v(f)$ **do**
    $p_i \leftarrow \mathbf{x}(v)$
  end
  $areas(f) \leftarrow length(0.5 * ((p_2 - p_1) \times (p_2 - p_3)))$
  $centroids(f) \leftarrow (p_1 + p_2 + p_3)/3$
  $normals(f) \leftarrow (p_2 - p_1) \times (p_3 - p_1)$
end

As we mentioned before we need the distances between two pair of faces and restrict the domain to a small radius based neighborhood of the reference point. We adopted a geometrical neighborhood, which follows the topology neighborhood expansion behavior, until a neighbor fall out of a defined

euclidean sphere (with a fixed radius and centered at the reference point). The first level of a topology neighborhood for a single face f is defined as the set of faces which share any of the vertices of f. Following a breadth-first traversal, we can expand the neighborhood considering the limits of the Euclidean sphere, and compute a sparse distance matrix containing the distances between two pair of faces mutually reached. If the faces were not mutually reached, the value in the sparse matrix is ∞ (such as 0 value in a typical sparse matrix). In Algorithm 2 we show how we compute this distance matrix which is helpful to define the distances and the geometrical neighborhoods. In the algorithm, *mesh* is the input mesh, $maximumDistance$ is the radius of the limiter Euclidean sphere, *centroids* is a vector containing face centroids, $distanceMatrix$ is the resulting sparse matrix of distances, and $N_f(f_i)$ is the vertex based face neighborhood (first level topology neighborhood) of face f_i.

Algorithm 2: Compute radius based face distances

Input : *mesh*, $maximumDistance$, *centroids*
Output: $distanceMatrix$

```
1  fill(distanceMatrix, Inf)
2  foreach f_i ∈ F do
3      c_i ← centroids(f_i)
4      queue ← empty()
5      queue.push(f_i, 0)
6      distanceMatrix(i, i) = 0
7      mark f_i as visited
8      while not queue.isEmpty() do
9          (f_c, d_c) ← queue.front()
10         distanceMatrix(i, c) = d_c
11         queue.pop()
12         foreach f_j ∈ N_f(f_i) do
13             c_j ← centroid(f_j)
14             if f_j not visited and
                length(c_i − c_j) <= maximumDistance then
15                 queue.push(f_j, length(c_i − c_j))
16             end
17             mark f_j as visited
18         end
19     end
20 end
```

Now, we have to build the optimization problem based on the subdomain

defined in the sparse distance matrix. In Algorithm 3 we show how to set up each term of the formulation and how to call the solver. *mesh* is the input mesh, *areas* is an array containing face areas, *centroids* is an array containing face centroids, *normals* is an array containing face normals, $distanceMatrix$ is the sparse distance matrix, f_k is the face of the reference point, α the parameter which controls the normal error difference, β the parameter which controls the distance to the reference point, γ the parameter which controls the regularity of $\mathbf{u}$, δ the parameter which controls the normal coherence regarding the reference point, and a_0 the area constraint. In the algorithm we use the following functions: $numOfNIV$ returns the number of non infinity values of a vector of the sparse distance matrix, $facesOfNIV$ returns the neighboring faces of a vector of the sparse distance matrix, $edgesOfNIV$ returns the edges of the neighborhood defined in a vector of the sparse distance matrix (avoiding boundary edges), $normalsOfNIV$ returns the normals of the neighborhood defined in a vector of the sparse distance matrix, $nonInfValues$ returns the distances of neighboring faces of a vector of the sparse distance matrix, $genSparseVector$ returns a sparse vector (with 0 values) based on the distribution of the correspoding $distanceMatrix$ vector, $zeros(m)$ initialize a m-dimensional vector containing 0s, $ones(m)$ initialize a m-dimensional vector containing 1s and $zeros(m, n)$ a $m \times n$ matrix containing 0s. To find the solution of the optimization problem we can use any solver which supports a quadratic term and the defined constraints.

We can control the number of variables for the optimization problem including the n nearest faces to the reference point instead of all the neighborhood. The latter is important to avoid optimization problems with a too large number of variables which can generate a bottleneck in our implementation.

Algorithm 3: Compute membership function u

Input : $mesh$, $areas$, $centroids$, $normals$, $distanceMatrix$, f_k, α, β, γ, δ, a_0

Output: u

$n \leftarrow numOfNIV(distanceMatrix(k))$
$F_k \leftarrow facesOfNIV(distanceMatrix(k))$
$E_k \leftarrow edgesOfNIV(distanceMatrix(k))$ `// only non-boundary edges of the subset`
$N_k \leftarrow normalsOfNIV(distanceMatrix(k))$
$\mathbf{d} \leftarrow nonInfValues(distanceMatrix(k))$
$\mathbf{f} \leftarrow zeros(n)$
$\mathbf{a} \leftarrow zeros(n)$
$\mathbf{Q} \leftarrow zeros(n, n)$
$\mathbf{G} \leftarrow zeros(n, n)$
foreach $f_i \in F_k$ **do**
    $\mathbf{a}(i) \leftarrow areas(i)$
    $\mathbf{f}(i) \leftarrow length(N_k(i) - normals(k))$
    $perimeter \leftarrow 0$
    foreach $f_j \in F_k$ **do**
        $\mathbf{Q}(i, j) \leftarrow length(N_k(i) - N_k(j))$
        if $e_{ij} \in E_k$ *and* $i \neq j$ **then**
            $\mathbf{G}(i, j) \leftarrow -length(e_{ij})$
            $\mathbf{G}(j, i) \leftarrow -length(e_{ij})$
            $perimeter \leftarrow perimeter + length(e_{ij})$
        end
    end
    $G(i, i) \leftarrow perimeter$
end
$\mathbf{A} \leftarrow diag(\mathbf{a})$
$\mathbf{H} \leftarrow \alpha\mathbf{A}^T\mathbf{Q}\mathbf{A} + \gamma\mathbf{G}^T\mathbf{G}$
$\mathbf{b}^T \leftarrow \beta\mathbf{d}^T areas(k)\mathbf{A} + \delta\mathbf{f}^T areas(k)\mathbf{A}$
$\mathbf{0} \leftarrow zeros(n)$
$\mathbf{1} \leftarrow ones(n)$
$\mathbf{u} = \arg\min_{\mathbf{u}} \mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{b}^T\mathbf{u} \;\; s.t. \;\; \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T\mathbf{u} = a_0$ `// solves the optimization problem`
$u \leftarrow genSparseVector(\mathbf{u}, distanceMatrix(k))$ `// returns a sparse vector with the same distribution followed in` $distanceMatrix(k)$

5
Denoising Algorithm

The proposed algorithm is based on a local iterative scheme as well as many of mesh denoising algorithms. We aim to compute new vertex positions such that the new mesh suffers less noise while preserving details. It is important to say that our algorithm follows two main stages: normal field filtering and vertex updating. The normal field filtering is divided into two steps: filtering based on adaptive patches and bilateral filtering. Here, we explain each step of the algorithm independently and how the entire algorithm works.

5.1
Precomputation

Due that our formulation can be very sensitive to scaling, we introduce a previous step to normalize vertex positions. First, we compute the average edge length just iterating over the edge set and accumulating the corresponding lengths to obtain the average value. In Algorithm 4 we show the pseudocode to compute it.

Algorithm 4: Average Edge Length

Input : E
Output: avg

```
1 foreach v ∈ V do
2 |   avg ← (0,0,0)
3 |   foreach e ∈ E do
4 |   |   avg ← length(e)
5 |   end
6 |   avg ← avg/length(E)
7 end
```

Then, we assume that the average edge length is equal to one unit in our rescaled space, such that our scale factor is equal to $1/avg$, where avg is the average edge length. In Algorithm 5 we show how to compute the new positions.

Algorithm 5: Normalization

Input : V,E
Output: X

```
foreach v ∈ V do
    avg ← averageEdgeLength(E)
    scaleFactor ← 1/avg
    foreach v ∈ V do
        X(i) ← x(v) * scaleFactor
    end
end
```

Due that it is important to retrieve the mesh in the original scale, we can use the inverse of the scale factor to compute the positions in our original space. In Algorithm 6 we show how to do it.

Algorithm 6: Retrieval

Input : V, $scaleFactor$
Output: X

```
foreach v ∈ V do
    ret ← 1 / scaleFactor
    foreach v ∈ V do
        X(i) ← x(v) * ret
    end
end
```


5.2 Adaptive patches computation

In this step, we compute the adaptive patch (defined by a membership function) for each face of the mesh. So, we first have to compute the areas, centroids, normals and the distance matrix, to use them in the adaptive patch computation algorithm. In Algorithm 7 we show how to compute all membership functions and store them as a membership matrix. The latter is a sparse matrix containing 0s as null values and the corresponding membership values as not null values. We follow the same idea used for the distance matrix.

Algorithm 7: Compute adaptive patches

Input : $mesh$, $areas$, $centroids$, $normals$, $distanceMatrix$, α, β, γ, δ

Output: $membershipMatrix$

```
foreach f_i ∈ F do
    membershipMatrix(i) ← adaptivePatch(mesh, areas,
      centroids, normals, distanceMatrix, f_i, α, β, γ, δ, a_0)
end
```

5.3 Adaptive patch based normal filtering

Once we have the membership functions, we use them to filter the normals. Membership functions work as local kernels in a filtering process, so we can use the membership values as weights to filter normals. The new normal of a face f can be computed as follows:

$$\mathbf{n}' = \sum_{f_i \in F} \mathbf{n}_i u_i a_i, \tag{5-1}$$

where u_i is the membership value of face f_i, a_i is the area of face f_i, and $\mathbf{n}'$ the new normal which should be normalized. We update the normals in an iterative manner.

In Algorithm 8 we show the implementation of this step. In the algorithm, $mesh$ is the input mesh with F as corresponding faces, $areas$ is an array containing face areas, $centroids$ is an array containing face centroids, $normals$ is an array containing face normals, $membershipMatrix$ is a sparse matrix containing all membership functions, $iterations$ is the number of filtering iterations to be performed, $facesOfNZV$ returns the faces with non zero membership value, $normalsOfNZV$ returns the normals of faces with non zero membership value, $nonZeroValues$ return non zero membership values, $areasOfNZV$ returns the areas of faces with non zero membership value, and $normalize$ return the unitary vector of a given vector.

Algorithm 8: Patch Based Filtering

Input : $mesh$, $areas$, $normals$, $membershipMatrix$, $iterations$
Output: $newNormals$

```
for i ← 1 to iterations do
    foreach f_i ∈ F do
        F_i ← facesOfNZV(membershipMatrix(i))
        N_i ← normalsOfNZV(membershipMatrix(i))
        u ← nonZeroValues(membershipMatrix(i))
        a ← areasOfNZV(membershipMatrix(i))
        newNormal ← (0, 0, 0)
        foreach f_j ∈ F_i do
            newNormal ← newNormal + N_i(j) * u(j) * a(j)
        end
        newNormals(i) ← normalize(newNormal)
    end
    normals ← newNormals
end
```


5.4 Bilateral normal filtering

After we apply the adaptive patch based filtering we filter the normals in a bilateral manner following the approach of [29]. The difference is that we use the neighborhoods and distances defined in our sparse distance matrix. So, for a face f_i we can compute the new normal as follows:

$$\mathbf{n}'_i = \frac{1}{W(f_i)} \sum_{f_j \in F} A_j K_c(distanceMatrix(i,j)) K_s(||\mathbf{n}_j - \mathbf{n}_i||)\mathbf{n}_j. \qquad (5\text{-}2)$$

We update these normals in an iterative manner.

We show the implementation of Algorithm 9, where $mesh$ is the input mesh with F as corresponding faces, $areas$ is an array containing face areas, $centroids$ is an array containing face centroids, $normals$ is an array containing face normals, $membershipMatrix$ is a sparse matrix containing all membership functions, $iterations$ is the number of filtering iterations to be performed, $facesOfNZV$ returns the faces with non zero membership value, $normalsOfNZV$ returns the normals of faces with non zero membership value, $centroidsOfNZV$ returns the centroids of faces with non zero membership value, σ_c a parameter of the spatial weight gaussian function, σ_s a parameter of the signal weight gaussian function, $nonZeroValues$ return non zero

membership values, $areasOfNZV$ returns the areas of faces with non zero membership value.

Algorithm 9: Bilateral Normal Filtering

Input : $mesh$, $areas$, $normals$, $centroids$, $distanceMatrix$, $iterations$, σ_c, σ_s

Output: $newNormals$

for $i \leftarrow 1$ **to** $iterations$ **do**
  foreach $f_i \in F$ **do**
    $F_i \leftarrow facesOfNZV(distanceMatrix(i))$
    $N_i \leftarrow normalsOfNZV(distanceMatrix(i))$
    $C_i \leftarrow centroidsOfNZV(distanceMatrix(i))$
    $\mathbf{u} \leftarrow nonZeroValues(distanceMatrix(i))$
    $\mathbf{a} \leftarrow areasOfNZV(distanceMatrix(i))$
    $newNormal \leftarrow (0, 0, 0)$
    $weightSum \leftarrow 0$
    foreach $f_j \in F_i$ **do**
      $spatialDistance \leftarrow length(centroids(i) - C_i(j))$
      $signalDistance \leftarrow length(normals(i) - N_i(j))$
      $spatialWeight \leftarrow exp(-0.5 * spatialDistance^2/\sigma_c^2)$
      $signalWeight \leftarrow exp(-0.5 * signalDistance^2/\sigma_s^2)$
      $weight \leftarrow \mathbf{a}(j) * spatialWeight * signalWeight$
      $newNormal \leftarrow newNormal + N_i(j) * weight$
      $weightSum \leftarrow weightSum + weight$
    end
    $newNormals(i) \leftarrow newNormal/weightSum$
  end
  $normals \leftarrow newNormals$
end

5.5 Vertex updating

With a set of filtered normals now we have to adapt the vertex positions to fit the involved faces to the new face normals. Taubin proposed in [26] to use orthogonality between the new normal and the edges of the corresponding face, such that the vertex updating step minimizes the following energy:

$$E(M) = \begin{cases} \mathbf{n}_f \cdot (\mathbf{x}_j - \mathbf{x}_i) \\ \mathbf{n}_f \cdot (\mathbf{x}_k - \mathbf{x}_j) \\ \mathbf{n}_f \cdot (\mathbf{x}_i - \mathbf{x}_k) \end{cases} \quad , \quad \forall f(i,j,k) \tag{5-3}$$

where $\mathbf{n}_f$ is the normal of face f and i, j, k its corresponding vertex indices. In a few words the optimization problem tries to find new vertex positions which preserve the normal $\mathbf{n}_f$. There are different ways to approximate a solution to this problem. In this work we adopt the approach of [28], which defines the new vertex position as:

$$\mathbf{x}'_i = \mathbf{x}_i + \frac{1}{|F_v(v_i)|} \sum_{f_k \in F_v(v_i)} \mathbf{n}'_k(\mathbf{n}'_k \cdot (\mathbf{c}_k - \mathbf{x}_i)) \tag{5-4}$$

where $F_v(v_i)$ represents the set of faces shared by the vertex v_i, $\mathbf{n}'_k$ the new normal of face f_k and $\mathbf{c}_k$ the centroid of f_k. We perform this update iteratively (vertex iterations).

In Algorithm 10 we show how to implement this step, where *mesh* is the input mesh, *newNormals* is an array containing the desired face normals, and *iterations* is the number of iterations to be performed.

Algorithm 10: Vertex updating

Input : *mesh*, *newNormals*, *iterations*
Output: *newPositions*

```
1  for i ← 1 to iterations do
2      foreach v_i ∈ V do
3          position ← x(v_i) tempPosition ← (0, 0, 0)
4          numFaces ← 0
5          foreach f_j ∈ N_f(v_i) do
6              tempPosition ← tempPosition + newNormals(j) *
                ⟨newNormals(j), (centroids(j) − position)⟩
7              numFaces ← numFaces + 1
8          end
9          newPositions(i) ← position + tempPosition
10     end
11     normals ← newNormals
12 end
```

5.6 Pipeline

Inspired by real noisy data processing, we proposed this algorithm for meshes with low noise intensity, similar to the noise produced in range scans or medical data reconstructions. In the case that the intensity of the noise is too high, we previously filter the mesh using a few iterations of bilateral filtering of normals with vertex updating [29]. We opted to prefilter the mesh in these cases because the adaptive patch computation can compute not desired solutions that can introduce artifacts when performing the filtering step. The latter happens due that the high normal variation in the input mesh turns unstable the optimization.

The proposed denoising algorithm works as follows. We receive as input a noisy mesh and return as output a denoised mesh. Then, we have to compute the information that normal filters need, such as areas, centroids and normals (precomputation). Using the computed centroids we obtain the distances between them. With this information, we compute the adaptive patches that we are going to use in the filtering process (adaptive patches computation). We filter the normal field using the adaptive patch based scheme, performing a number of iterations (adaptive patch based normal filtering). Then, we filter the new normal field using a bilateral scheme in an iterative manner (bilateral normal filtering). With these normals we perform a vertex updating step with a given number of iterations. All of these steps can be executed iteratively too (external iterations). We have several parameters involved here: the number of external iterations ($iterations$), the number of patch based filtering iterations ($patchFilteringIterations$), the number of bilateral normal filtering iterations ($bilateralNormalIterations$), the number of vertex updating iterations ($vertexIterations$), the maximum distance considered in the distance matrix ($maxDistance$), the maximum number of variables considered for the optimization problems ($maxVars$), the parameters of adaptive patch computation (α, β, γ and δ) and the parameters of the kernels in the bilateral normal filtering (σ_c and σ_s). In Figure 5.1 we show the pipeline of the algorithm and in Algorithm 11 the pseudocode.

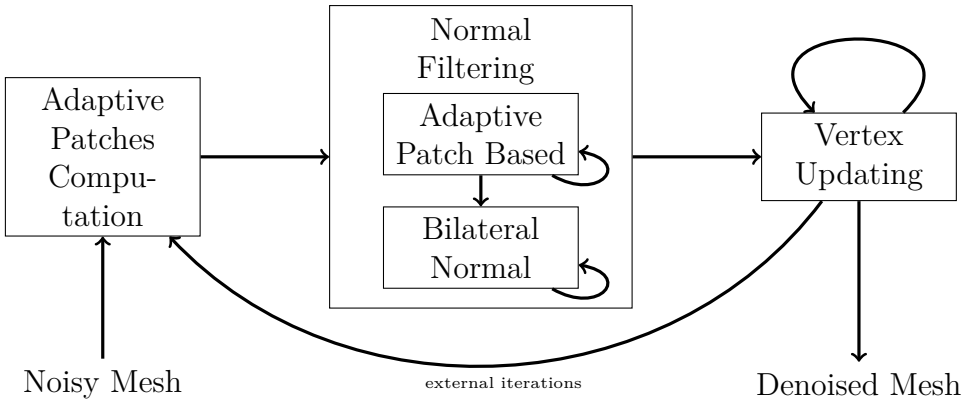


Figure 5.1: Pipeline of the proposed denoising algorithm.

Algorithm 11: Denoising

Input : $mesh$, $iterations$, $patchFilteringIterations$, $bilateralNormalIterations$, $vertexIterations$, $maxDistance$, $maxVars$, α, β, γ, δ, σ_c, σ_s

Output: $positions$

for $i \leftarrow 1$ **to** $iterations$ **do**
- $(areas, centroids, normals) = precomputation(mesh)$
- $distanceMatrix = computeDistances(mesh, centroids, maxDistance, maxVars)$
- $membershipMatrix = computeAdaptivePatches(mesh, areas, centroids, normals, distanceMatrix, \alpha, \beta, \gamma, \delta)$
- $normals \leftarrow patchBasedFiltering(mesh, areas, normals, membershipMatrix, patchFilteringIterations)$
- $normals \leftarrow bilateralNormalFiltering(mesh, areas, normals, centroids, distanceMatrix, bilateralNormalIterations, \sigma_c, \sigma_s)$
- $positions \leftarrow vertexUpdating(mesh, normals, vertexIterations)$

end

6
Denoising algorithm evaluation

This chapter discusses different strategies to evaluate visually and numerically the results of a denoising algorithm.

6.1
Visual Comparison

We can evaluate the results of denoising algorithms with different visualization methods. In this work, we will use three. The first one is a simple rendering of the mesh using flat shading. The color of the faces in this mode is helpful for noise identification. The second one is a rendering using mean curvature values as vertex colors, in an RGB scale from red to blue. The curvature is a good feature to compare the smoothness of two meshes. Also, it is related to a metric explained in the next section. And the last one is a normal color mapping over the mesh. This approach works like the flat shading but with more differentiation. All the visualization methods proposed here were used in several works about mesh smoothing. In Figure 6.1 we show an example.

6.2
Metrics

Geometry processing algorithms are usually evaluated measuring the similarity between the resulting output and the desired output (ground truth). In the case of 2-manifolds, which can be represented as meshes, several similarity metrics were proposed in the literature [37, 38, 39, 40, 41, 27,

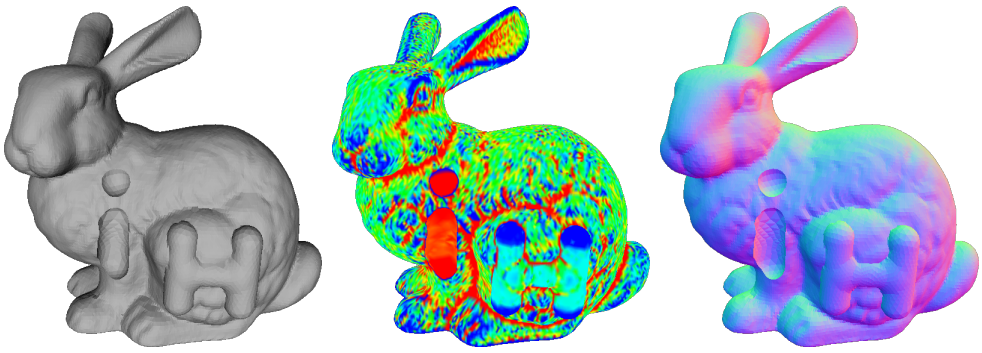


Figure 6.1: Visualization methods. From left to right: Flat shading, curvature color mapping and normal mapping.

42]. Most of them are related to measuring the 3D distance between the manifolds, such as the computation of the Hausdorff distance. Another family of metrics measures the difference regarding intrinsic or extrinsic properties of the manifold (e.g., curvature, normals, etc.). In the next section, we will explain some of these metrics which are relevant to the specific problem of mesh denoising. The classification of the metrics is shown in Table 6.1.

Type	**Metrics**
Distance based	Mean distance error [37] Quadric error metric [43] L^2 vertex-based mesh-to-mesh error metric [40, 41]
Normal based	Tangential Error Metric [39] L^2 normal-based mesh-to-mesh error metric [40, 41] Mean Square Angular Error [38, 27]
Curvature based	Discrete curvature error metric [39]

Table 6.1: Classification of metrics

In this section, we will denote the elements of the evaluated result with an apostrophe ($'$) and the elements of the desired result without it. We use M to represent a mesh conformed by a set of vertices V and a set of faces F. The coordinates of elements of V are represented as $\mathbf{X}$. When we denote the area of a vertex, it is equal to the barycentric area, and when we denote the area of a face it is equal to the triangle area. The centroid of a face is denoted by $\mathbf{c}$ and the normal as $\mathbf{n}$. $N_f(v)$ represents the neighboring faces of vertex v, $N_v(v)$ represents the neighboring vertices of vertex v, and $N_f(f)$ represents the neighboring faces of face f.

6.2.1 Mean distance error metric:

Let us denote by $d(x_1, x_2)$ the euclidean distance between x_1 and x_2. The minimum distance between a point x and a 2-manifold S is defined as follows:

$$e(x, S) = \min_{x' \in S} d(x, x'). \tag{6-1}$$

The Hausdorff distance measures how far two manifolds are from each other. Given two manifolds S' and S, the Hausdorff distance (d_H) between them is defined as:

$$d_H(S_1, S_2) = \max(E(S_1, S_2), E(S_2, S_1)) \tag{6-2}$$

$$E(S_1, S_2) = \max_{p \in S_1}(e(p, S_2)). \tag{6-3}$$

Using a one-side distance, i.e. $d(S_1, S_2) = E(S_1, S_2)$, Cignoni et al. introduced and implemented the mean distance for triangular meshes [37]. It uses regular sampled distances to compute a total distance between them. In the continuous setting, the mean distance is defined as follows:

$$E_M(S_1, S_2) = \frac{1}{area(S_1)} \int_{S_1} e(p, S_2) ds. \tag{6-4}$$

Adapting it to triangular meshes and using our notation:

$$E_M(M', M) = \frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} area(v'_i) dist(\mathbf{x}'_i, T), \tag{6-5}$$

where $T \in F$ is the nearest triangle to the point $\mathbf{x}'_i$

6.2.2 Quadric error metric:

The quadric error metric was introduced by Garland and Heckbert in [43], and then used in [39] to define another metric. Given a point $\mathbf{x}'$ of the resulting mesh M', the quadric which represents the distance between it and a point $\mathbf{x}$ (point of vertex v) of the desired mesh M, is computed as follows:

$$Q^v(\mathbf{x}') = \sum_{f \in \mathcal{N}_f(v)} area(f) Q^f(\mathbf{x}'), \tag{6-6}$$

where $Q^f(\mathbf{x}')$ measures the squared distance between the point $\mathbf{x}'$ and the plane generated by the face f. This quadric $Q^f(\mathbf{x}')$ can be arranged by:

$$Q^f(\mathbf{x}') = (\mathbf{n}^T \mathbf{x}' + d)^2, \tag{6-7}$$

where n is the unit normal at face f, $d = -\mathbf{n}^T \mathbf{x_k}$ and $\mathbf{x_k}$ is any point defined on f. In this work, we compute the mean quadric error, using vertex areas as weights:

$$QE_v = \frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} area(v'_i) Q^v(\mathbf{x}'_i), \tag{6-8}$$

where $v \in V$ is the nearest vertex to $\mathbf{x}'_i$.

6.2.3 Tangential error metric:

This metric was introduced in [39]. It measures the magnitude of a difference vector between two normals. In the case of mesh smoothing, these

normals correspond to the resulting vertex and desired vertex respectively. The unit normal of a vertex can be approximated as a weighted sum of neighboring faces normals:

$$\mathbf{n}_i = \sum_{f_j \in \mathcal{N}_f(v_i)} w_j \mathbf{n}_j. \tag{6-9}$$

The weights w_j usually are the face areas, i.e. $area(f_j)$. Lets assume that $\mathbf{n}_i$ is normalized (i.e. $\mathbf{n}_i = \mathbf{n}_i/||\mathbf{n}_i||$). The error between the resulting normal $\mathbf{n}'$ and the desired normal $\mathbf{n}$ which is the normal of v, is defined as follows:

$$T^v(\mathbf{n}') = ||\mathbf{n} - \mathbf{n}'||, \tag{6-10}$$

where $\mathbf{n}$ is the normal of vertex $v \in V$. Considering all the mesh, we can compute the mean tangential error metric:

$$TE_v = \frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} area(v'_i) T^v(\mathbf{n}'_i), \tag{6-11}$$

where $v \in V$ is the nearest vertex to $\mathbf{v}'_i$.

6.2.4
L^2 vertex-based mesh-to-mesh error metric:

This metric measures the L^2 average distance between each vertex of the resulting mesh and the nearest triangle of the desired mesh, using barycentric area as weight and total area as normalization factor [40, 41]. It is defined as follows:

$$E_v = \sqrt{\frac{1}{3\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} \sum_{f'_j \in \mathcal{N}_f(v'_i)} area(f'_j) dist(\mathbf{x}'_i, T)^2}, \tag{6-12}$$

where $T \in F$ is the nearest triangle to the point $\mathbf{x}'_i$ and $dist$ measures the L^2 distance between them. This expression can also be written in the following form:

$$E_v = \sqrt{\frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} area(v'_i) dist(\mathbf{x}'_i, T)^2}. \tag{6-13}$$

6.2.5
L^2 normal-based mesh-to-mesh error metric:

It measures deviations between the corresponding face (triangle) normals of two meshes M and M' [40, 41]. Given a face $f'_i \in F'$ with normal $\mathbf{n_i}$, and

its corresponding nearest face $f \in F$ with normal $\mathbf{n}$, the L^2 normal-based mesh-to-mesh error metric is defined as:

$$E_n = \sqrt{\frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{f'_i \in F'} area(f'_i)||\mathbf{n} - \mathbf{n}'_i||^2}. \quad (6\text{-}14)$$

6.2.6
Mean square angular error metric:

The Mean Square Angular Error (MSAE) is a normal error metric which measures the angle difference between the resulting face normal and the corresponding desired normal [38, 27]:

$$MSAE = E[\angle(\mathbf{n}, \mathbf{n}')], \quad (6\text{-}15)$$

where $\mathbf{n}$ is the corresponding desired normal of $\mathbf{n}'$. It can be computed as follows:

$$MSAE = \frac{1}{n'_f} \sum_{f'_i \in F'} acos(\mathbf{n}^T \mathbf{n}'_i), \quad (6\text{-}16)$$

where n'_f is the number of faces of the resulting mesh M', and $\mathbf{n}$ is the normal of the nearest face $f \in F$ to face $f'_i \in F'$.

6.2.7
Discrete curvature error metric:

Kim et al. used curvature to define a robust metric for evaluation of mesh simpilification algorithms [39]. Their approach uses gaussian curvature, mean curvature or principal curvatures to define curvature on each vertex of a triangle mesh. Assuming that we have a curvature mapping for each vertex $c : V \to \mathbb{R}$, the discrete curvature error metric for a resulting vertex $v' \in V'$ regarding its corresponding desired vertex $v \in V$ (nearest vertex in M), is defined as follows:

$$C^v(v') = ||c(v) - c(v')||. \quad (6\text{-}17)$$

The mean discrete curvature error can be computed by:

$$CE_v = \frac{1}{\sum_{f'_k \in F'} area(f'_k)} \sum_{v'_i \in V'} area(v'_i) C^v(v'_i), \quad (6\text{-}18)$$

where $v \in V$ is the nearest vertex to $v' \in V'$. Based on [44], the discretization

of curvatures are described in the following sections.

In our experiments, we only use the mean curvature to define the curvature of a mesh. The mean curvature is defined by $\kappa_H = (\kappa_1 + \kappa_2)/2$. Using the Laplace-Beltrami operator based on cotangent weights, mean curvature can be discretized as:

$$K(v) = \frac{1}{2area(v)} \sum_{v_i \in \mathcal{N}_v(v)} (\cot \alpha_i + \cot \beta_i)(\mathbf{x} - \mathbf{x}_i), \tag{6-19}$$

$$\kappa_H(v) = \frac{||K(v)||}{2}, \tag{6-20}$$

where α_i and β_i are the opposite angles to the edge which connects vertex v and v_i.

6.2.8 Area difference and volume difference:

To measure area and volume preservation, we compute the difference of area and volume between the resulting mesh M' and the desired mesh M. The area of a mesh is calculated as the sum of all faces of a mesh: $\sum_{f \in F} area(f)$. The volume of a mesh is computed following the approach of Zhang and Chen [45], where the total mesh volume is computed as the sum of the signed volumes of tetrahedrons formed by the joint of each triangle and the origin. Given a triangle f with points $A = (x_1, y_1, z_1)$, $B = (x_2, y_2, z_2)$ and $C = (x_3, y_3, z_3)$, whose orientation for normal computation is ACB, the signed volume of the corresponding tetrahedron is computed as:

$$TetVol(f) = \frac{1}{6}(-x_3y_2z_1 + x_2y_3z_1 + x_3y_1z_2 - x_1y_3z_2 - x_2y_1z_3 + x_1y_2z_3). \tag{6-21}$$

The total mesh volume is defined as:

$$Vol(M) \sum_{f_i \in F} TetVol(f_i). \tag{6-22}$$

7
Results

To evaluate the proposed method, this chapter presents some visual and numerical results using synthetic and real data.

7.1
Implementation details

All of our code was implemented using C++ programming language. To reduce the time complexity and memory space consumption (space complexity) for mesh data representation, we need to use efficient data structures. For this reason, we used an implementation of a half-edge data structure contained in OpenMesh library [46]. Neighbors, areas, normals, centroids, and angles can be computed navigating over the mesh using iterators and circulators of this data structure. To solve the quadratic optimization problems we used the CPLEX library[1] which supports all constraints that we define. Also, we developed an interface to interact with patch behavior using Qt library and OpenGL. This interface was useful to formulate the optimization problem and to find suitable parameters for our experiments.

In the case of metrics implementation, we used a KD-Tree data structure for nearest neighbor search (i.e., nanoflann [47]). The problem of finding the nearest vertex of a target vertex, it can be obtained using a simple nearest neighbor query in the KD-Tree. The problem is more complicated when finding the nearest triangle to a target vertex or the nearest triangle to a target triangle. In both cases, we first select a set of candidate triangles using a range query and then use brute force to select the nearest triangle of this set. Given a set $A = X \cup C$, where C is a set containing all face centroids, the set of candidate triangles (faces) F_c is defined as:

$$F_c = \{f \in F | f \supset \mathbf{b}, \mathbf{b} \in B\},$$

$$B = \{\mathbf{a} \in A | dist(\mathbf{a}, \mathbf{p}) \leq r\},$$

where $\mathbf{p}$ is the target point, r is the tolerance radius, and $f \supset \mathbf{b}$ denotes that face f contains the point $\mathbf{b}$. In other words, the candidate faces are all faces

[1] https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer

containing points of **A** which are within the tolerance radius. The tolerance radius we use is equal to two times the average edge length. In the case of the nearest triangle to a vertex, we use the coordinate of the target vertex as **p**. In the case of the nearest triangle to a triangle, we use the centroid of the target triangle as **p**. To compute the distance between a point and a triangle, we just project the point on the triangle plane and find the nearest edge or vertex regarding the projection. In the case of the triangle to triangle distance, we use the same method of point to triangle distance, using the centroid as point. In [48], the authors describe the point to triangle distance in more detail.

All our experiments were performed on an Intel (R) Core (TM) i7-4770 CPU @ 3.40GHz processor with 16,0 GB RAM and Windows 8.1 64-bit operating system.

7.2 Datasets

All the meshes used in these experiments were obtained from the AIM@SHAPE Shape Repository [2], The Stanford 3D Scanning Repository[2] and the SHREC15: Range Scans based 3D Shape Retrieval [49]. Most of the images were generated using MeshLab[3]. For some algorithms, we used the implementation of [31] provided in their repository[4]. The parameters of other algorithm for our test cases were based on the parameters provided in the corresponding work. In the case that the test case was not used in a work, we adjust manually the parameters following the recommendations or adopting similar parameters used for similar shapes.

7.3 Algorithm parameters

Our method was compared with the following anisotropic denoising algorithms: Bilateral mesh denoising [23], Non-iterative, feature-preserving mesh smoothing [24], Fast and effective feature-preserving mesh denoising [28], Bilateral normal filtering for mesh denoising [29], Mesh denoising via L_0 minimization [22], Guided mesh normal filtering [31], Mesh denoising based on normal voting tensor and binary optimization [33], and Robust and High Fidelity Mesh Denoising [34]. The parameters used for this comparison are detailed in Tables 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7 and 7.8 respectively, where $||e||$ is the average edge length. For parameter explanation, please read the corresponding reference or the supplementary material provided in [31].

[2]http://graphics.stanford.edu/data/3Dscanrep/
[3]http://www.meshlab.net/
[4]https://github.com/bldeng/GuidedDenoising

Table 7.1: Parameters used for Bilateral mesh denoising [23].

mesh	iterations
Block	15
Devil	12
Fandisk	10
Joint	15
Balljoint	10
Gargoyle	20
Building	20
Keyboard	20

Table 7.2: Parameters used for Non-iterative, feature-preserving mesh smoothing [24].

mesh	σ_f	σ_g
Block	$1.3\|\|e\|\|$	$1.2\|\|e\|\|$
Devil	$1.4\|\|e\|\|$	$1.6\|\|e\|\|$
Fandisk	$1.3\|\|e\|\|$	$1.4\|\|e\|\|$
Joint	$1.3\|\|e\|\|$	$1.2\|\|e\|\|$
Balljoint	$1.0\|\|e\|\|$	$1.0\|\|e\|\|$
Gargoyle	$1.0\|\|e\|\|$	$1.0\|\|e\|\|$
Building	$1.0\|\|e\|\|$	$1.0\|\|e\|\|$
Keyboard	$1.0\|\|e\|\|$	$1.0\|\|e\|\|$

Table 7.3: Parameters used for Fast and effective feature-preserving mesh denoising [28]

mesh	T	normal iterations	vertex iterations
Block	0.4	30	20
Devil	0.5	8	8
Fandisk	0.55	20	40
Joint	0.4	30	20
Balljoint	0.9	10	10
Gargoyle	0.5	10	10
Building	0.5	5	10
Keyboard	0.5	5	10

The parameters used for our proposal are described in Table 7.9. We use $2||e||$ as maximum distance and 100 as the maximum number of variables for the optimization problems (with the exception that sharpSphere uses 20 as the maximum number of variables, and dragon do not use a specific maximum distance or maximum number of variables). The area constraint we selected is equal to 20% of the total area of the neighborhood delimited by maximum distance and the maximum number of variables. The bilateral normal filtering step is executed with $\sigma_s = 0.35$ and σ_c as the average distance between face

Table 7.4: Parameters used for Bilateral normal filtering for mesh denoising [29].

mesh	σ_s	normal iterations	vertex iterations
Block	0.35	45	20
Devil	0.45	6	6
Fandisk	0.35	25	20
Joint	0.35	45	20
Balljoint	0.7	10	10
Gargoyle	0.7	10	10
Building	0.7	5	10
Keyboard	0.7	5	10

Table 7.5: Parameters used for Mesh denoising via L_0 minimization [22].

mesh	λ	α_0	β_0	μ_α	μ_β	β_{max}
Block	0.55	0.00389	0.001	0.5	1.414	1000
Devil	0.01	0.00149	0.001	0.5	1.414	1000
Fandisk	0.01	0.00346	0.001	0.5	1.414	1000
Joint	0.55	0.00389	0.001	0.5	1.414	1000
Balljoint	0.1	0.0001	0.001	0.5	1.414	1000
Gargoyle	0.1	0.0001	0.001	0.5	1.414	1000
Building	0.01	0.0001	0.001	0.5	1.414	1000
Keyboard	0.1	0.0001	0.001	0.5	1.414	1000

Table 7.6: Parameters used for Guided mesh normal filtering [31].

mesh	σ_c	σ_s	normal iterations	vertex iterations
Block	$2\|\|e\|\|$	0.3	40	30
Devil	$2\|\|e\|\|$	0.35	6	6
Fandisk	$2\|\|e\|\|$	0.25	25	20
Joint	$2\|\|e\|\|$	0.3	40	30
Balljoint	$2\|\|e\|\|$	0.7	10	10
Gargoyle	$2\|\|e\|\|$	0.7	10	10
Building	$2\|\|e\|\|$	0.7	5	10
Keyboard	$2\|\|e\|\|$	0.7	5	10

centroids.

7.4 Initial Comparison

As a first experiment, working on the sharpSphere mesh, we want to show the behavior of our algorithm compared to the uniform Laplacian smoothing. Uniform Laplacian smoothing is an isotropic denoising method (does not preserve details and produces mesh shrinkage). We corrupted the mesh with

Table 7.7: Parameters used for Mesh denoising based on normal voting tensor and binary optimization [33].

mesh	radius	τ	iterations
Devil	0.1	1.0	30
Fandisk	0.3	0.2	50
Joint	0.3	0.05	60
Balljoint	ND	ND	ND
Gargoyle	ND	ND	ND

Table 7.8: Parameters used for Robust and High Fidelity Mesh Denoising [34].

mesh	σ_s	λ_I	iterations
Fandisk	0.55	0.2	100
Joint	1	0.2	150

Table 7.9: Parameters used for our proposal.

mesh	α	β	γ	δ	external iterations	patch based iterations	bilateral iterations	vertex iterations
SharpSphere	1.0	2.0	0.8	8	3	3	5	10
Dragon	1.0	1.0	0.2	10	2	3	0	10
Block	1.0	1.0	0.1	30	3	5	2	10
Devil	1.0	1.0	0.01	40	1	5	2	10
Fandisk	1.0	1.0	0.2	10	3	5	2	10
Joint	1.0	2.0	0.3	30	3	6	2	10
Balljoint	1.0	1.0	0.1	5	2	3	2	10
Gargoyle	1.0	1.0	0.2	10	2	5	1	10
Building	1.0	1.0	0.2	20	2	3	1	10
Keyboard	1.0	1.0	0.2	20	2	3	1	10

artificial noise (i.e., Gaussian noise following normal direction and σ equal to 0.1 times the average edge length), and then performed the uniform Laplacian smoothing and our method over it. In the case of our method, we tested it using the bilateral normal filtering step and without using it. In Figure 7.1 we show the corresponding results and in Figure 7.2 the corresponding curvature mapping.

In Table 7.10 we show the error between the original mesh (not corrupted) and the resulting one, using all metrics presented before. The label "ours" represents our method without bilateral filtering step and the label "ours (*)" our method using it.

As we can see, our algorithm successfully removes the noise while preserving details without introducing too much deformation. The resulting

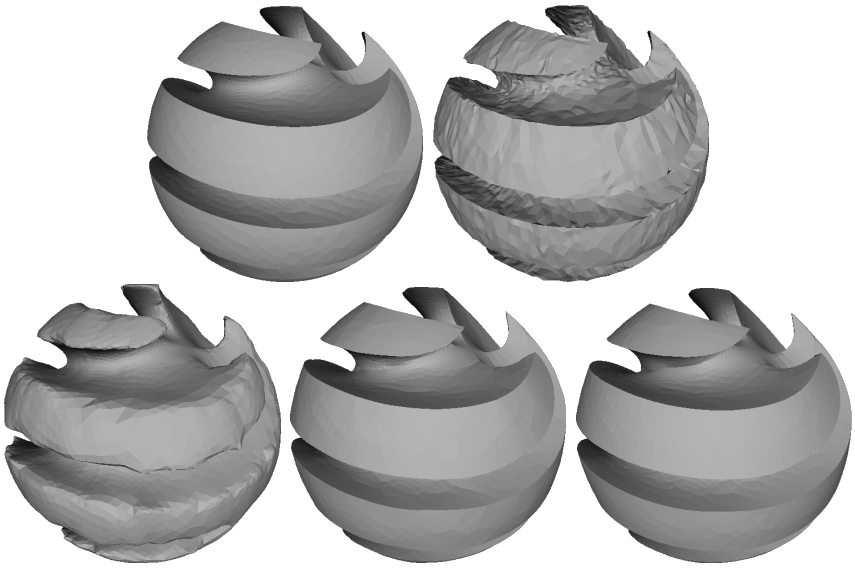


Figure 7.1: An example using our denoising algorithm. First row from left to right: original mesh and noisy mesh. Second row from left to right: resulting mesh using uniform Laplacian smoothing, resulting mesh of our proposal wihout using the bilateral normal filtering step, and resulting mesh of our proposal using it.

shape is very near to the original one. The Laplacian smoothing does not preserve details, introduces deformation and suffers shrinkage. The latter is based on the visualization of the resulting mesh and the high error for each metric. The resulting mesh which uses the bilateral normal filtering step looks more smooth than the other, but it is worse regarding the quantitative analysis (metrics). Depending on the application we can choose the behavior of our proposed algorithm.

Table 7.10: Results for sharpSphere mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
Uniform	0.010836	0.022663	0.019531	8.975630	0.059854	0.086991	0.192965	0.120947	0.029940
Ours	0.000962	0.001673	0.001288	6.572030	0.012719	0.011032	0.076498	0.005788	0.000364
Ours (*)	0.001548	0.003932	0.001643	9.372330	0.036556	0.016079	0.085570	0.009088	0.001526

In [31] the authors proposed the computation of consistent patches to obtain a guidance normal. Due that our proposal was inspired by this idea we can compare these guidance normals and the average normals weighted by our patch membership functions. So, using the same corrupted mesh, we computed the corresponding normals and compared them using the MSAE and the L2 Normal Based metric. In Table 7.11 we show these errors.

Our estimated normal field have a lower error than the guidance normal field of [31]. To show it visually, we performed a vertex updating step using

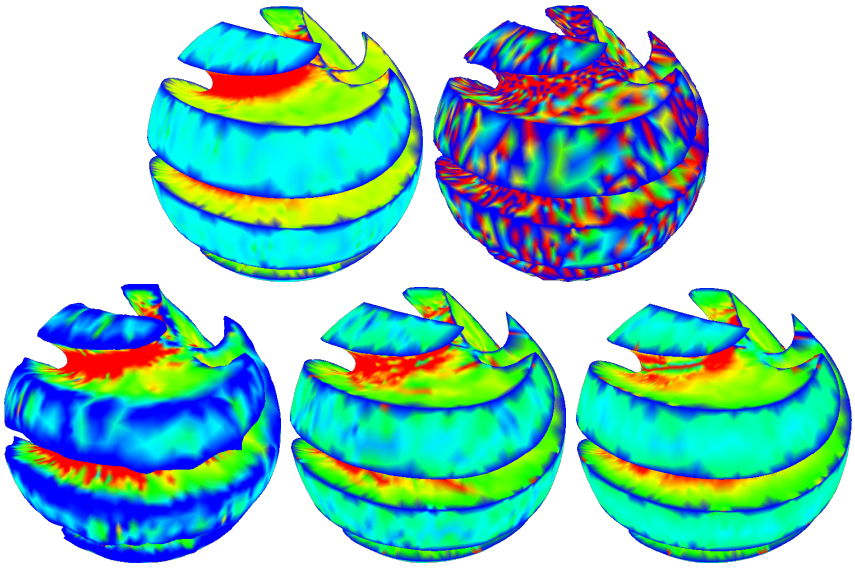


Figure 7.2: An example using our denoising algorithm (showing curvature mapping). First row from left to right: original mesh and noisy mesh. Second row from left to right: resulting mesh using uniform Laplacian smoothing, resulting mesh of our proposal wihout using the bilateral normal filtering step, and resulting mesh of our proposal using it.

Table 7.11: Errors of estimated normals

	MSAE	L2 Normal Based
Guidance normals [31]	8.34329	0.01603
Our normals	5.90240	0.00816

these normals and 20 as the number of iterations. In Figure 7.3 we show the generated meshes.

Based on the latter experiment, we can say that our estimated normal field is better than the guidance normal field estimated in [31]. Our approach generates less flat regions when the triangulation is irregular. Their approach generate large flat regions because their patch selection depends on regular topology neighborhoods.

To have a notion of the execution time of each step of our denoising algorithm, in Table 7.12 we show the timing values for the example presented in this section. We show the computation of the distance matrix, the computation of the adaptive patches, the execution of patch-based normal filtering and bilateral normal filtering, and the vertex updating step. As we can see, the bottleneck of our algorithm is in the computation of adaptive patches, due that quadratic optimization problems have high computational cost.

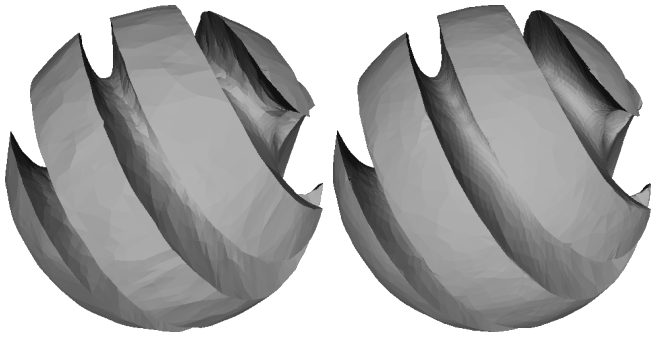


Figure 7.3: Results after 20 iterations of vertex updating using estimated normals. Left: Guided normals using [31]. Right: Average normal weighted by patch membership function.

Table 7.12: Time for each step of the algorithm performed over sharpSphere mesh (10443 vertices and 20882 faces). Total time equal to 51.472.

Step	time	percentage
Distance computation	1.379s	7%
Adaptive patches	15.676s	84%
Normal filtering	1.524s	8%
Vertex updating	0.023s	1%

7.5 Comparison with other algorithms using artificial noise

A common way to measure the effectiveness of a mesh denoising algorithm is by generating synthetic data. We add artificial noise to a clean mesh, perform a denoising algorithm over it, and then measure the similarity between the denoised mesh and the clean mesh. When having a noisy mesh as input without an optimal version (clean mesh) to compare it (e.g., 3D scan, isosurface from medical volume, etc.), the results can only be evaluated visually.

Some metrics presented in this work were designed for error measurement of remeshing algorithms, but are useful for denoising algorithms too. In fact, several works about mesh denoising use them for comparison purposes. In general, all metrics measure the similarity between two meshes regarding a specific feature (coordinates, normal field or curvature).

Distance-based metrics are the most commonly used, for this reason, we use different approaches measuring mesh-to-mesh distance; These approaches use sampled distances based on vertex-to-vertex distance or vertex-to-face distance. This will help us to measure how far, in a spatial sense, is the denoised mesh to the original (clean mesh).

The normal field of a mesh describes the shape of the mesh on the Gaussian sphere (first-order behavior). Comparing this mapping is helpful to

define how similar is a shape to another. So, this kind of metrics measures how similar are two meshes regarding their shapes. In the context of mesh denoising, two shapes can be spatially very close but far regarding the normal field.

The curvature is a feature which describes the variation of the normal field (second-order behavior). In that sense, curvature will give us a higher order feature to compare two meshes. Following the idea of [39], it will give us a more robust comparison.

Area difference and volume difference are helpful to measure robustness regarding area and volume preservation, and consequently to have the notion if the mesh was shrunken or deformed too much.

We will present four test cases to compare the results visually (flat shading, curvature mapping, and normal mapping) and using all the exposed metrics.

7.5.1 Block mesh

This is an irregular mesh presenting sharp features with large flat and rounded areas. We corrupted the mesh with artificial Gaussian noise following vertex normal directions. The standard deviation σ used for the Gaussian distribution is equal to $0.1l$, where l is the average edge length of the corresponding mesh.

In Figure 7.5, we show the results obtained through this mesh. As we can see, the results of [23] and [24] are not good; their approaches deform too much the shape. [31] introduces an area with wrong normal direction in some corners of the shape (See Figure 7.4). [28] and [29] generate smooth meshes with sharp feature preservation. But the original mesh has small flat regions in rounded areas, it is not totally smooth there (see the curvature mapping for better understanding). Also, these approaches are very sensible to triangulation irregularity. [22] tries to preserve these small flat regions but introducing too large ones, resulting in undesired artifacts. Our method generates a better result regarding the others, because we preserve sharp features and most of the flat small regions, while denoising the entire shape. The patterns generated by the curvature mapping are close to the original model than the other approaches.

In Table 7.13 we show the errors for all metrics evaluating each algorithm. Each row represents an algorithm and each column a metric. As we can see, our proposal has the minimum error in distance-based metrics. So, our resulting mesh is closer to the original than the results obtained from other algorithms.

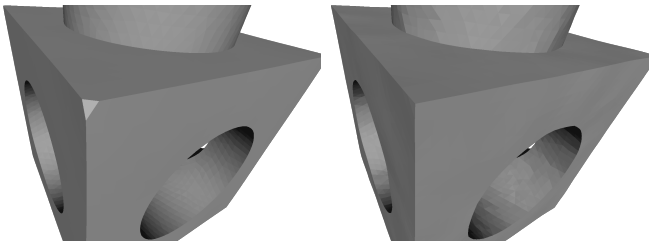


Figure 7.4: Wrong normal direction in the corner of the resulting mesh using the Guided mesh normal filtering. Left: resulting mesh of [31]. Right: our result.

In the case of normal based metrics, we have the minimum error for metrics that integrate the error over the corresponding area. MSAE is less robust than the others in this sense. Our curvature error is very similar to the lowest, and we can say that our method has a good preservation of area and volume in this case.

Table 7.13: Results for block mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
[23]	0.030743	0.052414	0.042734	9.477970	0.035246	0.043968	0.109589	0.103718	0.045399
[24]	0.002910	0.004648	0.004019	5.537650	0.011929	0.004498	0.074113	0.011371	0.009549
[28]	0.002159	0.011088	0.002844	2.172140	0.004683	0.002574	0.029590	0.003375	0.000300
[29]	0.001960	0.010192	0.002583	2.088220	0.004298	0.002258	0.029038	0.003223	0.000210
[22]	0.012640	0.020213	0.019013	6.127930	0.024389	0.021592	0.109556	0.040437	0.024038
[31]	0.001249	0.004180	0.001698	1.845830	0.002676	0.001282	0.028262	0.002161	0.000849
Ours	0.000565	0.001280	0.000743	1.906880	0.002001	0.000741	0.029958	0.000186	0.000433

7.5.2 Devil mesh

This mesh presents different kind of features with irregular triangulation. We corrupted this mesh with artificial Gaussian noise following random directions. The standard deviation σ used for the Gaussian distribution is equal to $0.15l$, where l is the average edge length of the corresponding mesh. To initialize the data for our algorithm, we performed one iteration of bilateral normal filtering and seven iterations of vertex updating to smooth high-intensity noise.

In Figure 7.6, we show the results obtained for this mesh. [23] and [24], deformed too much the original mesh. [22] introduces too many flat regions. [33] better preserve thin regions like the horns and teeth but does not totally remove the noise (see the noise near the teeth). [28] and [29] generate a smooth result near to the original, but they suffer shrinkage in the mentioned thin regions (due that these regions have large triangles). Also, these methods do not preserve too many sharp features near the eyes. [31] generates large flat regions due to the irregularity of the triangulation. We think that our method

generates a more balanced result regarding the mentioned problems of other methods.

In Table 7.14 we show the errors for all metrics evaluating each algorithm. Each row represents an algorithm and each column a metric. We can see that our result is not the best (minimum) in any metric but in most cases is close to it. Due that we opted to generate a balanced result regarding some features, the numerical results are balanced too. The errors of [29] are very low because the input shape has large smooth regions which is a characteristic of their resulting meshes.

Table 7.14: Results for devil mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
[23]	0.061277	0.110973	0.236219	19.697900	0.055170	0.047678	0.090284	0.051443	0.045645
[24]	0.001293	0.002800	0.002289	21.237300	0.021589	0.013026	0.087991	0.000364	0.002621
[28]	0.001439	0.002880	0.003540	14.043200	0.012654	0.008911	0.055849	0.007806	0.000582
[29]	0.000713	0.001537	0.001824	12.171400	0.009654	0.005781	0.054567	0.005617	0.000425
[22]	0.002531	0.004560	0.007108	13.830100	0.017459	0.010314	0.114528	0.001686	0.001786
[31]	0.001623	0.003079	0.005048	10.454200	0.015233	0.008054	0.094668	0.002629	0.001326
[33]	0.000737	0.001548	0.001493	16.880800	0.014129	0.006974	0.079952	0.000209	0.002375
Ours	0.000987	0.001902	0.002686	11.574200	0.010632	0.006796	0.075106	0.003970	0.000722

7.5.3 Fandisk mesh

This mesh presents sharp features with flat and rounded areas. We corrupted the mesh with artificial Gaussian noise following random directions. The standard deviation σ used for the Gaussian distribution is equal to $0.3l$, where l is the average edge length of the corresponding mesh. To initialize the data for our algorithm, we performed three iterations of bilateral normal filtering and seven iterations of vertex updating to smooth high-intensity noise.

In Figure 7.7, we show the results obtained for this mesh. [23] does not remove all the noise and introduces some not desired artifacts. [28], [29], [33] and [34] blurred too much the edges with low dihedral angle between involved faces. [22] better preserve these features but blurring them a little and introducing flat regions. [31] has a correct preservation of these edges when the dihedral angle is high enough. Our result is the most similar to the original mesh because we preserved better these edges without introducing not desired flat regions. If we see the curvature mapping of the original mesh, these challenging edges are in thin regions with high curvature (positive or negative), and our result generates the most similar result in this sense.

In Table 7.15 we show the errors for all metrics evaluating each algorithm. Each row represents an algorithm and each column a metric. Regarding the distance based metrics, we have not the minimum error in all of them, but

all errors are close to it. In the case of normal based metrics we have the minimum error for all of them, so, our algorithm better preserves the shape of the original mesh. Our curvature error is not the minimum because we introduce some small flat regions with curvature 0. We preserve better the area and volume than the other algorithms.

Table 7.15: Results for fandisk mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
[23]	0.000330	0.000671	0.000010	9.821170	0.041399	0.038493	0.598858	0.073705	0.016398
[24]	0.000238	0.000433	0.000007	9.984250	0.043886	0.016946	0.780315	0.007715	0.008510
[28]	0.000128	0.000364	0.000004	4.188150	0.014546	0.017077	0.138360	0.016164	0.000733
[29]	0.000081	0.000213	0.000003	3.377610	0.008664	0.011561	0.131381	0.012807	0.001007
[22]	0.000355	0.000630	0.000011	6.979130	0.028042	0.029579	0.197488	0.056925	0.017092
[31]	0.000046	0.000117	0.000002	2.627640	0.007719	0.006241	0.113402	0.007576	0.000049
[33]	0.000072	0.000191	0.000002	3.289700	0.009118	0.009494	0.122689	0.008392	0.001656
[34]	0.000066	0.000185	0.000002	2.493830	0.006311	0.004888	0.097299	0.010766	0.000840
Ours	0.000048	0.000156	0.000001	2.351650	0.005791	0.004378	0.116578	0.004723	0.000813

7.5.4 Joint mesh

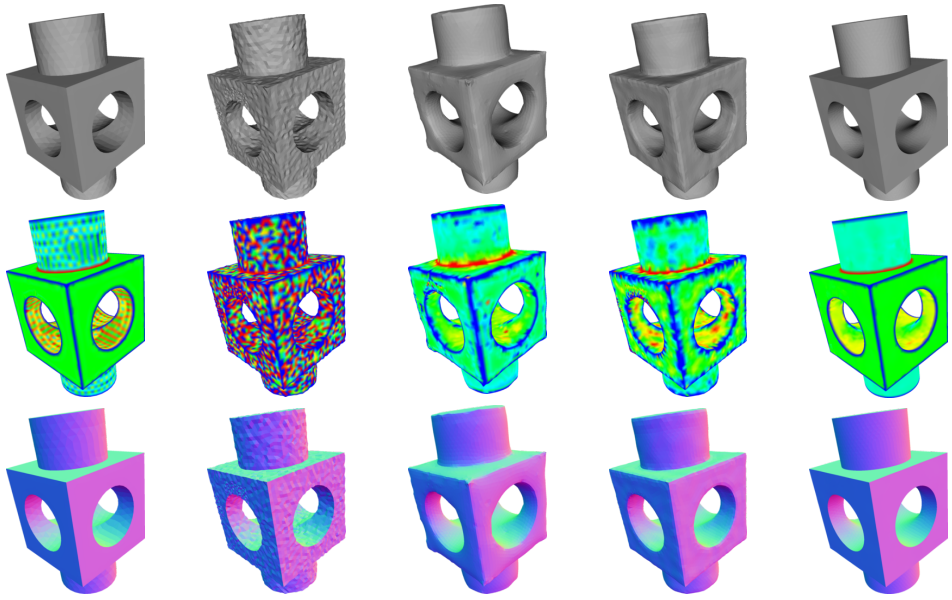
As in the case of block mesh, this one has an irregular triangulation and presents sharp features with large flat and rounded areas. We corrupted the mesh with artificial Gaussian noise following vertex normal directions. The standard deviation σ used for the Gaussian distribution is equal to $0.35l$, where l is the average edge length of the corresponding mesh. To initialize the data for our algorithm, we performed three iterations of bilateral normal filtering and seven iterations of vertex updating to smooth high-intensity noise.

In Figure 7.8, we show the results obtained for this mesh. [23] does not remove all the noise and introduces some not desired artifacts. [28], [29] and [22] introduce some artifacts in sharp regions. [31] generates flat regions in rounded areas deforming the holes of the volume represented by the shape. The results of [33] and [34] are very similar to the original mesh. Our algorithm generates a shape, preserving the sharp features and rounded areas but not sufficiently smooth to reach the original shape smoothness. Maybe performing more bilateral filtering iterations we can obtain a more similar result.

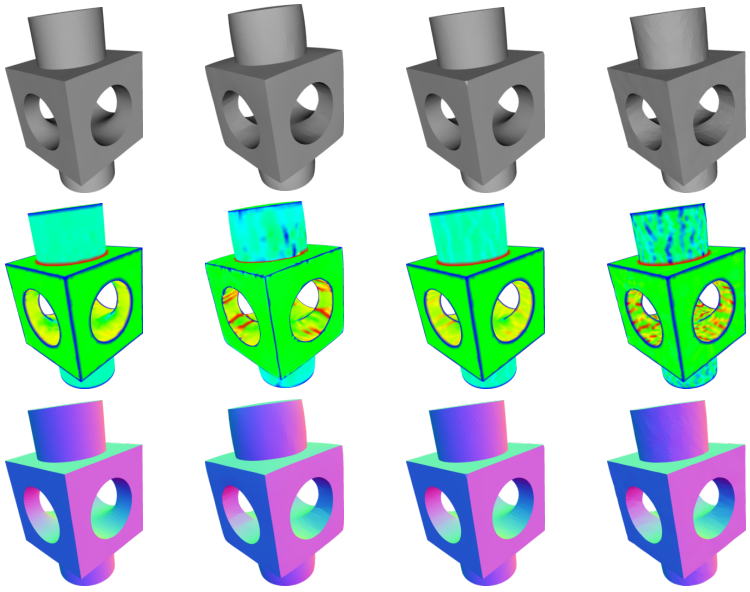
In Table 7.16 we show the errors for all metrics evaluating each algorithm. Each row represents an algorithm and each column a metric. We have better preservation of area and volume, and better results in L2 distance-based metrics. In fact, the best results are spread between our proposal, [33], and [34].

Table 7.16: Results for joint mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
[23]	5.28e-6	1.23e-5	4.28e-9	5.339180	0.016951	0.020971	0.973744	0.026317	0.003270
[24]	4.44e-6	8.12e-6	3.62e-9	6.708570	0.020659	0.011200	1.242240	0.005358	0.001581
[28]	7.06e-7	2.43e-6	6.09e-10	1.470930	0.002508	0.003977	0.227227	0.002814	0.000115
[29]	5.5e-7	1.11e-6	4.86e-10	1.250500	0.001493	0.003002	0.201129	0.000855	0.000152
[22]	4.9e-6	1.34e-5	4.14e-9	3.187840	0.005773	0.010923	0.218453	0.019501	0.002809
[31]	5.82e-7	1.48e-6	4.78e-10	1.304530	0.001645	0.003050	0.154981	0.000719	0.000215
[33]	4.84e-7	1.29e-6	4.2e-10	0.930990	0.000756	0.003247	0.113735	0.000917	0.000223
[34]	5.69e-7	1.06e-6	4.89e-10	0.986475	0.000882	0.000977	0.160960	0.003271	0.000583
Ours	5.67e-7	1.04e-6	4.67e-10	1.051820	0.000601	0.001352	0.180188	0.000203	0.000066

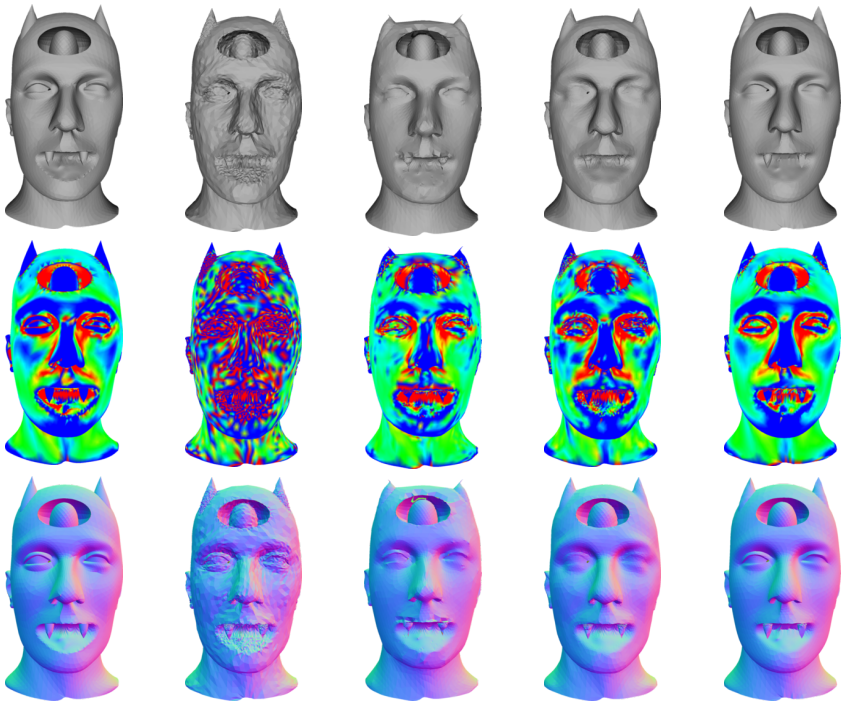


7.5(a): 7.5(b): 7.5(c): 7.5(d): 7.5(e):

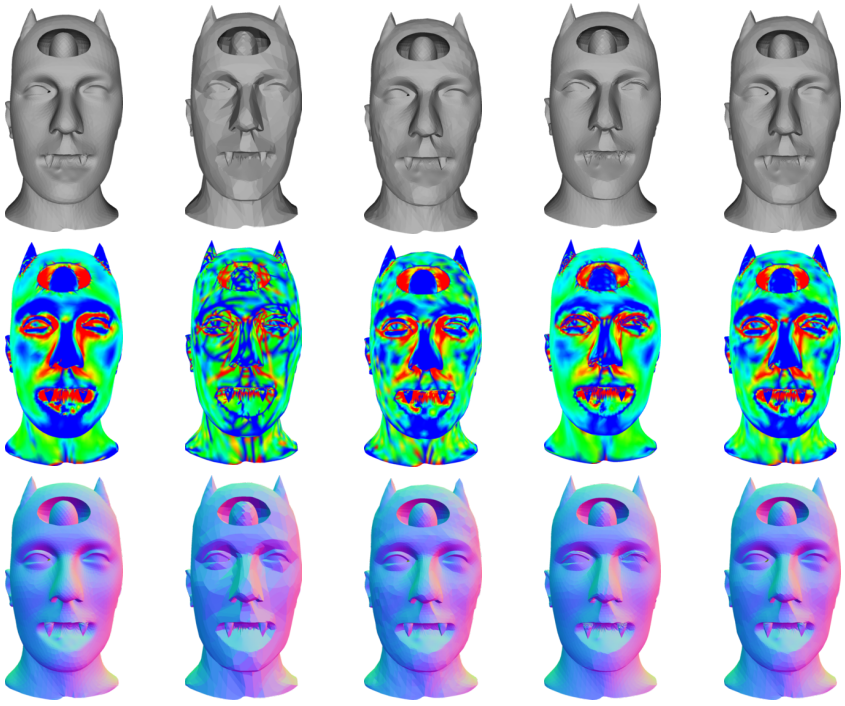


7.5(f): 7.5(g): 7.5(h): 7.5(i):

Figure 7.5: Results obtained for block model. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.5(a): Original. 7.5(b): Noisy. 7.5(c): [23]. 7.5(d): [24]. 7.5(e): [28]. 7.5(f): [29]. 7.5(g): [22]. 7.5(h): [31]. 7.5(i): Our method.



7.6(a): 7.6(b): 7.6(c): 7.6(d): 7.6(e):



7.6(f): 7.6(g): 7.6(h): 7.6(i): 7.6(j):

Figure 7.6: Results obtained for devil mesh. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.6(a): Original. 7.6(b): Noisy. 7.6(c): [23]. 7.6(d): [24]. 7.6(e): [28]. 7.6(f): [29]. 7.6(g): [22]. 7.6(h): [31]. 7.6(i): [33]. 7.6(j): Our method.

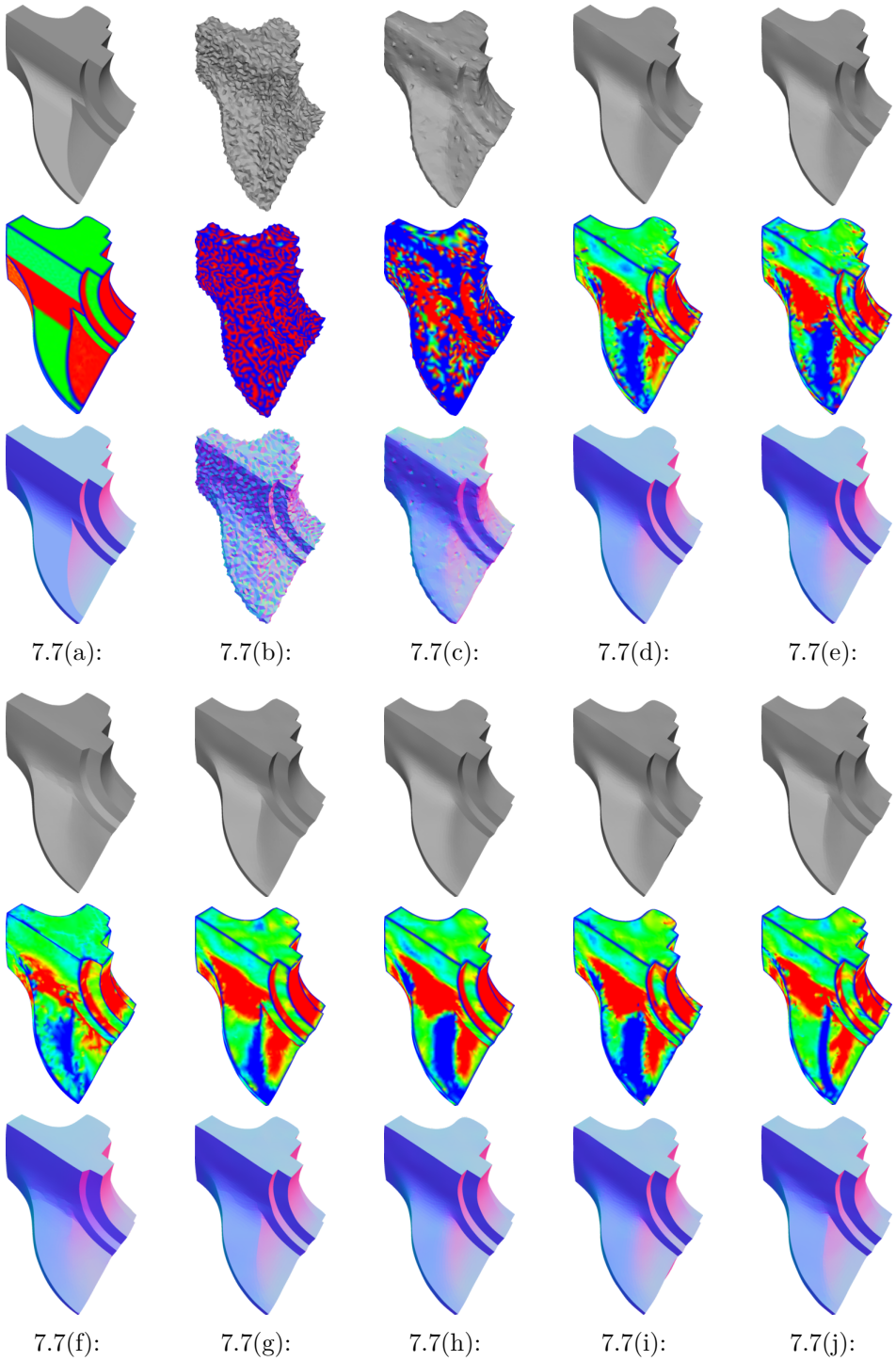
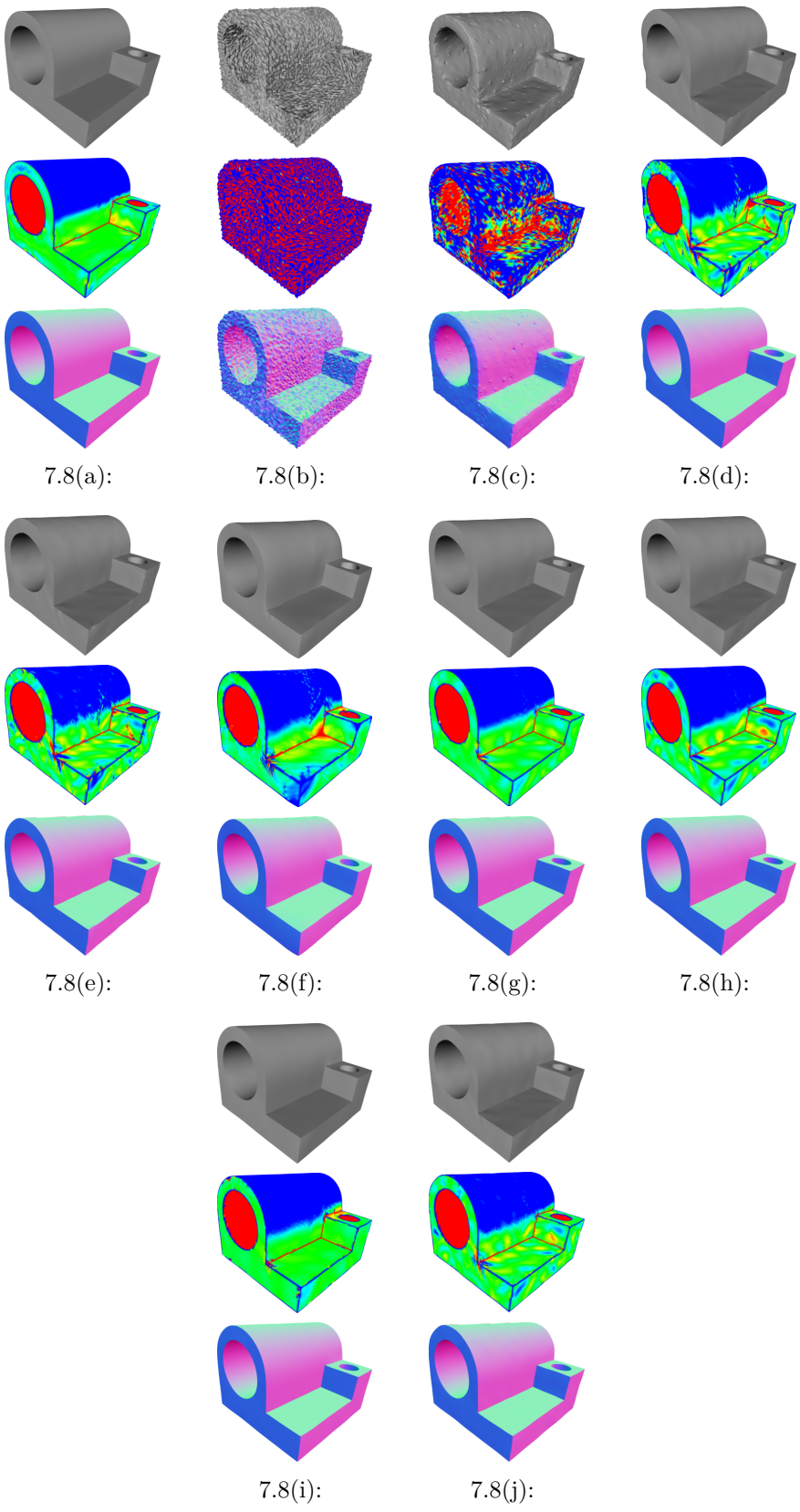


Figure 7.7: Results obtained for fandisk mesh. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.7(a): Original. 7.7(b): Noisy. 7.7(c): [23]. 7.7(d): [28]. 7.7(e): [29]. 7.7(f): [22]. 7.7(g): [31]. 7.7(i): [33]. 7.7(h): [34]. 7.7(j): Our method.



7.8(a): 7.8(b): 7.8(c): 7.8(d):

7.8(e): 7.8(f): 7.8(g): 7.8(h):

7.8(i): 7.8(j):

Figure 7.8: Results obtained for joint mesh. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.8(a): Original. 7.8(b): Noisy. 7.8(c): [23]. 7.8(d): [28]. 7.8(e): [29]. 7.8(f): [22]. 7.8(g): [31]. 7.8(i): [33]. 7.8(h): [34]. 7.8(j): Our method.

7.6 Comparison with other algorithms using real data

In this section, we will visually compare the results over real data generated from medical reconstruction and data acquired from 3D scans. We have 4 test cases, and we will try to explain the behavior of our proposal compared to the other ones.

7.6.1 Gargoyle mesh

This mesh was acquired using a 3D scanner. It has several holes and irregular triangulation (connection of vertices with a high difference between depth values in the acquisition).

In Figure 7.10, we show the results obtained for this mesh. As in the previous case, [23] does not successfully remove the noise. [22] removes the noise but removing small details too. The other approaches generate blurred results in these small details. For example, the details that are near the neck, are blurred in all of these cases (See Figure 7.9). In counterpart, our resulting mesh better preserves these details and generates thinner regions with high curvature resulting in a well-defined mesh.

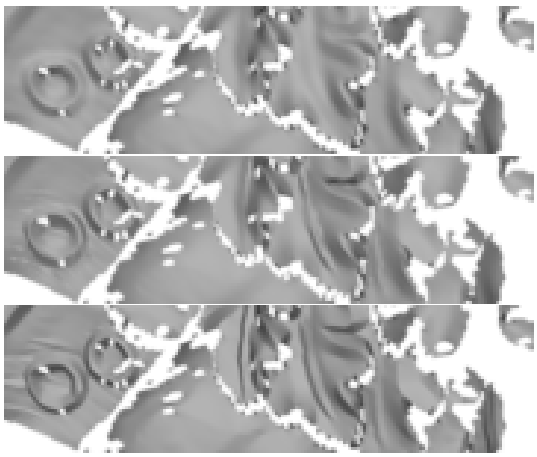


Figure 7.9: Gargoyle mesh details preserved with our denoising algorithm. First row: [29]. Second row: [33]. Third row: ours.

7.6.2 BallJoint mesh

This mesh was reconstructed from medical data and represents a bone of the human body. It has large smooth regions corrupted with noise and some sharp edges.

In Figure 7.11, we show the results obtained for this mesh. [23] and [24] do not remove the noise very well. [22] generates too many flat regions, removing the smoothness of the shape. The curvature mapping of this approach shows thin regions with high curvature. [29] and [31] smooth too much the shape, removing some sharp features. [33] does not preserve the continuity of sharp features. The results of [28] and our denoising algorithm successfully remove the noise while preserving the sharp features and their continuity. The difference is that their result is more blurred than ours which is more defined.

7.6.3 Building mesh

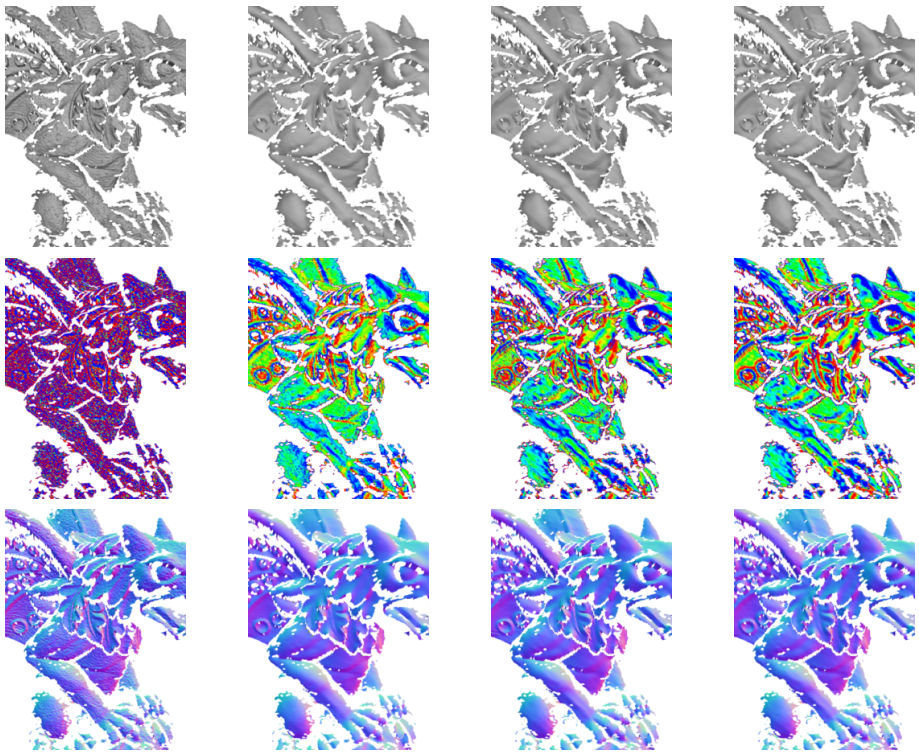
This mesh was acquired using a 3D scanner and represents the front of a building with some details.

In Figure 7.12, we show the results obtained for this mesh. [23] does not successfully remove the noise. [28], [29], and [31] blurred too much the sharp regions like the intersections between bricks. [22] better preserves these regions but removing small details. Our approach generates a well-defined mesh, preserving most of the details and removing the noise.

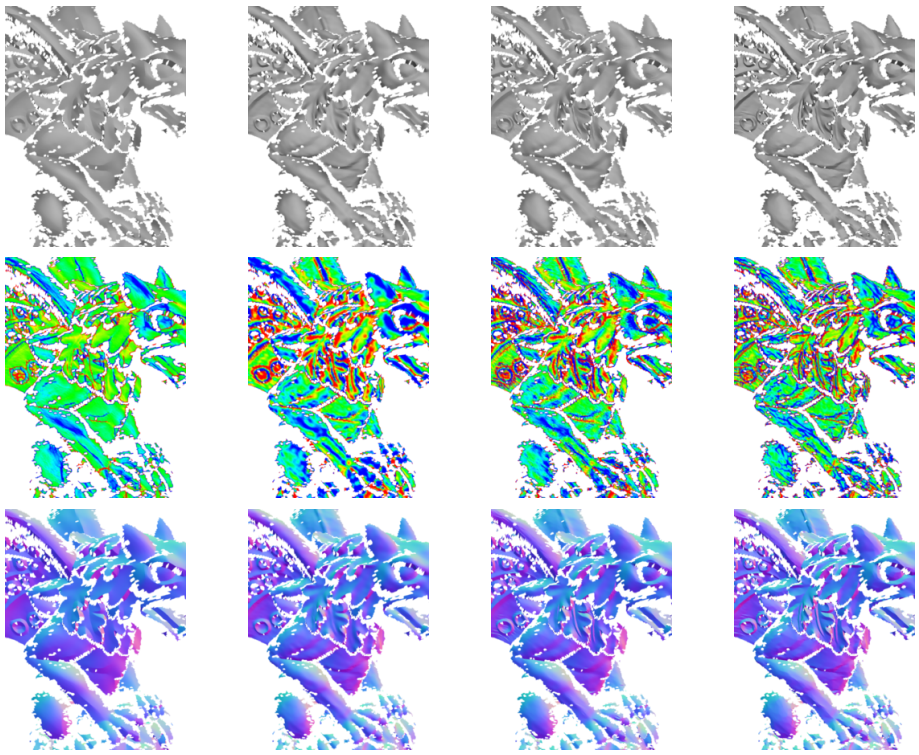
7.6.4 Keyboard mesh

This mesh was also acquired using a 3D scanner and represents a keyboard of a computer. The mesh has edges and several flat regions.

In Figure 7.13, we show the results obtained for this mesh. [23] does not remove all the noise inside flat regions. [28], [29] and [31] remove this noise but generating rounded edges. [22] and our approach generate more defined sharp features and flat areas.

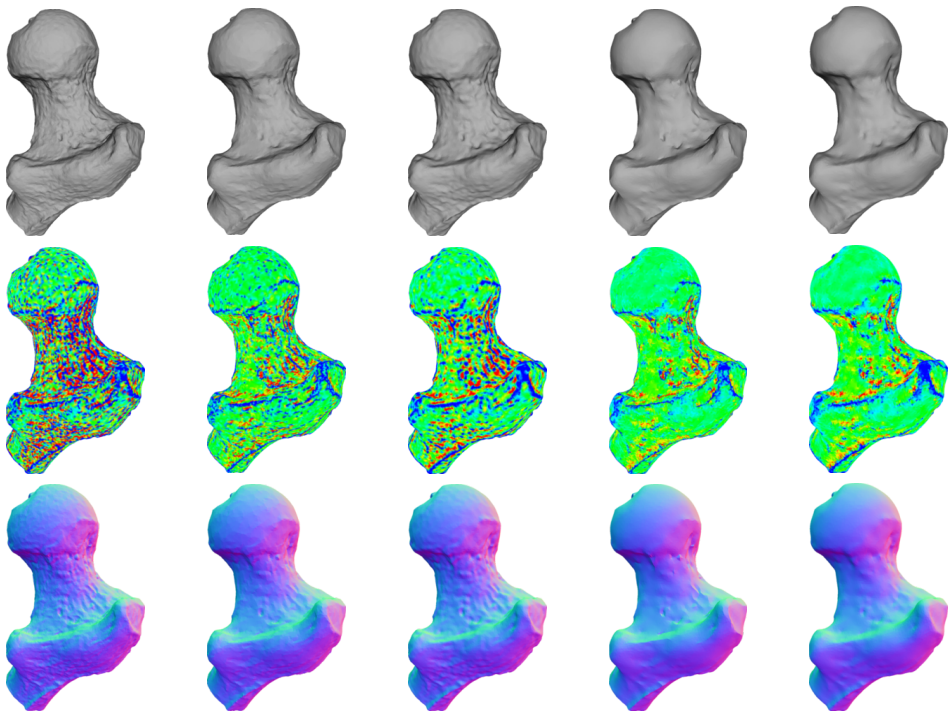


7.10(a): 7.10(b): 7.10(c): 7.10(d):

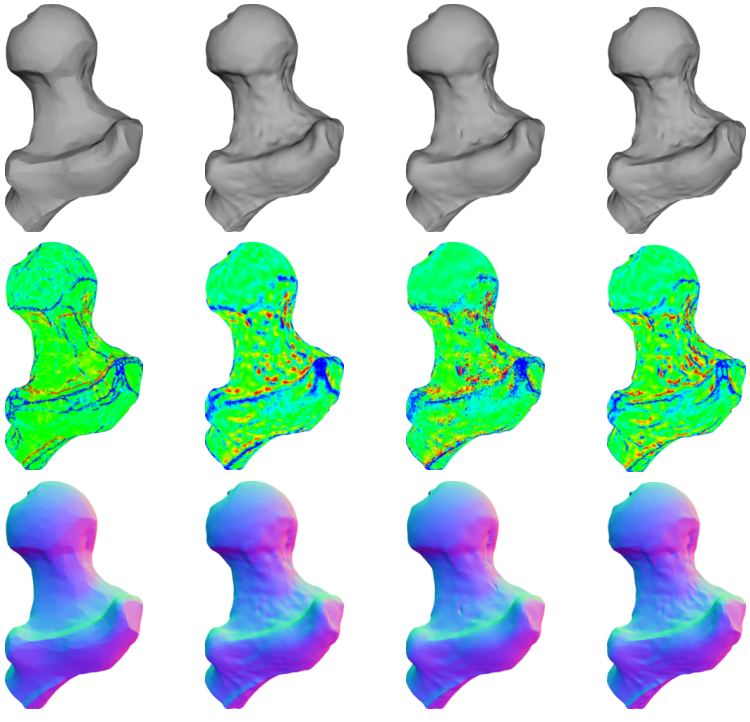


7.10(e): 7.10(f): 7.10(g): 7.10(h):

Figure 7.10: Results obtained for gargoyle model. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.10(a): Noisy. 7.10(b): [23]. 7.10(c): [28]. 7.10(d): [29]. 7.10(e): [22]. 7.10(f): [31]. 7.10(g): [33]. 7.10(h): Our method.

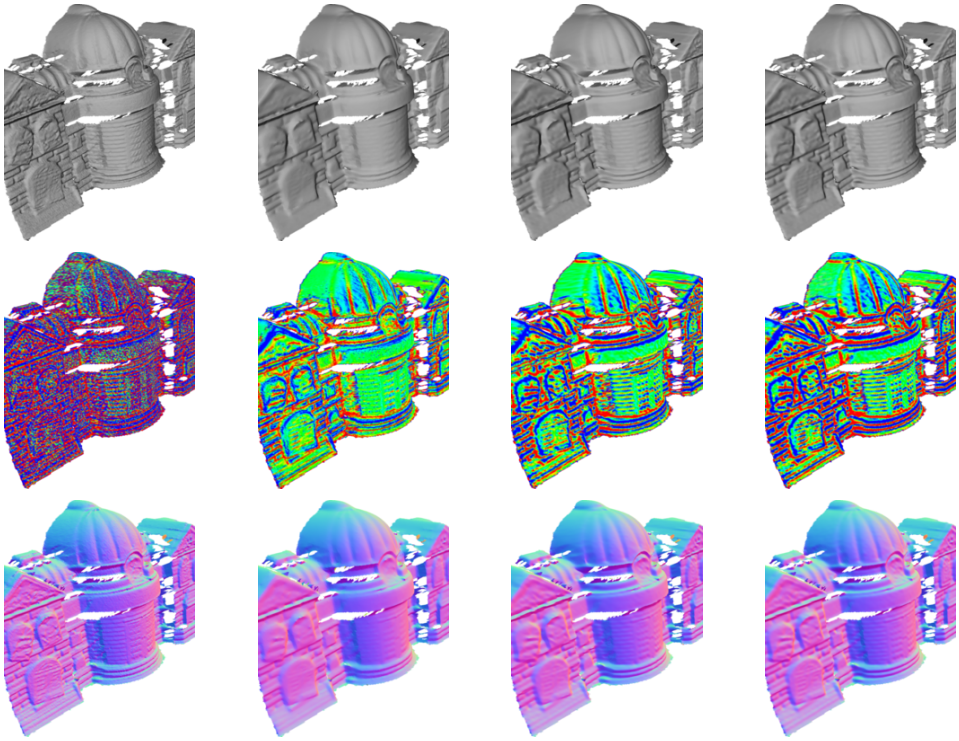


7.11(a): 7.11(b): 7.11(c): 7.11(d): 7.11(e):

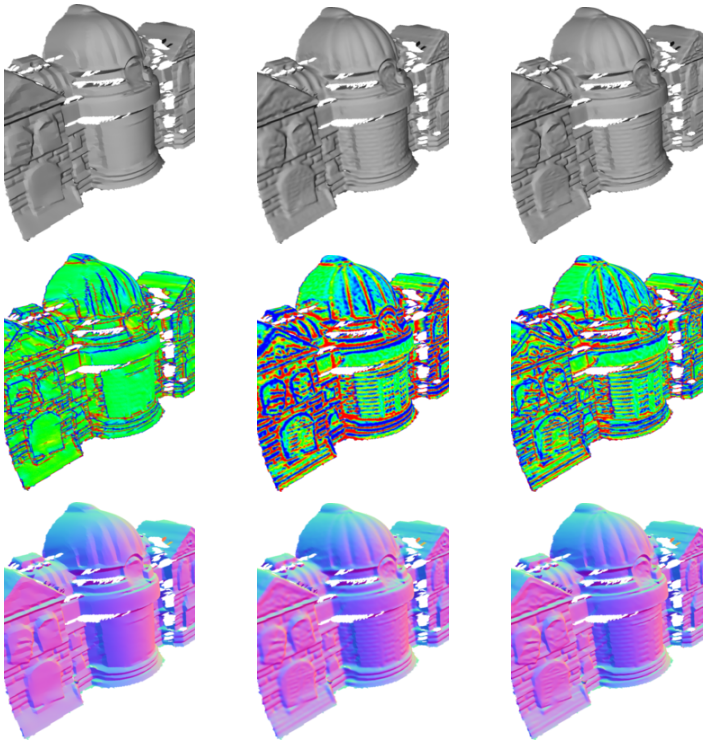


7.11(f): 7.11(g): 7.11(h): 7.11(i):

Figure 7.11: Results obtained for ball joint model. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.11(a): Noisy. 7.11(b): [23]. 7.11(c): [24]. 7.11(d): [28]. 7.11(e): [29]. 7.11(f): [22]. 7.11(g): [31]. 7.11(h): [33]. 7.11(i): Our method.

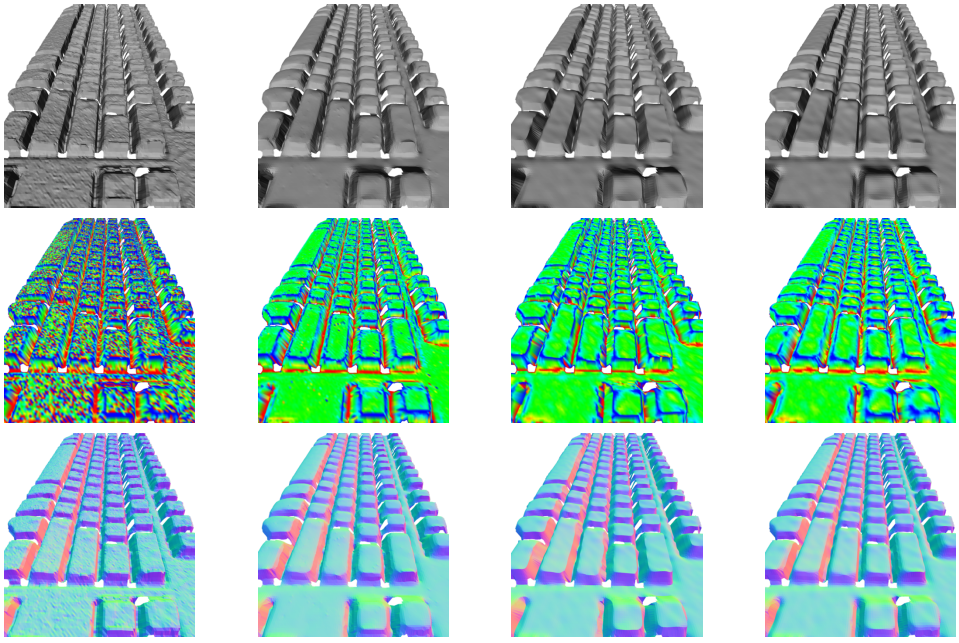


7.12(a): 7.12(b): 7.12(c): 7.12(d):

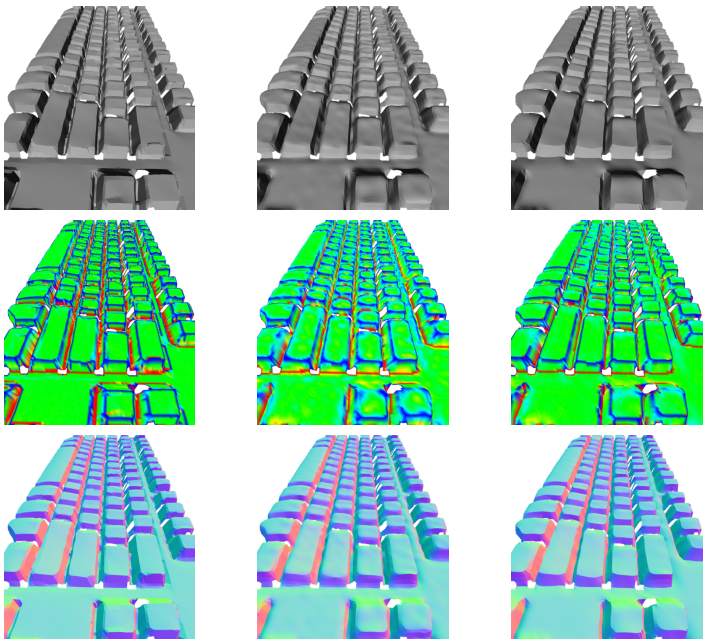


7.12(e): 7.12(f): 7.12(g):

Figure 7.12: Results obtained for building model. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.12(a): Noisy. 7.12(b): [23]. 7.12(c): [28]. 7.12(d): [29]. 7.12(e): [22]. 7.12(f): [31]. 7.12(g): Our method.



7.13(a): 7.13(b): 7.13(c): 7.13(d):



7.13(e): 7.13(f): 7.13(g):

Figure 7.13: Results obtained for keyboard model. For each sub-figure, first row shows a flat rendering, second row shows the mean curvature and third row shows the normal map. 7.13(a): Noisy. 7.13(b): [23]. 7.13(c): [28]. 7.13(d): [29]. 7.13(e): [22]. 7.13(f): [31]. 7.13(g): Our method.

7.7 Performance

To measure the performance of our implementation we did some experiments varying the size of the input and the maximum number of variables for the optimization problems.

For the first experiment, we used the dragon mesh, resampling it to 100K, 75K, 50K, 25K and 10K number of faces. Then we added Gaussian noise following vertex normal directions with an intensity of $0.1l$ (l equal to average edge length). We want to measure the execution time for all of them using the same parameters. The maximum distance allowed in the distance computation is two times the average distance between two face centroids, and the maximum number of variables allowed in the optimization problems is 20 (the parameters of the algorithm are defined in experiment specifications). In Table 7.17 we show the execution times and in Figure 7.14 its respective graph.

Table 7.17: Execution time of our denoising algorithm over meshes (decimated from dragon mesh) with different number of faces.

number of faces	10K	25K	50K	75K	100K
time (s)	16.953	48.03	99.482	139.197	181.457

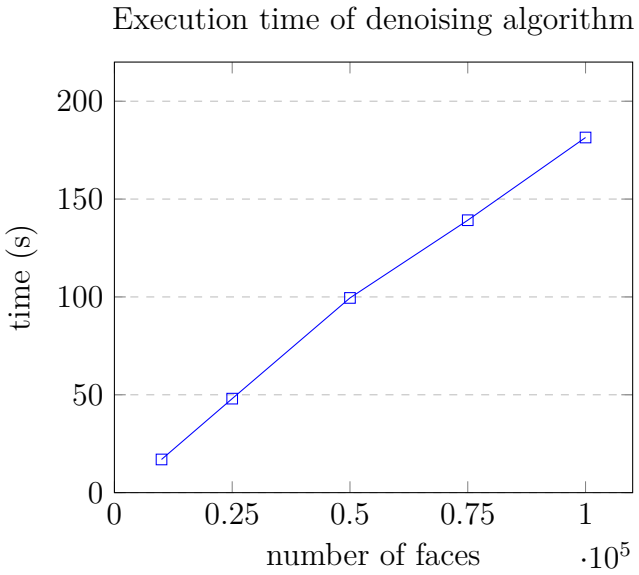


Figure 7.14: Execution time of our denoising algorithm over meshes (decimated from dragon mesh) with different number of faces.

As we can see, the function that describes the execution time regarding the number of faces, has a linear growth. This happens because the optimiza-

tion problems have the same maximum number of variables in all cases, resulting in a constant time operation. Due that the adaptive patches computation step iterates over all triangles the computational cost is $O(n)$ with n as the number of faces. The other steps of the algorithm are linear too if the number of iterations nad maximum distance are sufficiently low. It is important to say that the assumed constant time for each optimization problem can be very high depending on the size. In Figure 7.15 we show the resulting meshes for each decimated dragon mesh and in Table 7.18 the corresponding numerical results. These results were not generated tunning the parameters to obtain better results regarding the metrics because we wanted to use same values in all cases to have a better comparison.

Table 7.18: Results for decimated dragon meshes

Number of faces	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
10K	2.61e-7	4.62e-7	1.35e-11	17.88320	0.09160	0.05182	0.09356	0.03355	0.00655
25K	8.37e-8	1.57e-7	1.59e-12	13.54610	0.05850	0.02670	0.09931	0.01441	0.00403
50K	3.17e-8	6.07e-8	2.98e-13	10.55370	0.03674	0.01583	0.08550	0.00603	0.00119
75K	1.81e-8	3.48e-8	1.12e-13	9.08896	0.02825	0.01229	0.08093	0.00321	0.00065
100K	1.23e-8	2.38e-8	5.74e-14	8.24435	0.02394	0.01046	0.07852	0.00224	0.00052

In the second experiment, we opted to increment the maximum number of variables for the optimization problem. We performed the algorithm over the dragon mesh with 50000 number of faces. We also fixed the maximum allowed distance to 5 times the average distance between face centroids, to reach the maximum number of variables. The timing results are shown in Table 7.19 and its respective graph in Figure 7.16. We can see that the function describing the execution times has an exponential growth since the quadratic problem optimizations depend on the complexity of the solver. We recommend fixing the number of variables to a small value based on the topology of the mesh.

Table 7.19: Execution time of our denoising algorithm using different maximum number of optimization problem variables, over the dragon mesh with 50000 number of faces.

number of variables	20	40	60	80	100
time (s)	125.418	210.603	339.913	521.772	839.643

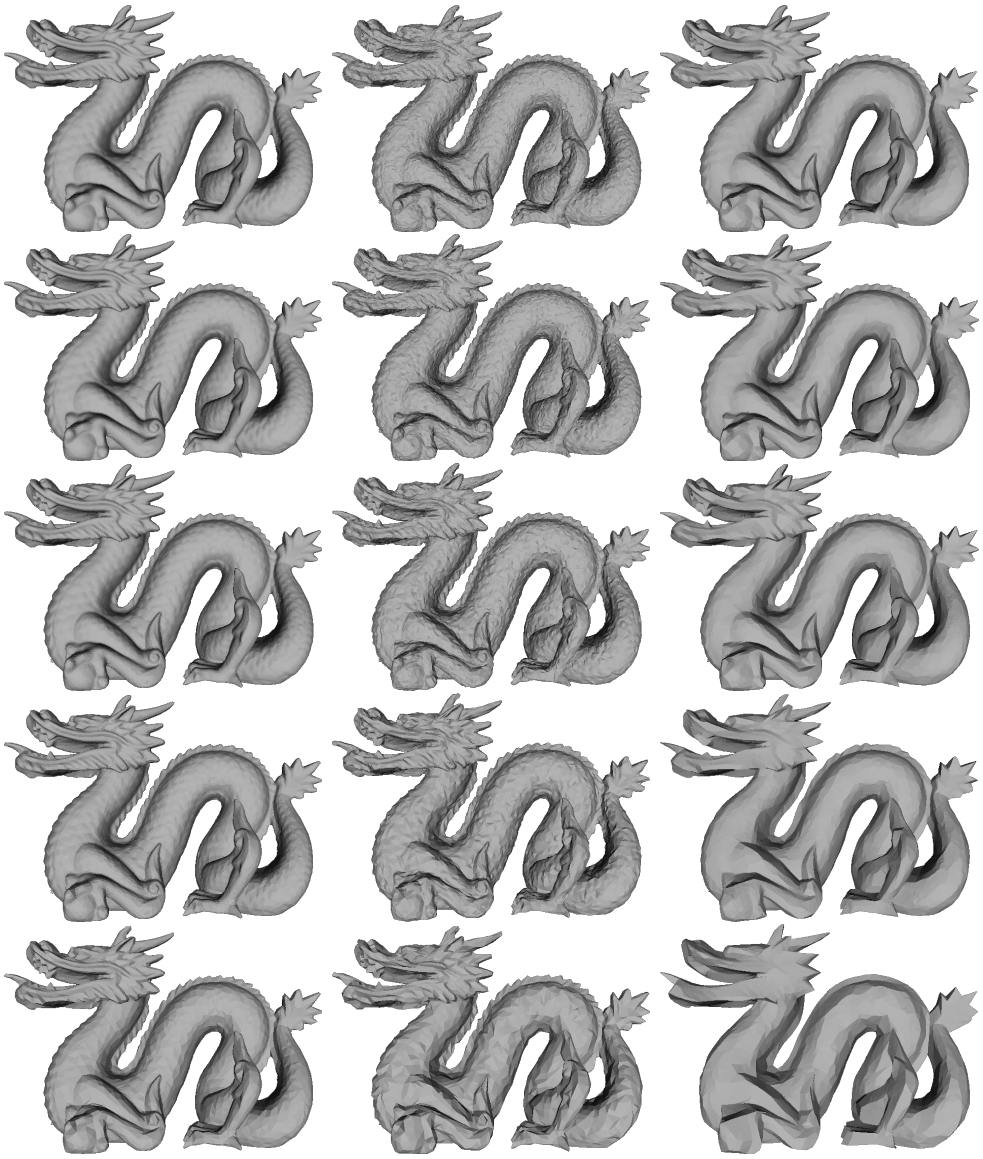


Figure 7.15: An example using our denoising algorithm. First row from left to right: original mesh and noisy mesh. Second row from left to right: resulting mesh using uniform laplacian smoothing, resulting mesh of our proposal wihout using the bilateral normal filtering step, and resulting mesh of our proposal using it.

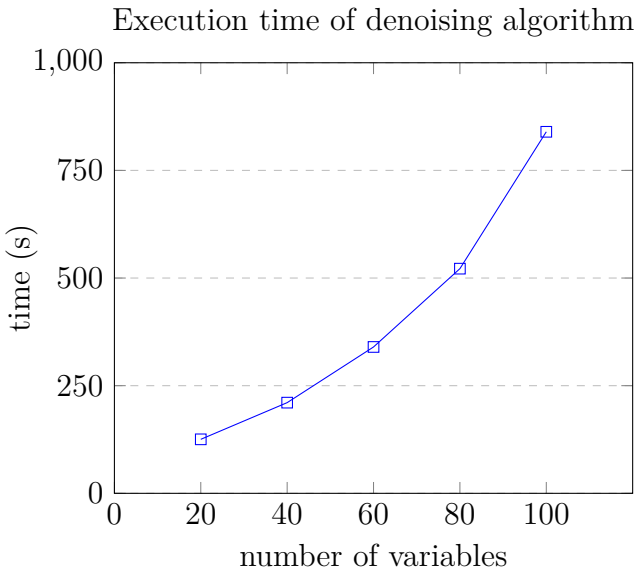


Figure 7.16: Execution time of our denoising algorithm using different maximum number of optimization problem variables, over the dragon mesh with 50000 number of faces.

7.8 Denoising meshes generated from ultrasound exams

As a use case, we perform a comparison of our denoising algorithm over two meshes generated from medical ultrasound data. In specific, the meshes were obtained from fetal ultrasound exams using the following steps: an active contour based segmentation is performed to describe the shape of the fetus and then a mesh is reconstructed using the algorithm Marching Cubes [50]. Both meshes present the staircase artifact which is a common mesh distortion when reconstructing isosurfaces of slice based volumes. We perform the filtering of these meshes in two steps. First we remove the staircase artifact using our algorithm with the following parameters: $\alpha = 1.0$, $\beta = 1.0$, $\gamma = 0.2$, $\delta = 0$, external iterations = 1, patch based iterations = 2, bilateral iterations = 0, and vertex iterations = 10. Then we remove the noise using our algorithm with the following parameters: $\alpha = 1.0$, $\beta = 1.0$, $\gamma = 0.2$, $\delta = 15$, external iterations = 2, patch based iterations = 3, bilateral iterations = 5, and vertex iterations = 10. In Figure 7.17 and Figure 7.18 we show our results compared with other methods.

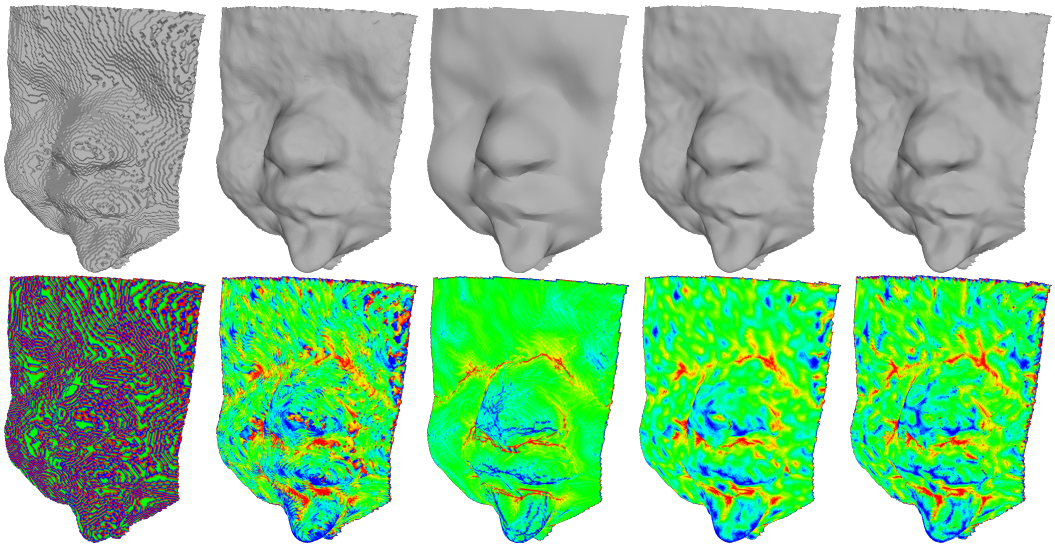


Figure 7.17: Results obtained from a mesh representing a fetus face. First row: flat rendering. Second row: curvature mapping. First column: noisy mesh with staircase artifact. Second column: [29]. Third column: [22]. Fourth column: [31]. Fifth column: Our method.

As we can see in both cases, [29] does not remove the noise generated by the staircase aritfact, [22] smooth too much the mesh, and [31] preserve better the details. Our results present better definition of details resulting in thinner and continuous regions with high or low curvature. For example, in the case of the fetus face, our algorithm preserves more details of the lips.

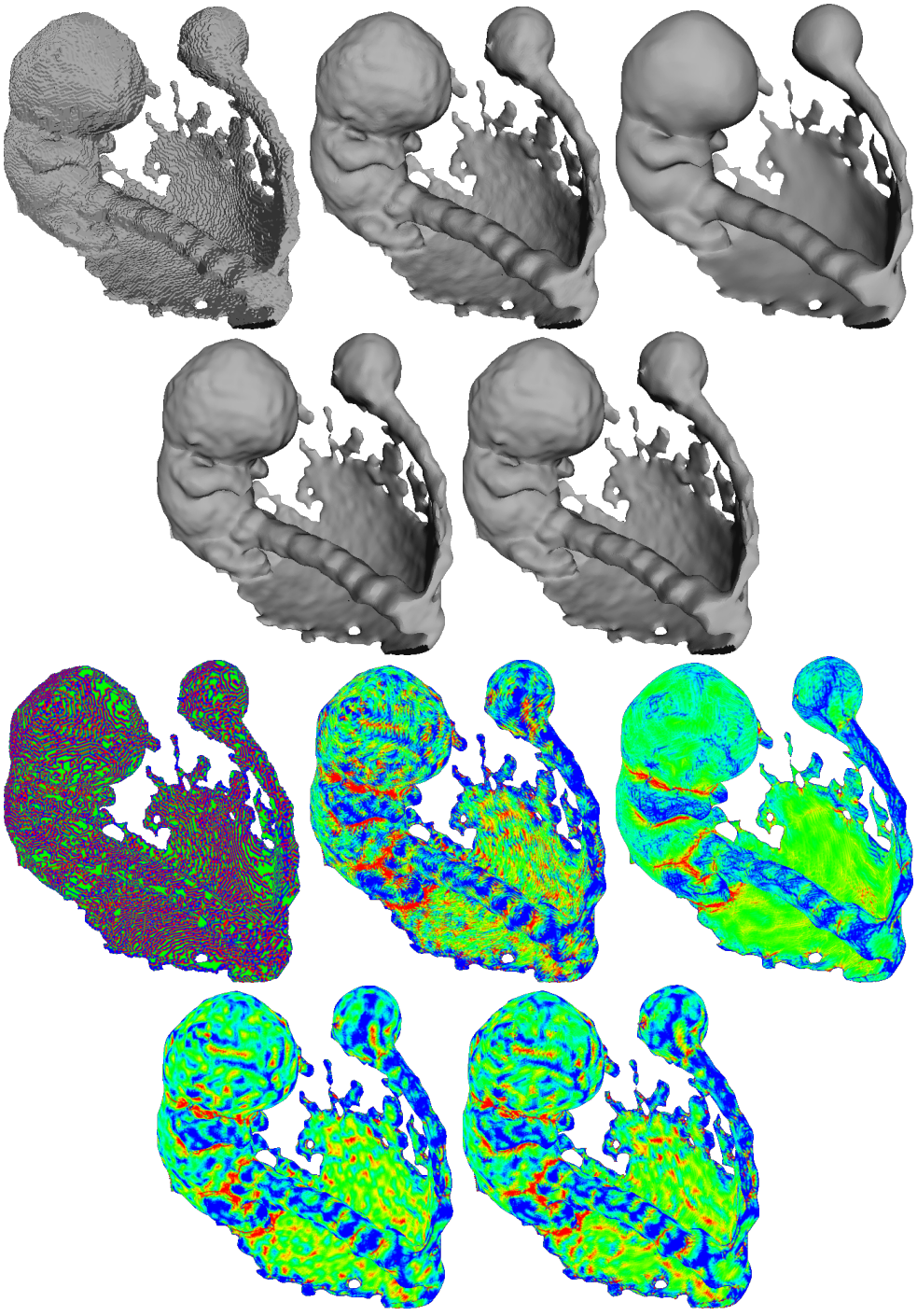


Figure 7.18: Results obtained from a mesh representing a fetus body. First group: flat rendering. Second group: curvature mapping. Starting from left to right and from top to bottom: noisy mesh with staircase artifact, [29], [22], [31] and our method.

8
Conclusion and future work

We proposed an algorithm for triangular mesh denoising with detail preservation. The core of our algorithm is a new adaptive patch computation. We compared our proposal with the current state-of-the-art denoising algorithms. Our comparison is performed visually and numerically using several metrics. Finally, we also present performance results to illustrate the complexity of our algorithm (using different sizes of input data and a different number of variables for the optimization problems).

The pipeline of our denoising algorithm is very simple and similar to other methods. The main difference lies in the new adaptive patch computation step. The result of our method is dependent on the choice of the optimization parameters, that presents some complexity but also allows great flexibility. We explain the behavior when changing the values of the parameters, allowing the reader to have a more intuitive notion of how the solution is influenced. The denoising algorithm can have multiple behaviors when tuning these parameters. For example, we can convert our filter to a near Gaussian blurring filter if we set a high value to β, or convert it to a bilateral filter if only take into account the distance to the reference point term and the normal coherence term.

The adaptive patch formulation has a quadratic and a linear term, and integrates them over the involved area. The proportion between a parameter of a quadratic term and a parameter of a linear term, strongly depends on the scale of the mesh. For this reason, we proposed to normalize the mesh before performing these operations. Also, our formulation can result in a non-convex quadratic optimization problem. For example, in the case of block mesh, we have 76% of non-convex problems due to the large flat regions that can have multiple local minima solutions. We can use a non-convex optimization solver, but it highly increment our computational time. In these cases we use the local minima as our solution yielding in good results in our denoising algorithm. The terms that we use in the formulation are helpful to ensure that the local minima is a desired solution.

Using synthetic noise, we obtained low error regarding the metrics and in most cases the lowest one (compared with the other algorithms). The

visualization of these results and the results using real data, looks better than the visualization of the others in most cases. Tuning the parameters of our algorithm, we can address different problems which depend on the nature of the data. The parameters for our algorithm used in our experiments are not the best ones; we tried to obtain them by manual tests evaluating the visualization of the result. Also, the tool that we implemented to show the behavior of the patches with dynamic interaction with the parameters, was helpful for our tests.

The computational time of our algorithm has a near-linear cost but with high constant time for the number of triangles. The limitation of the number of variables allows us to define our algorithm in this way. The quadratic optimization problems are the heaviest operations in the pipeline.

We only used the face normal field to describe the shape of the mesh. It is possible to use the vertex normal field too and generate a more robust description of the shape that allows us to compute more robust adaptive patches. This merging of normal fields was used in other denoising algorithms. Also, we can use the normal field generated by approximated tangent planes to depend less on basic element (vertex or face) description. We will experiment with these possibilities in future work.

Our approach was focused on denoising of triangular meshes, using face centroids as sampled points of the represented 2-manifold. We can use the vertex position as sampled points and perform the optimization problems on this domain instead of the face based domain. The vertex based domain was better studied in the literature and has more accurate approximations (e.g., gradient norm operator). If we work on this domain, we can extend our proposal to 3D point clouds.

We found that our adaptive patches based on normal fields can be used in other applications like mesh segmentation, remeshing or feature detection. Also, instead of using normal fields to describe the data, we can use other descriptors like curvature, heat kernels, saliency, etc. We hope to address other applications in later work.

Bibliography

[1] BOTSCH, M.; KOBBELT, L.; PAULY, M.; ALLIEZ, P. ; LEVY, B.. **Polygon Mesh Processing**. Ak Peters Series. Taylor & Francis, 2010.

[2] **Aim@shape shape repository**. `http://visionair.ge.imati.cnr.it`. Accessed: 2017-05-01.

[3] BADE, R.; HAASE, J. ; PREIM, B.. **Comparison of fundamental mesh smoothing algorithms for medical surface models**. In: IN SIMULATION UND VISUALISIERUNG (2006, p. 289–304, 2006.

[4] TERZOPOULOS, D.. **The computation of visible-surface representations**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(4):417–438, Jul 1988.

[5] WELCH, W.; WITKIN, A.. **Free-form shape design using triangulated surfaces**. In: PROCEEDINGS OF THE 21ST ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '94, p. 247–256, New York, NY, USA, 1994. ACM.

[6] KOBBELT, L.. **Discrete fairing**. In: IN PROCEEDINGS OF THE SEVENTH IMA CONFERENCE ON THE MATHEMATICS OF SURFACES, p. 101–131, 1997.

[7] ABRAMOWSKI, S.; MÜLLER, H.. **Geometrisches Modellieren**. Reihe Informatik. BI-Wiss.-Verlag, 1991.

[8] CANANN, S. A.; STEPHENSON, M. B. ; BLACKER, T.. **Optismoothing: An optimization-driven approach to mesh smoothing**. Finite Elements in Analysis and Design, 13(2):185 – 190, 1993.

[9] FREITAG, L. A.. **On combining laplacian and optimization-based mesh smoothing techniques**. ASME APPLIED MECHANICS DIVISION-PUBLICATIONS-AMD, 220:37–44, 1997.

[10] AMENTA, N.; BERN, M. ; EPPSTEIN, D.. **Optimal point placement for mesh smoothing**. Journal of Algorithms, 30(2):302–322, 1999.

[11] TAUBIN, G.. **Curve and surface smoothing without shrinkage**. In: COMPUTER VISION, 1995. PROCEEDINGS., FIFTH INTERNATIONAL CONFERENCE ON, p. 852–857. IEEE, 1995.

[12] TAUBIN, G.. **Geometric signal processing on polygonal meshes**. Eurographics State of the Art Reports, 4(3):81–96, 2000.

[13] DESBRUN, M.; MEYER, M.; SCHRÖDER, P. ; BARR, A. H.. **Implicit fairing of irregular meshes using diffusion and curvature flow**. In: PROCEEDINGS OF THE 26TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[14] VOLLMER, J.; MENCL, R. ; MUELLER, H.. **Improved laplacian smoothing of noisy surface meshes**. In: COMPUTER GRAPHICS FORUM, volumen 18, p. 131–138. Wiley Online Library, 1999.

[15] WEI, M.. **Feature-preserving Surface Mesh Smoothing, Denoising and Applications in Biomedical Modeling**. PhD thesis, THE CHINESE UNIVERSITY OF HONG KONG (HONG KONG), 2014.

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

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

[24] 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.

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

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] LI, T.; WANG, J.; LIU, H. ; LIU, L.-G.. **Efficient mesh denoising via robust normal filtering and alternate vertex updating**. 2016.

[33] YADAV, S.; REITEBUCH, U. ; POLTHIER, K.. **Mesh denoising based on normal voting tensor and binary optimization**. arXiv preprint arXiv:1607.07427, 2016.

[34] YADAV, S. K.; REITEBUCH, U. ; POLTHIER, K.. **Robust and high fidelity mesh denoising**. CoRR, abs/1711.05341, 2017.

[35] 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.

[36] FOI, A.; KATKOVNIK, V. ; EGIAZARIAN, K.. **Signal-dependent noise removal in pointwise shape-adaptive dct domain with locally adaptive variance**. In: SIGNAL PROCESSING CONFERENCE, 2007 15TH EUROPEAN, p. 2159–2163. IEEE, 2007.

[37] CIGNONI, P.; ROCCHINI, C. ; SCOPIGNO, R.. **Metro: Measuring error on simplified surfaces**. Technical report, Paris, France, France, 1996.

[38] NEHORAI, A.; HAWKES, M.. **Performance bounds for estimating vector systems**. IEEE Transactions on Signal Processing, 48(6):1737–1749, Jun 2000.

[39] KIM, S.-J.; KIM, S.-K. ; KIM, C.-H.. **Discrete differential error metric for surface simplification**. In: 10TH PACIFIC CONFERENCE ON COMPUTER GRAPHICS AND APPLICATIONS, 2002. PROCEEDINGS., p. 276–283, 2002.

[40] YAGOU, H.; OHTAKE, Y. ; BELYAEV, A.. **Mesh smoothing via mean and median filtering applied to face normals**. In: GEOMETRIC MODELING AND PROCESSING. THEORY AND APPLICATIONS. GMP 2002. PROCEEDINGS, p. 124–131, 2002.

[41] BELYAEV, A.; OHTAKE, Y.. **A comparison of mesh smoothing methods**. In: ISRAEL-KOREA BI-NATIONAL CONFERENCE ON GEOMETRIC MODELING AND COMPUTER GRAPHICS, volumen 2. Citeseer, 2003.

[42] ALFACE, P. R.; DE CRAENE, M. ; MACQ, B. B.. **Three-dimensional image quality measurement for the benchmarking of 3d watermarking schemes**. In: ELECTRONIC IMAGING 2005, p. 230–240. International Society for Optics and Photonics, 2005.

[43] GARLAND, M.; HECKBERT, P. S.. **Surface simplification using quadric error metrics**. In: PROCEEDINGS OF THE 24TH ANNUAL

CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '97, p. 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[44] MEYER, M.; DESBRUN, M.; SCHRÖDER, P. ; BARR, A. H.. **Discrete Differential-Geometry Operators for Triangulated 2-Manifolds**, p. 35–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[45] ZHANG, C.; CHEN, T.. **Efficient feature extraction for 2d/3d objects in mesh representation**. In: PROCEEDINGS 2001 INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (CAT. NO.01CH37205), volumen 3, p. 935–938 vol.3, 2001.

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

[47] BLANCO, J. L.. **nanoflann: a c++ header-only fork of flann, a library for nearest neighbor (nn) wih kd-trees**, 2014.

[48] ASPERT, N.; SANTA-CRUZ, D. ; EBRAHIMI, T.. **Mesh: measuring errors between surfaces using the hausdorff distance**. In: PROCEEDINGS. IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, volumen 1, p. 705–708 vol.1, 2002.

[49] GODIL, A.; DUTAGACI, H.; BUSTOS, B.; CHOI, S.; DONG, S.; FURUYA, T.; LI, H.; LINK, N.; MORIYAMA, A.; MERUANE, R. ; OTHERS. **Shrec'15: Range scans based 3d shape retrieval**. 2015.

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