Aix-Marseille University
École Doctorale en Mathématiques et
Informatique de Marseille E.D 184

Faculté des Sciences
CNRS, Laboratory LSIS, UMR 7296
163, Avenue de Luminy 13288 Marseille cedex 9, France

PhD THESIS
in Computer Science

3D Modeling of elevation surfaces from voxel structured point clouds extracted from seismic cubes

Presented by
NGUYEN Van Sinh

To obtain the Degree of Doctor of Aix-Marseille University

Defended on October 25, 2013 in front of the committee composed of:

Samir Akkouche	Lyon 1 University	Professor, Reviewer
Gudrun Albrecht	Valenciennes University	Professor, Committee Member
Alexandra Bac	Aix-Marseille University	Assoc.Professor, Co-Supervisor
Luc Biard	Grenoble 1 University	Assoc.Professor, HDR, Committee Member
Marc Daniel	Aix-Marseille University	Professor, Supervisor
Marc Neveu	Dijon University	Professor, Reviewer

Number assigned by the library

Acknowledgements

It is said that: “No guide, No realization”. Indeed, that is entirely true on the path of my scientific research. During the time of my PhD research at Aix-Marseille University, I was having a lucky opportunity to work with Professor Marc DANIEL and Associate Professor Alexandra BAC, my thesis advisors. First of all, I would like to express my utmost gratitude to my supervisor Prof. Marc Daniel and co-supervisor Assoc. Prof. Alexandra Bac, for their continuous support and enthusiastic guidance over the pass more than three years. Their assistance throughout my years of study here has been invaluable. They have made their great support available from the critical thinking to the analysis of a scientific paper. They were willingly subjected to several early, very rough, versions of the papers, this dissertation, and their many comments and suggestions have undoubtedly improve things. I really learned from my advisors the way to study in science.

I would also like to thank and extend my appreciation to members of my reviewers and thesis examination committees: Prof. Gudrun ALBRECHT, University Valenciennes; Assoc.Prof.HDR Luc BIARD, University Grenoble 1; Prof. Marc NEVEU, University Dijon; and Prof. Samir AKKOUCHE, University Lyon 1 for their serious evaluations, valuable contributions and constructive suggestions.

This is also a great chance to show my appreciation and gratitude to all members, PhD students of G-Mod research group in CRNS Laboratory LSIS 7296, who have supported me during my study at Aix-Marseille University. I also thank Dr. TRAN Nam-Van and Dr. Philippe Verney for their clear explanation of the previous work that they have done; without their help, it is difficult for me to have finished this thesis.

The important issue to do this thesis is the finance. I am very honored and thankful to have received a full scholarship which supported by a cooperation between Vietnam’s Government (MOET) and France’s Government (Campus France) throughout the past few years. I was incredibly grateful for my scholarships; without them, I would not be able to go to France and study abroad. Besides, I have also supported the time by the International University, Vietnam National University of Ho Chi Minh City. I would like to thank all

their valuable helps.

Finally, I deeply appreciate the encouragement and support from my family in Vietnam. They always encourage, interest and motivate me in my work; they are also proud of me and my work. Especially, my wife DO Thanh and my lovely son Bao-Khang always give me limitless love and spiritual encouragement. She has been a source of constant support throughout the many long days of mine spent working on this dissertation. For this, and many other things, I am profoundly grateful.

NGUYEN Van Sinh

Résumé

L'Infographie est un domaine important de l'informatique largement utilisé dans le monde réel. L'arrivée des cartes graphiques grand public rapides et peu coûteuses a conduit à un important besoin de programmation des différentes tâches géométriques pour les applications, y compris les modèles informatiques, la visualisation scientifique, l'analyse d'images médicales, la simulation et les mondes virtuels. Les types d'applications évoluent pour profiter des avancées en modélisation géométrique (basée sur des modèles mathématiques). Reconstruire des surfaces avec des données provenant d'une technique d'acquisition automatique entraîne toujours le problème de la masse des données acquises. Cela implique que les procédés habituels ne peuvent pas être appliqués directement. Par conséquent, un processus de réduction des données est indispensable. Un algorithme efficace pour un traitement rapide préservant le modèle original est un outil précieux pour la construction d'une surface optimale et la gestion des données complexes.

Dans cette thèse, nous présentons des méthodes pour construire une surface géologique optimale à partir d'une quantité énorme de points 3D extraits de cubes sismiques. Appliquer le processus à l'ensemble des points induit un risque important de contraction de la surface de sorte que l'extraction de la frontière initiale est une étape importante permettant une simplification à l'intérieur de la surface. La forme globale de la surface sera alors mieux respectée pour la reconstruction de la surface triangulaire finale. Nos propositions sont basées sur la régularité des données qui permet, même si des données sont manquantes, d'obtenir facilement les informations de voisinage.

Tout d'abord, nous présentons une nouvelle méthode pour extraire et simplifier la frontière d'une surface d'élévation définie par un ensemble de voxels dans un grand volume 3D où des données sont manquantes. Deuxièmement, une méthode pour simplifier la surface à l'intérieur de sa frontière est présentée. Elle comprend une étape de simplification grossière optionnelle suivie par une étape plus fine basée sur l'étude des courbures. Nous tenons également compte du fait que la densité de données doit changer graduellement afin de recevoir à la dernière étape d'une surface triangulée avec de meilleurs triangles. Troisièmement, nous avons proposé une nouvelle méthode rapide pour trianguler la surface après simplification.

Mots-clés: Nuages de Points, Extraction de la Frontière, Simplification de la Frontière, Simplification de surfaces, Triangulation.

Abstract

Computer graphics is an important field of computer science widely used in the real world. The advent of fast and inexpensive consumer graphics hardware has led to an essential demand for knowledge of how to program various geometric tasks for applications including computational models, scientific visualization, medical image analysis, simulation, and virtual worlds. The types of applications are themselves evolving to take advantage of the advance in geometric modeling (based on mathematical models).

Reconstructing surfaces with data coming from an automatic acquisition technique always entails the problem of mass of data. This implies that the usual processes cannot be applied directly. Therefore, it leads to a mandatory data reduction process. An effective algorithm for a rapid processing while keeping the original model is a valuable tool for constructing an optimal surface and managing the complex data.

In this dissertation, we present methods for building an optimal geological surface from a huge amount of 3D points extracted from seismic cubes. Applying the process to the whole set of points induces an important risk of surface shrinking so that the initial boundary extraction is an important step permitting a simplification inside the surface. The global surface shape will then be better kept for the reconstruction of the final triangular surface. Our proposals are based on the regularity of data which permits, even if data are missing, to easily obtain the neighboring information.

Firstly, we present a new method to extract and simplify the boundary of an elevation surface given as voxels in a large 3D volume having the characteristics to be sparse. Secondly, a method for simplifying the surface inside its boundary is presented with a rough optional simplification step followed by a finer one based on curvatures. We also keep into consideration that the density of data must gradually change in order to receive in the last step a triangulated surface with better triangles. Thirdly, we have proposed a new and fast method for triangulating the surface after simplification.

Keywords: Points Clouds, Boundary Extraction, Boundary Simplification, Surface Simplification, Triangulation.

Contents

Acknowledgements i
Résumé iii
Abstract iv
Table of Contents v
List of Figures viii
List of Tables xiii
List of Algorithms xiv
Nomenclature xv

1 Introduction **1**
1.1 Background and motivation 1
1.2 Problem statement 2
1.3 Objects and scopes 3
1.3.1 Methodology 3
1.3.2 Geological data model 3
1.4 Thesis structure 5

2 Previous Works on 3D Elevation Surface Modeling **7**
2.1 Tran Nam-Van's work 8
2.1.1 Proposed methods 8
2.1.2 Obtained results and existed problems 8
2.2 Philippe Verney's work 10
2.2.1 Images processing 10
2.2.2 Structure of a sparse matrix 11
2.2.3 From a sparse matrix to a 3D volume 13
2.2.3.1 Data analysis 14
2.2.3.2 Proposed algorithm 14
2.2.3.3 Results 15
2.3 Conclusion 16

3 Boundary Extraction and Simplification **17**
3.1 Introduction 17
3.2 Related work 18
3.2.1 Boundary detection 18
3.2.2 Boundary simplification 21

3.3 Methods for boundary extraction . . . 22
3.3.1 Overview . . . 22
3.3.2 Definitions . . . 23
3.3.2.1 Definition of a square neighborhood . . . 23
3.3.2.2 Definition of different types of boundary . . . 24
3.3.3 Algorithms . . . 25
3.3.3.1 Extracting boundary by distance growing . . . 26
3.3.3.2 Extracting boundary by clockwise scanning . . . 28
3.3.4 Implementation . . . 29
3.4 Methods for boundary simplification . . . 30
3.4.1 Background and Definitions . . . 30
3.4.1.1 Line segment extraction . . . 30
3.4.1.2 3D polyline simplification . . . 31
3.4.2 Algorithm . . . 31
3.5 Results . . . 32
3.5.1 Boundary Extraction . . . 32
3.5.2 Boundary simplification . . . 32
3.6 Discussion and evaluation . . . 33
3.7 Conclusion . . . 38

4 Surface Simplification **39**
4.1 Introduction . . . 39
4.2 Related work . . . 40
4.2.1 Simplification of triangular meshes . . . 40
4.2.2 Simplification of point clouds . . . 41
4.3 Method for simplifying the inside of a surface . . . 44
4.3.1 Rough simplification . . . 44
4.3.1.1 Overview . . . 44
4.3.1.2 Algorithm . . . 45
4.3.2 Elaborate simplification . . . 45
4.3.2.1 Overview . . . 45
4.3.2.2 Analysis . . . 46
4.3.2.3 Subdivision according to the boundary density . . . 46
4.3.2.3.1 Notation and formula construction . . . 47
4.3.2.3.2 Boundary density criteria . . . 47
4.3.2.4 Subdivision according to the curvature . . . 48
4.3.2.4.1 PCA flatness criteria . . . 49
4.3.2.5 Algorithms . . . 50
4.4 Results . . . 52
4.5 Discussion and evaluation . . . 62

4.6 Conclusion . . . 62

5 Surface Triangulation **63**
5.1 Introduction . . . 63
5.2 Related work . . . 65
5.2.1 The methods in 2D . . . 65
5.2.2 The methods in 3D . . . 67
5.3 Methods for triangulating the surface . . . 70
5.3.1 Overview . . . 70
5.3.2 Concepts, notation and definition . . . 71
5.3.3 Building a seed triangle . . . 73
5.3.4 Searching conditions in one side of an edge . . . 75
5.3.5 Delaunay Criterion . . . 75
5.3.6 Neighboring points search based on the voxel tracing . . . 77
5.3.6.1 Computing the discrete mediatrix L . . . 78
5.3.6.2 Computing the successive dilatations L' . . . 79
5.3.6.3 Algorithm based on the voxel traversal search . . . 80
5.3.7 Triangulating a surface . . . 82
5.3.8 Processing the outside triangles on the boundary . . . 83
5.4 Implement . . . 84
5.5 Results . . . 86
5.6 Discussion and evaluation . . . 89
5.7 Conclusion . . . 90

6 Conclusion and Future Work **93**
6.1 Conclusion and contributions . . . 93
6.1.1 Boundary extraction . . . 94
6.1.2 Boundary simplification . . . 95
6.1.3 Rough simplification . . . 95
6.1.4 Elaborate simplification . . . 96
6.1.5 Surface triangulation . . . 97
6.2 Future work . . . 97

Bibliography **97**

Appendix **105**
Résumé . . . 105

List of Figures

1.1 3-D oblique view of a portion of the Black Warrior Basin (data from [Jr.06]). Thin vertical lines are wells; semi-transparent surfaces outlined in black are faults . . . 3
1.2 A geological model used in the oil exploration [Mas10] . . . 4
1.3 Collection of geological data: a) by a marine seismic acquisition; b) by processing seismic images . . . 4
1.4 General model of building an optimal geological triangulated surface . . . 5

2.1 A geological surface simplified with method [BTD07]: a) initial model, 112kb vertices; b) output model, 3kb vertices . . . 9
2.2 Before and after filling the holes of a geological surface [BTD08] . . . 9
2.3 Before and after processing the fault of a geological surface [Tra08] . . . 9
2.4 Extracting the information from geological images. [Ver09] . . . 10
2.5 Filtering the information of seismic images. [Ver09] . . . 11
2.6 Thinning of the reflections. [Ver09] . . . 12
2.7 Determining the surface and computing its thickness. [Ver09] . . . 12
2.8 Sparse Matrix. [Ver09] . . . 13
2.9 3D points are extracted from the sparse matrix and insert into a 3D volume. 14
2.10 An elevation surface (346796 points) defined by a sparse 3D volume. . . . 15
2.11 (a) An elevation surface of 257629 points; (b) An elevation surface of 886639 points. . . . 16

3.1 A method to extract a boundary in [SS07] (e.g. $\angle XpA$: a smallest angle) . 19
3.2 Method to extract a boundary of Shen Wei [Wei08] . . . 19
3.3 Method to extract a boundary of Xianfeng [XXFJ08] . . . 20
3.4 Method to extract the boundary points of Kalogerakis [KNSS09] . . . 21
3.5 Method to extract a boundary of Sait [SKM11] . . . 21
3.6 Method to simplify the boundary of Douglas [DP73] . . . 22
3.7 Method to simplify the boundary of Garrido and Meijers [GBGS98, Mei11] 22
3.8 The k_ring neighborhood with $k = 3$. . . . 23
3.9 Topological boundaries of 8-connectivity . . . 24
3.10 An exterior boundary with $k = 1$ . . . 25
3.11 2 cases to extract the exterior boundary . . . 26
3.12 The growth algorithm based on distance. . . . 27
3.13 The results of two cases with $k = 2$ . . . 28

3.14 The growth algorithm based on clock-wise. 29
3.15 Compute the cartesian coordinates of q. 29
3.16 Simplify the 3D boundary 31
3.17 Comparison of computing strategies between the methods: Sampath (a), ShenWei (b) and Our method (c). 33
3.18 The processing time of boundary extraction usually depends on the parameter k, although sometimes it may be affected by the shape of the surface. 34
3.19 Rate of boundary simplification with different values of threshold t. 34
3.20 Extracting an exterior boundary of the surface with $k = 1$ and $k = 3$. The shape of the boundary does not change on the convex parts of the surface but created a hole. 35
3.21 The resolution of the exterior boundary of a geological surface is highest with $k = 1$. 35
3.22 Extract an exterior boundary of the surface with different values of k. The shape of the boundary has changed on the concave parts of the surface. . . 35
3.23 Exterior boundary in the case of a geological surface: (a) the original 15626 surface points and the extracted boundary 1025 points with $k = 1$; processing time: 18 ms. (b) Simplified boundary with threshold $t < 0.4$; number of boundary points: 660; rate of reduction: 35%; the red color points are boundary points; the white color points (on the boundary) are simplified . 36
3.24 Before (a) and after (b) simplification; the boundary points reduce from 118 to 24; rate: 80%, while preserving the initial shape of this boundary (i.e. the characteristic points are preserved). 36
3.25 An exterior boundary of a geological surface after extracting with many values of k (if $k = 1$, the resolution of the boundary is high (highest); otherwise, the resolution of the boundary is low and some small holes H are created). 37
3.26 An exterior boundary of a geological surface after simplifying with many values of t: a) the original boundary (boundary points: 1024); b) after simplifying with $t = 0.3$ (boundary points: 381, simplification rate: 63%); c) after simplifying with $t = 0.5$ (boundary points: 106, simplification rate: 90%). 37
3.27 The shape of the boundary has been changed on the concave parts after extracting the boundary using many values of k. The resolution of the boundary is inversely proportional to the value of k: the smaller the value of k, the higher the resolution of the boundary; and in contrast, the more increasing values of k, the more rough boundaries we obtain. 38

4.1 Simplification of a triangular mesh by edge-collapse [Gar99] 41
4.2 Non-edge pair (v_i, v_j) is contracted, joining previous unconnected area [Gar99] 41
4.3 Simplification of a 3D point cloud by using cluster [PGK02] 42
4.4 Determining of neighboring points [ZG10] 43
4.5 Clustering by passing messages between data points [FD07] 43

4.6 Decimation of 3D point clouds by using RBF, K-NN kernel [MWZ10] . . . 44
4.7 a) The size of a cell. b) The barycenter of the points (red color) in the cell. 45
4.8 2D illustration of the octree subdivision, we actually handle in the 3D structure. . . . 46
4.9 The level of subdivision in a cell. . . . 47
4.10 Computing the average subdivision level of neighboring cells (cell a is computed based on cells b, c and d). . . . 48
4.11 Estimation of the curvature in a cell: (a) Computing the orientation of points; (b) The points are approximately on a plane within a cell (λ_0 is very small, λ_1 and λ_2 are large); (c) λ_0 is large or ($\lambda_0 \simeq \lambda_1 \simeq \lambda_2 \simeq 1$) or ($\partial \simeq 1/3$) $\Rightarrow$ this cell is subdivided. . . . 50
4.12 Illustration of the elaborate algorithm. . . . 51
4.13 Rough simplification: the shape of the initial surface is not preserved and received results are not accurate using a large cell size. a) a geological surface; b) after simplifying with cell size: $s = 3$ (Δ_{max}: 0.006, Δ_{avg}: 0.0002); c) after simplifying with cell size: $s = 6$ (Δ_{max}: 0.017, Δ_{avg}: 0.0003). . . . 54
4.14 Shape comparison by computing the approximation error of the surface after simplifying with the same size of neighboring distance: b) using the rough method (Δ_{max}: 0.0142, Δ_{avg}: 0.0004); c) using the cluster method (Δ_{max}: 0.0296, Δ_{avg}: 0.0009). . . . 55
4.15 Comparison of the shape of the surface between the two methods by using the same size of neighboring distance: a) an input surface of 23559 points; b) after using the elaborate method, remaining points: 2305, the approximation error between (a) and (b) is Δ_{max}: 0.007, Δ_{avg}: 0.0007; c) after using the cluster method, remaining points: 801, the approximation error between (a) and (c) is Δ_{max}: 0.015, Δ_{avg}: 0.003. . . . 57
4.16 a) Input surface with 66049 points; b) After simplifying by using the elaborate method (s=8, $\partial \leq 0.09$, remaining points: 1840), the points are constrained from the boundary to the inside; c) A good triangular surface can be obtained in a further meshing step (the approximation error between (a) and (c) is Δ_{max}: 0.018; Δ_{avg}: 0.002) . . . 57
4.17 Comparison of the shape of a geological surface: a) input surface of 664582 points; b) after simplifying by using the elaborate method (s=8, $\partial \leq 0.15$), the simplification rate: 89%; the approximation error between (a) and (b) Δ_{max}: 0.0248; Δ_{avg}: 0.0004. . . . 58
4.18 Comparison of the approximation errors: a) input surface with 2136 kb; b) after simplifying by using the elaborate method, s=8, $\partial \leq 0.12$, the remain data: 309 kb; the approximation errors between (a) and (b) are Δ_{max}: 0.015; Δ_{avg}: 0.0005; c) after simplifying by using the cluster method, cluster size = 8, the remain data: 46 kb; the approximation errors between (a) and (c) are Δ_{max}: 0.034; Δ_{avg}: 0.0014. . . . 58
4.19 Determining of neighboring points. . . . 62

5.1 2D Delaunay triangulation by using a uniform grid [FP93]. 66
5.2 Sweep-line DT triangulation by using an advancing front [DZ08]. 67
5.3 Intuitive illustration of Poisson reconstruction in 2D [KBH06]. 68
5.4 A 2D illustration of BPA algorithm [BMR+99]: a) a reconstructed curve connects the points using a circle of radius ρ pivots; b) the sampling density is too low to create the edges with the user-specified radius ρ; c) the user-specified radius ρ is too large to reconstruct some high curvature regions. . 69
5.5 Reconstructing a triangular mesh based on IPD [LTW04]. 69
5.6 Triangulation of a 3D point cloud by projecting it onto the 2D grid. 71
5.7 Some concepts and notations used in the surface triangulation 71
5.8 A case of $CDBE$ 72
5.9 The general diagram of our method 73
5.10 Determination of the first Delaunay edge. 74
5.11 Condition of non-intersection for generated triangles. 75
5.12 The voxels traversal search 77
5.13 Iteration of Woo algorithm 80
5.14 Computing the successive dilatations of the discrete mediatrix of e_i. 81
5.15 a) The convex hull of a triangular surface; b) The outside triangles of the boundary. 84
5.16 Determination of an outward triangle on the boundary. 84
5.17 Searching a neighboring point: a) computing the compactness; b) voxel traversal search. 86
5.18 Comparison of the processing times between the methods: Computing the compactness (CTC); Voxel traversal search (VTS) and Ball pivoting (BP). On this graph, we do not plot the last example with 886639 points because it is too far from other examples and therefore spoils the graph. 87
5.19 Processing the triangular faces on the boundary; (a) an input surface of 3D point clouds with boundary points (red color); (b) a triangular surface; (c) after removing the outside triangles; similarity, (d) before and (e) after deleting the outward triangles on the boundary. 88
5.20 a) A geological surface of 3D point clouds (232 kb). b) After simplifying by using the elaborate method (cell size = 8, $\partial \leq 0.12$) and triangulating, the size of surface: 18 kb; the approximation error between (a) and (b) is Δ_{max}: 0.020; Δ_{avg}: 0.0006; the triangular faces vary in density from the boundary to the inside of the surface. 89
5.21 a) The input surface of 3D point clouds with 2629 kb. b) After simplifying by using the elaborate method (cell size = 8, $\partial \leq 0.09$) and triangulating, the size of surface: 68 kb; the approximation error between (a) and (b) is Δ_{max}: 0.018; Δ_{avg}: 0.002; the characteristics of the surface are well preserved. 90

6.1 Description of our methods for constructing an optimal geological surface. . 94
6.2 Boundary extraction of the geological surface. 95
6.3 Boundary simplification of the geological surface. 95

6.4 Rough simplification of the geological surface. 96
6.5 Elaborate simplification of the geological surface. 96
6.6 Triangulation of the geological surface. 97

List of Tables

4.1 Comparison between the rough method and the cluster method. We use the same size of a neighboring distance between the points (cell size: s = cluster size = 6), and run them on the same a computer. (p.output: output points; s.rate: simplification rate) . . . 53

4.2 Comparison of the rough method: we use the different cell-sizes and run them on the same a computer (s.rate: simplification rate). The results with cell size (s=3) can be found in table 4.3 . . . 53

4.3 Comparison between the rough method (cell size $s = 3$) and the elaborate method. Time1: the computing time by using only step2 (initial cell size $s = 8$); Time2: the total computing time by using both steps (rough first: cell size $s = 3$; then elaborate: initial cell size $s = 8$); p.output: output points, s.rate: simplification rate. . . . 56

4.4 Comparison of the best results between the cluster method (cluster size s=8) and the global method (cell size: rough s=3; elaborate s=10). We run on the same a computer. p.output: output points; s.rate: simplification rate. . 59

4.5 Comparison between the rough method (cell size $s = 6$) and the global method. In the global method, we use both steps to simplify the surface (rough first: cell size $s = 6$; then elaborate: initial cell size $s = 10$); p.output: output points, s.rate: simplification rate. . . . 60

4.6 Comparison between the rough method (cell size $s = 8$) and the global method. In the global method, we use both steps to simplify the surface (rough first: cell size $s = 8$; then elaborate: initial cell size $s = 10$); p.output: output points, s.rate: simplification rate. . . . 61

5.1 Comparison of the processing times between the methods. We use the same input data points and run on the same a computer (Intel 2CoreDue, 2GB of Ram). . . . 87

List of Algorithms

2.1 ExtractCoordinates(file f) 15
3.1 GrowthDistance(p) 27
3.2 GrowthClockWise(p) 28
3.3 SimplifyBoundary() 31
4.1 RoughSimplification(s) 45
4.2 SimplifyBoundaryCells(s) 51
4.3 SimplifyInnerCells(s) 52
5.1 Circumcircle(p_2^i, p_1^i, p_3^i) 77
5.2 WooLine(e_i, k, rl) 79
5.3 SearchNeighbors(e_i, p_3^i) 82
5.4 MeshGenerating(S) 83

Nomenclature

The list below contains the mathematical symbols and notations that are used most frequently throughout the thesis:

G	=	the 2D grid (bounding box of the projection of the 3D point clouds on the x, y plane)
S	=	the subset of cells containing projected points ($S \subseteq G$)
C	=	the regular grid of size s built over G
N_p	=	the number of points in S
p_i	=	a point i^{th} in S
p_b	=	a boundary point of surface S
k	=	the distance between points for determining a neighboring point p_n of p_i ($1 \leq$ k (integer) $\leq$ threshold)
S_q	=	a cell on the 2D grid belonging to S
N_{sq}	=	the number of points in S_q
p_q	=	barycenter of the points included in S_q
C_q	=	a cell (size s) on the 3D grid
N_{cq}	=	the number of points in each C_q
N_{bp}	=	the number of boundary points in C_q
C_v	=	covariance matrix
T_{first}	=	the first triangle face of triangular surface S
T_{next}	=	the next triangle face which adjacent to T_{first}
$\triangle(p_1, p_2, p_3)$	=	a triangle formed by three points: p_1, p_2, p_3
$\angle(p_1, p_2, p_3)$	=	an angle at p_2
e_i	=	an edge i^{th} of a triangle face
e_b	=	a boundary edge of a triangle face
kb	=	Kilobyte
ms	=	Millisecond

Chapter 1

Introduction

Contents

1.1 Background and motivation 1
1.2 Problem statement 2
1.3 Objects and scopes 3
1.3.1 Methodology 3
1.3.2 Geological data model 3
1.4 Thesis structure 5

1.1 Background and motivation

In recent years, the research cooperation between scientists in geology, computer science and geometric modeling have been developed. From 2004, a scientific cooperation program between the universities: Ecole des Mines de Paris, Poitiers University, Strasbourg University, Laboratory LSIS (Aix-Marseille University) and the IFP (French Institute of Petroleum) [Gui06, Tra08, Ver09] has started. The goal is to improve an innovative solution for simplification of the 3D model production of oil reservoirs. Thereafter, the software productions have been developed as a tool for simulating and modeling the oil reservoir. Among many tools, RML (Reservoir Modeling Line) is developed by IFP to construct a structure model using parametric surfaces.
In 2006, an exhaustive study of the necessary functions for the development of geometrical modeling conducted at the IFP has raised a set of issues. They decided to work on a new version of this soft based on the project "Geological Evolution Scheme" and "Modeling Guided by Geology" which can process the whole model. At present, one drawback of the current tools is that they cannot handle a mass of data.
One associated research is the subject of this thesis: It concerns the creation of a geological triangulated surface built from a huge amount of 3D points extracted from seismic cubes.

This is to be able to manipulate the resulting surfaces of geological interfaces in an interactive environment. The aim of this thesis is to combine geometry, topology and physics to develop methods more efficient, robust and reliable than currently available tools in the IFP.

1.2 Problem statement

The acquisition techniques allow obtaining the different data of oil reservoirs through seismic acquisition. During many years, the data of oil reservoirs have been changed on both quantity and structure. At present, our problem is to obtain a high quality definition of oil reservoirs in order to get the best estimation of the potential petrol they may contain. Promising results have already been obtained [Tra08, BDRT09, BTD08, BTD07] starting from 3D scattered points. The main issue is the huge number of data points on these surfaces.
The difficulty now is to be able to process a very large point clouds (a mass of data). The input data are clouds of huge amount of uncertain 3D points, containing a lot of noise, representing geological formation boundaries (horizons) and geological formation deformation area (faults) on one side. In addition, 3D point clouds correspond to the intersection of these geological events and well path trajectories. These clouds of points are characterized by an important difference on the scale between the x, y plane (some kilometers) and the z variation (some decimeters). In fact, a basin model can be reached from 10 to 100 km.
The quantity of information can be very large, up to several tens of millions of points. So the memory, processing time and methods to process these surfaces should be studied because the existing ways cannot handle them. Even if we can use more memory allocation (extension RAM, HDD), the time required to process these surfaces is currently not acceptable.
In this research, the most important difference from the previous works comes from the data. The received data were before a set of 3D points defined by x, y, z coordinates. These data were in fact extracted from seismic information and a fine analysis shows us that many data are missing. This leads to the methods developed in Van's thesis [Tra08] and there is a lock due to the mass of data.
These data were obtained from the treatment of seismic data organized in the 3D volume as defined by Philippe Verney [Ver09]; and a fine study showed us that important information exist in the 3D volume and were actually lost when compiled to obtain the x,y,z file. Therefore, in this thesis, we decided to proceed data directly in this seismic cube. The new data have been analyzed, simplified and organized in a matrix. Different information of points are missing because the seismic acquisition does not permit to measure all the points in the 3D volume. Therefore, this matrix is a sparse matrix. The fundamental point is that data are known on a 3D regular grid. The neighborhood relationships are a great advantage for 3D points processing. We can easily determine a neighboring point of a point based on the distance between them. Having such neighborhood provides an opportunity to have a completely different approach of the first processing yielding an acceptable tri-

angulation. Nevertheless, in this thesis, the problem of fault on the surface is not handled because if two points exist with the same coordinates x, y but different z coordinates, this has been discarded before our work.

1.3 Objects and scopes

1.3.1 Methodology

The objects and scopes of our work are focused on the methods and algorithms for surface simplification, surface reconstruction and surface triangulation of 3D point clouds. We introduce previous works on 3D elevation surface modeling in chapter 2. Thereafter, the existing methods and our methods will be presented in detail in the next chapters.

1.3.2 Geological data model

In order to understand an overview of a geological data model, we introduce in this section the existing methods for constructing a geological model and data automatic acquisition technique. The 3D simulation techniques are applied in the field of geology. Richard et al [Jr.06] provided an overview of the full techniques and methods for constructing a 3D geological model (see figure 1.1).

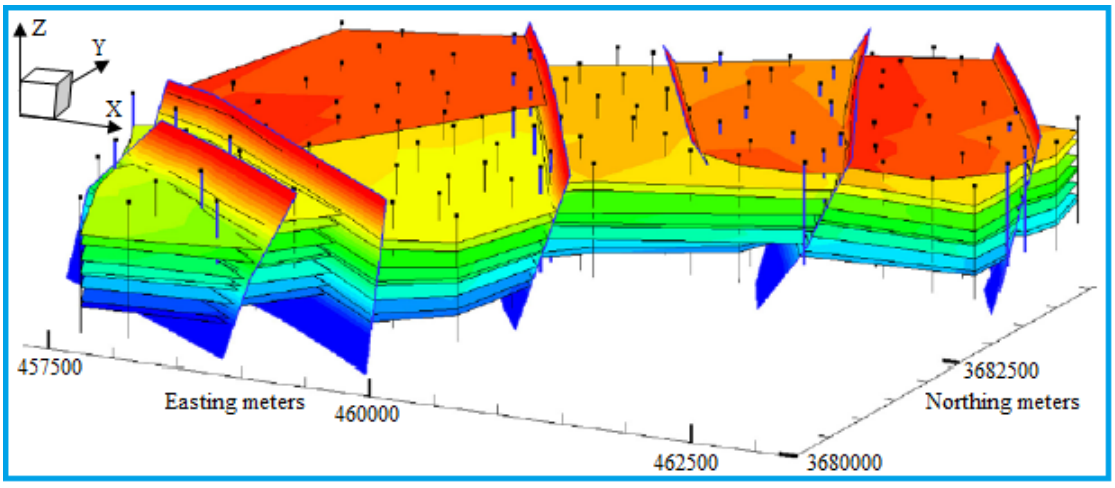

Figure 1.1: 3-D oblique view of a portion of the Black Warrior Basin (data from [Jr.06]). Thin vertical lines are wells; semi-transparent surfaces outlined in black are faults

The existing researches [BSP+04, Gui06, Tra08, Ver09, Mas10] in the oil exploration industry have shown that the computation, building and simulation of oil reservoir on the computer are based on geometric modeling (see figure 1.2). In order to get the data of oil reservoir, the scientists have used a special device to get the information of oil by using

an automatic acquisition technique or processing the seismic images (see figure 1.3). After obtaining the data, they are processed and structured as 3D point clouds into a sparse 3D volume for the next surface processing. The description of data processing will be completed in chapter 2.

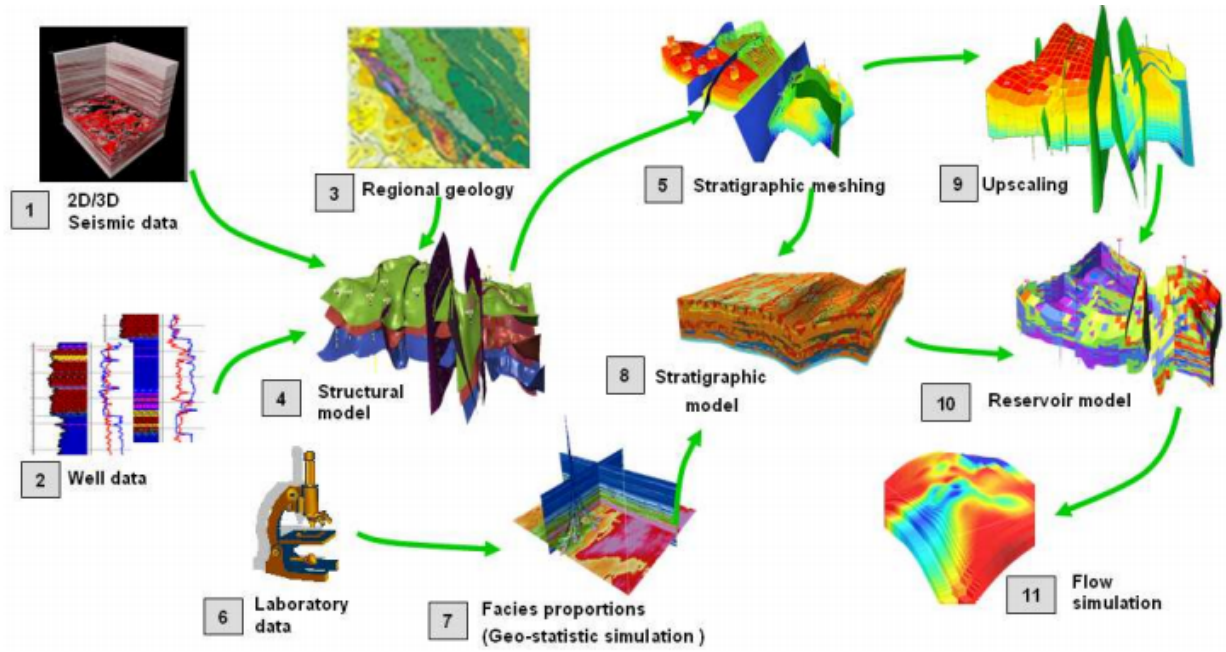

Figure 1.2: A geological model used in the oil exploration [Mas10]

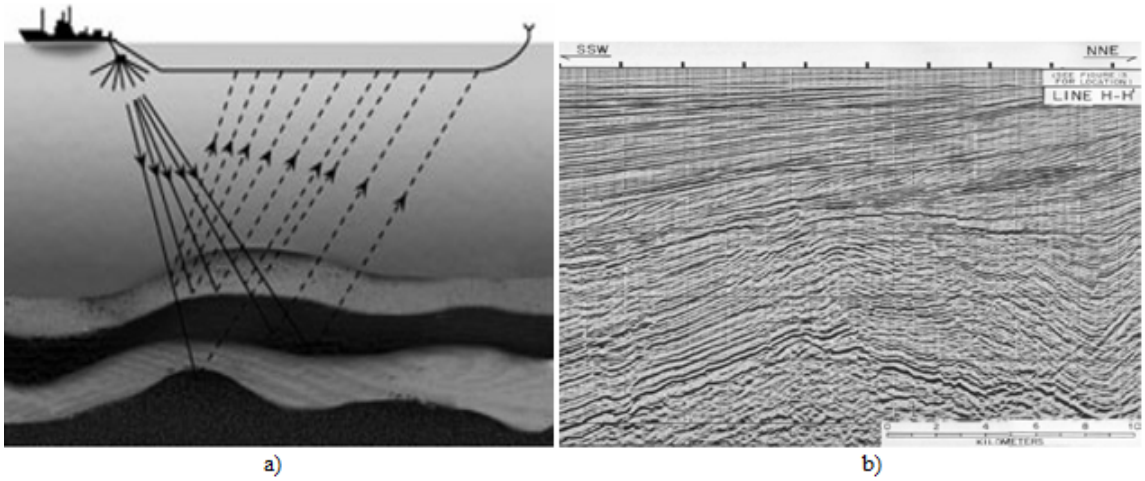

Figure 1.3: Collection of geological data: a) by a marine seismic acquisition; b) by processing seismic images

1.4 Thesis structure

The general model of our work is described in figure 1.4. Starting from the input surface of 3D point clouds, we first make an extraction and simplification of the boundary of the surface. The surface inside its boundary is then simplified. The last work is triangulating the surface after simplification.

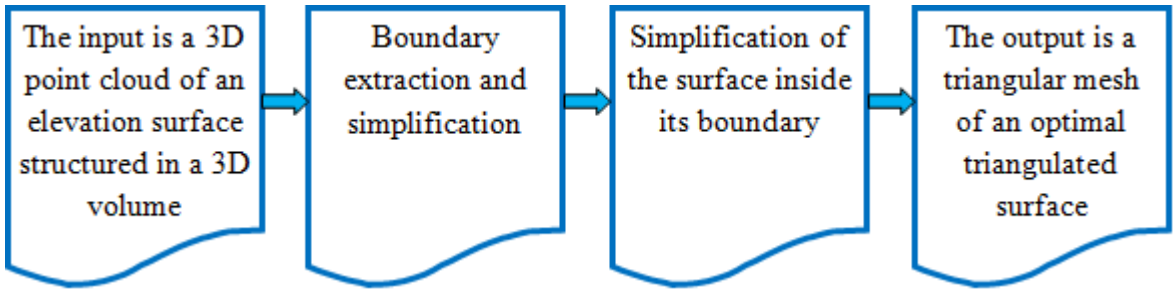

Figure 1.4: General model of building an optimal geological triangulated surface

We present previous works for constructing a geological surface and seismic data processing in chapter 2. Chapter 3 introduces the methods for boundary extraction and simplification of a surface of 3D point clouds. In the first part, we present the definitions of an exterior boundary of the surface. The second part, the methods for extracting this boundary are presented in detail. In the third part, we present a method to simplify this 3D boundary. In chapter 4, we propose the methods for simplifying the surface inside its boundary. This method based on the subdivision following the boundary density of a surface, and according to the curvature of the surface. Chapter 5 addresses a method for triangulating a surface defined by a sparse 3D volume. The last chapter (Chapter 6) presents our conclusions, discussions, evaluations, contributions and strategical directions for the future researches.

Chapter 2

Previous Works on 3D Elevation Surface Modeling

Contents

2.1 Tran Nam-Van's work **8**
2.1.1 Proposed methods 8
2.1.2 Obtained results and existed problems 8
2.2 Philippe Verney's work **10**
2.2.1 Images processing 10
2.2.2 Structure of a sparse matrix 11
2.2.3 From a sparse matrix to a 3D volume 13
2.3 Conclusion **16**

Our works in this thesis are studied and developed based on the previous works of 3D elevation surface modeling. In this section, we first review the methods for processing the surfaces of 3D point clouds as it was done before. These methods such as simplification, hole-filing and reconstruction a triangular mesh which have been proposed and presented by Dr.Tran Nam Van [Tra08] in his PhD thesis in 2008. This presentation will allow the reader to understand the important difference of approach we introduce in this thesis. Then, we present in detail the processing of geological data, which leads to the data structure in a sparse 3D volume, starting point of the new approach. This work have been studied and finished in the PhD thesis in 2009 by Dr.Philippe Verney [Ver09].
In order to understand this section, the reader only needs to know what is a planar triangulation. A planar triangulation of a convex point set S is a set of triangles which entirely cover S and where triangles intersect only in one point or one edge (no overlap). We will propose more results on triangulation on chapter 5.

2.1 Tran Nam-Van's work

The work of Van's thesis is a part of collaboration with the IFP - French Institute of Petroleum. The starting data is a set of triangles obtained by a triangulation method of the *TTL* library. The points which were triangulated were actually extracted from the seismic 3D volume presented in the next section. All the neighboring information implicitly existing in this volume was in fact lost.

2.1.1 Proposed methods

In the first work, he proposed a method for simplifying the triangular mesh surface where vertices are clustered, followed by an iterative edge collapse step. More precisely, vertices are first clustered into surface patches through an adaptive segmentation process (using both absolute discrete curvature and principal component analysis); the edge collapse process is based on quadratic error metrics [BTD07].
According to the characteristics of the geological surfaces and the data acquisition technology (e.g. geo-seismic or geo-magnetic), data can miss in some regions of the surfaces, holes exist on these surfaces. Such holes are not acceptable either for geologists or computer scientists and can induce unexpected results when reconstructing the 3D models. For these reasons, in the next step, he studied a method for filling the holes of a triangular surface. The method consists in a multistep approach: first, holes areas are identified; second, fill the holes; and the last, the surface is refined in order to preserve the initial shapes of the surface [BTD08].
One of the specific characteristic in geological surfaces is the fault. This is a fracture of geological layers (horizons) generated by landslides or volcano moving in the earth. In order to improve the geological models, after the two above steps, he suggested a method for detecting and reconstructing the faults of the surfaces. The method includes pre-processing meshes, detection of faults, reconstruction and extrapolation of faults. The obtained results of Van's work are presented in the next section.

2.1.2 Obtained results and existed problems

The hybrid simplification method allowed Van to solve his initial problems, namely, to simplify voluminous data while preserving strongly bend areas and curvatures. After simplifying a triangular mesh, a large number of triangles are simplified while preserving the original geometric shapes of the surfaces (see figure 2.1). This method can also be applied to simplify any types of surface triangulation.
The next results obtained in this thesis are the methods for filling and fairing the holes of the triangular meshes. Hole fairing is performed by minimizing a discretization of the thin-plate energy, which avoids the estimation of normals, tangents and curvatures on the hole neighborhood. This method produces meshes of good quality; the reconstructed surfaces are smooth and very close to the initial model. This method can be also applied for the reconstruction of fault surfaces with ridge and ravine curves (see figure 2.2).

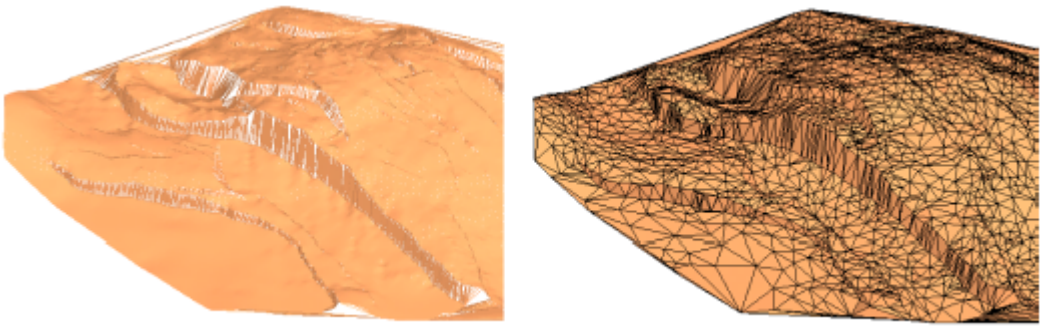

Figure 2.1: A geological surface simplified with method [BTD07]: a) initial model, 112kb vertices; b) output model, 3kb vertices

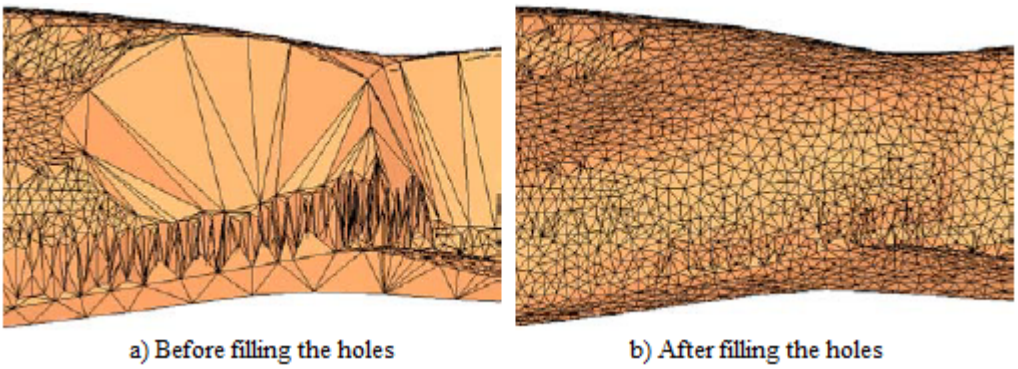

Figure 2.2: Before and after filling the holes of a geological surface [BTD08]

The last results but not least important is detecting, reconstructing and optimizing the detected faults of the surfaces. Figure 2.3 is an illustration of this processing:

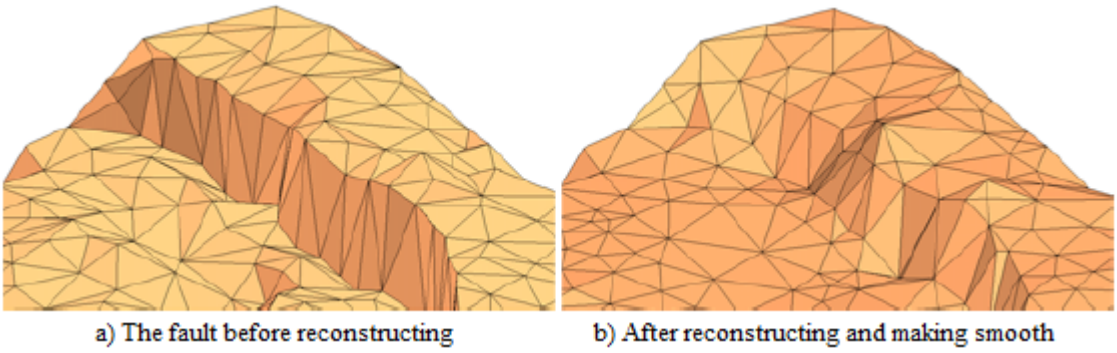

Figure 2.3: Before and after processing the fault of a geological surface [Tra08]

However, there exists a drawback in data processing. The computing of Delaunay Triangulation for a geological triangular surface can be slow with a very large point cloud. Thus, the geological data need to be pre-processed in order to reduce a number of data points

before modeling the surfaces. The next section, we will describe a method for processing the geological data of Philippe Verney in his PhD thesis.

2.2 Philippe Verney's work

In Philippe Verney's work [Ver09], the input data are seismic images. He suggested a method to process these images in order to get the information of geological surfaces in a seismic block. One of the output result is a sparse matrix, which stores the information, attributes of geological model by voxels.

2.2.1 Images processing

In order to save the memory of the computer and reduce the size of a seismic block for a further processing step, only the areas containing strong reflections (by analyzing the geological images) are interesting. Then, this data are put into a table (see figure 2.4), column by column with the following attributes:

- A seismic image is a lateral succession of traces (columns).
- Each trace (column) is composed by a vertical succession of positive and negative ranges of values (values are called amplitudes and range of positive or negative values are called reflections).
- Commonly, the highest absolute values of a range are located at the middle of a range.

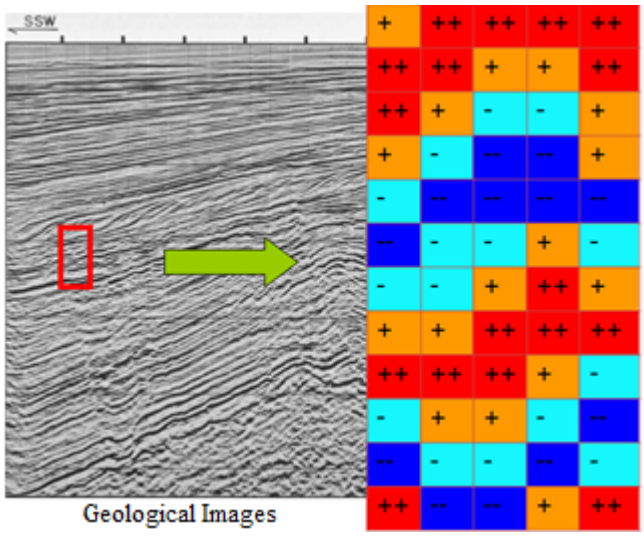

Figure 2.4: Extracting the information from geological images. [Ver09]

In a next step, he proposed a method for filtering these data to reduce the number of pixels in the table. Figure 2.5 is an illustration, which presents a table with five traces (column), related to a seismic section. Only the highest amplitude values are stored in the memory, all blank cells are ignored. For example, the first reflection at the top left of the table has a thickness of four pixels. After threshold, this thickness is reduced to two pixels. Therefore, it can reduce a half of the required memory to store the table.

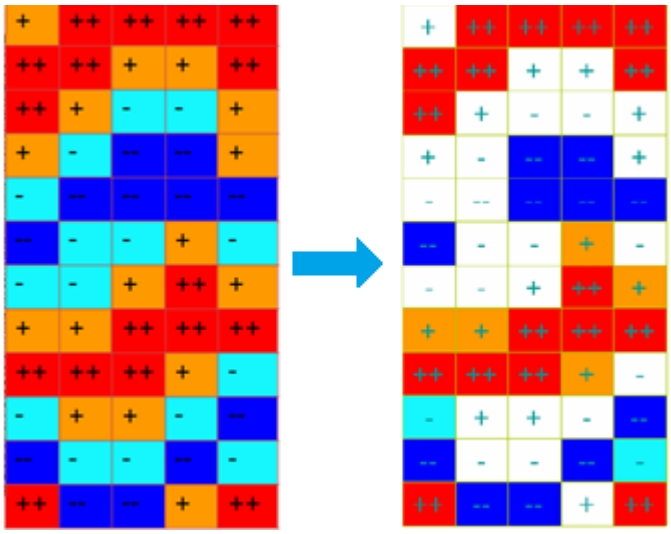

Figure 2.5: Filtering the information of seismic images. [Ver09]

However, the reflector associated with this reflection is only a surface passing through the maximum of this reflection. So it is a surface that passes through a trace to a single point and therefore has a thickness of one pixel. For this reason, he chose to thin any thickness of each reflection to a single pixel. This pixel is arbitrarily fixed in the middle of the reflection to provide a better lateral continuity. After two operations, it is possible to transform the representation of a seismic image initially formed by a 3D matrix-full amplitude values into a coarser form of a 3D matrix hollow. The last results are shown in figure 2.6.
The next step is detecting the surface; then, adding the attribute to the surface. As presented in figure 2.7: on the left, the label is assigned by means of a surface ID of connected voxels; on the right: the thickness computing.

2.2.2 Structure of a sparse matrix

The obtained results of geological images processing are stored in a sparse matrix (see figure 2.8). It can be defined as follow:

- Sparse matrix is a partial information of a seismic block in 3D Grid.
- A sparse matrix contains a set of layers.
- A layer is a volume which is delimited by two or more horizons.

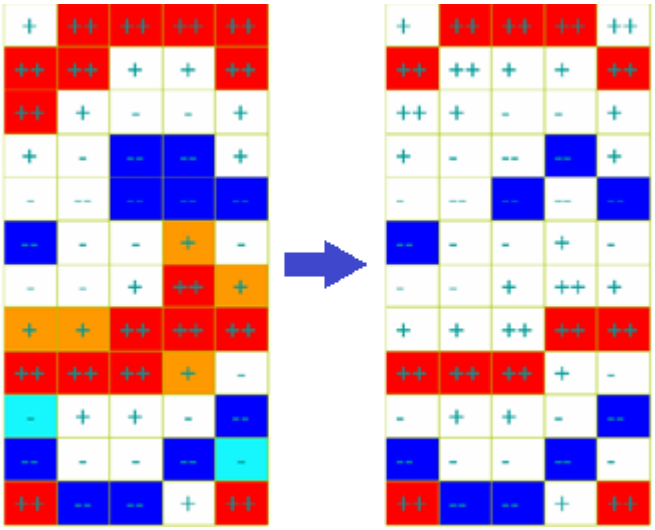

Figure 2.6: Thinning of the reflections. [Ver09]

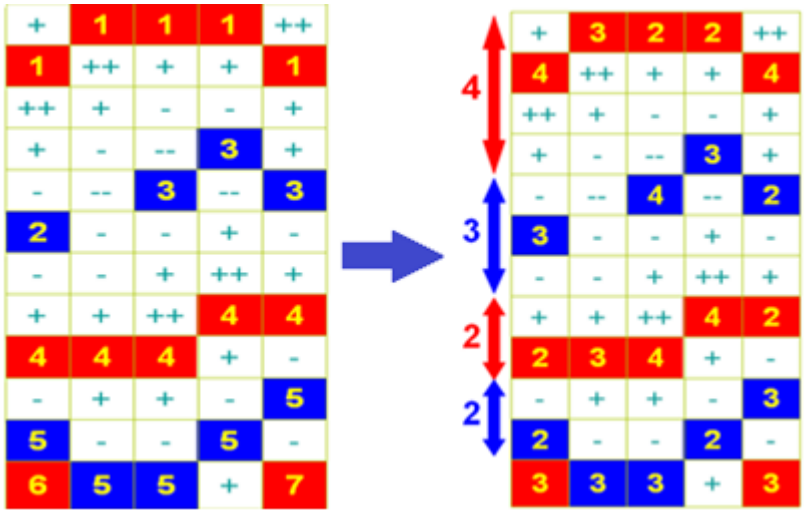

Figure 2.7: Determining the surface and computing its thickness. [Ver09]

- A horizon is a surface composed of two or more patches.
- A patch is a part of a surface containing a lot of voxels.
- A voxel is equivalent to one point in a 3D point cloud.

We can see on the left of figure 2.8 that there are a lot of empty voxels. Each non-empty voxel in the sparse matrix has five attributes (one of them is the mathematical attribute and four other are geological attributes) as below:

- *Z index* of a voxel (mathematical attribute) is the discrete index of the voxel in the vertical dimension of the seismic cube.

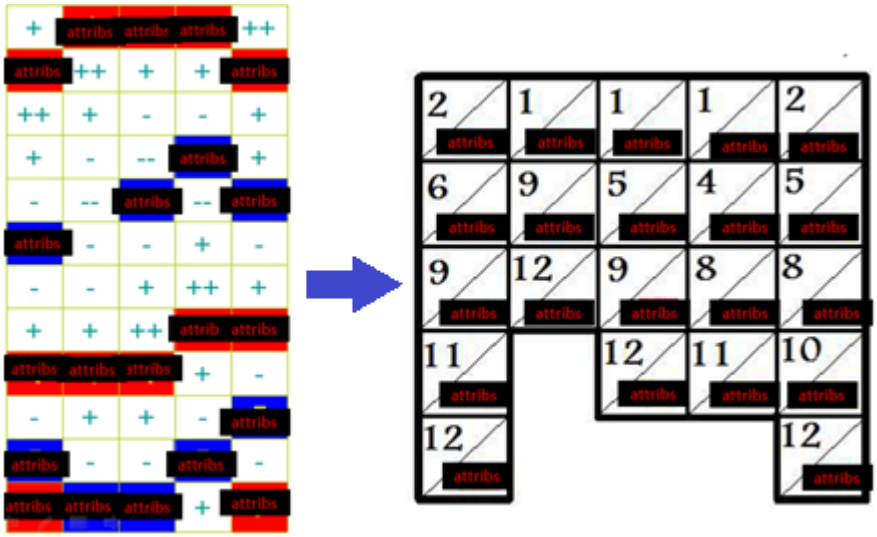

Figure 2.8: Sparse Matrix. [Ver09]

- ID of the surface (horizon), the voxel is part of. (The ID of a surface is also the ID of an horizon because an horizon is a surface composed of two or more patches; the patches (belong to a surface) have the same ID of their surface).
- *Thickness* (*width*) of the surface (i.e. of the value range) at this location.
- *Mean amplitude* of the surface (i.e. of the value range) at this location.
- *real Z* position is the result transformed from *Z index*. (*real Z* is used in other application and is not described in this presentation).

In the end, the structure of a sparse matrix looks like the right hand side of figure 2.8. The sparse matrix is then stored into a dedicated structure where the number of colums are the same but not the number of lines. For example, look at the right hand side of figure 2.8: number "2" (from the top-left conner) is an attribute belong to row two and column one of figure on the left hand side; similarly, number "6" is row six and column one, etc. Each voxel is identified by its column and its Z index in the original matrix.

2.2.3 From a sparse matrix to a 3D volume

In order to process the geologic surfaces in the sparse matrix, we first extract the 3D coordinates (xyz) of each voxel from its five attributes in the sparse matrix. Then, insert these voxels into a 3D volume. Thereafter, each voxel is considered as a 3D point, and has three real coordinates xyz in this 3D volume (see figure 2.9). Finally, the 3D volume contains a set of horizons (i.e. surfaces, because an horizon is a surface). There are some data that have been discarded, but implicitly the 3D volume contains the neighborhood information for each point. The method for extracting 3D points in the sparse matrix is presented in the next section.

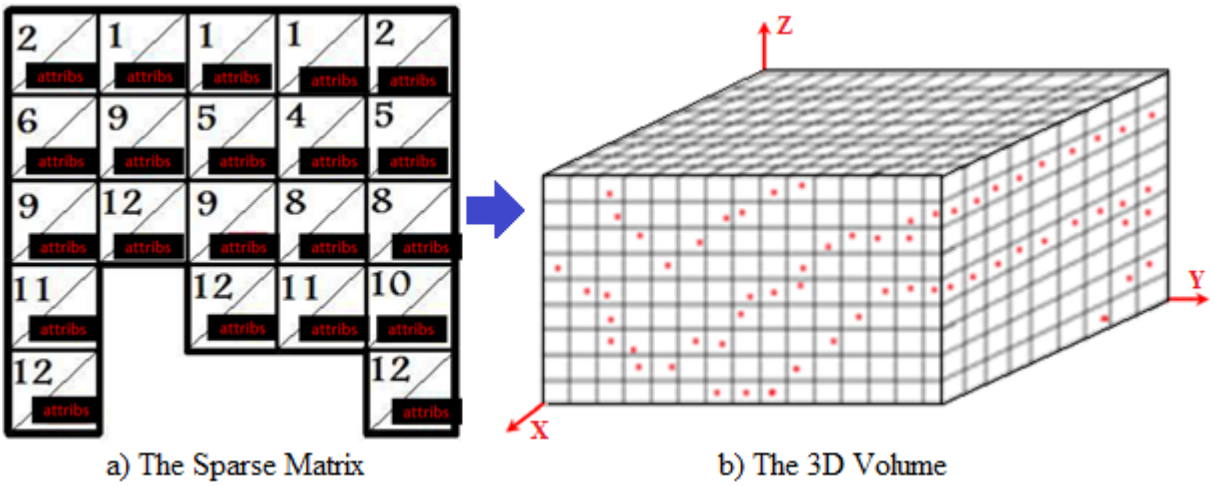

Figure 2.9: 3D points are extracted from the sparse matrix and insert into a 3D volume.

2.2.3.1 Data analysis

As we described above, each voxel in the sparse matrix contains five attributes. They are organized in a file, line by line. Each line corresponds to one column of the sparse matrix (the same column of the original seismic matrix). One line can have more than 5 values; they are equivalent to the five attributes of voxel ($Z\,index$, ID of surface, $width$, $amplitude$, and $real\,Z$). Each line can have $(5 \times n)$ values where n is the number of voxels of that column. For example: if a line contains 10 values, it means that the corresponding column has two voxels; 15 values, means three voxels; etc. The sequence is always in the same order: ($Z\,index$ of voxel 1, ID surface (horizon), the voxel 1 belongs to, $width$ of voxel 1, $amplitude$ of voxel 1, $real\,Z$ of voxel 1); ($Z\,index$ of voxel 2, ID surface of voxel 2, $width$ of voxel 2, $amplitude$ of voxel 2, $real\,Z$ of voxel 2); etc.
This file includes thousands of lines (rows). The first line always contains three numbers, representing three dimensions of the volume of this matrix (nX, nY, nZ):

- nX: the number of cells we can find for one constant Y and one constant Z;
- nY: the number of cells we can find for one constant X and one constant Z;
- nZ: the number of cells we can find for one constant X and one constant Y;

Actually, a voxel is a cell of a 3D grid. We have a 3D volume containing $(nX \times nY \times nZ)$ number of voxels/cells, but inside there are a lot of empty cells; and an empty cell is not written on the file.

2.2.3.2 Proposed algorithm

From the above analysis, we did not see the values of coordinates x and y in every voxels. There is only the $Z\,index$ (first attribute of voxel in the sparse matrix) and also the

coordinate values z of voxels. The algorithm below (Algorithm 2.1) is used to extract the coordinates (xyz) for each voxel.

Algorithm 2.1 ExtractCoordinates(file f)

```
read f;
x = 0; y = 0;
readOneLine;
while not end of file do
    print x, y, z (z from Z index of every voxels in line);  //write to an output file
    x++;
    if x == nX then  //nX is the first number of the first line of the file
        assign the value of x back to 0;
        y++;
        if y == nY then //nY is the second number of the first line of the file
            exit;
        end if
    end if
    readOneLine;
end while
```

2.2.3.3 Results

After implementing algorithm 2.1, the obtained results are a set of surfaces (horizons) of 3D point clouds defined by a sparse 3D volume. Each surface is composed of two or more patches. An example is proposed in figure 2.10 and 2.11.

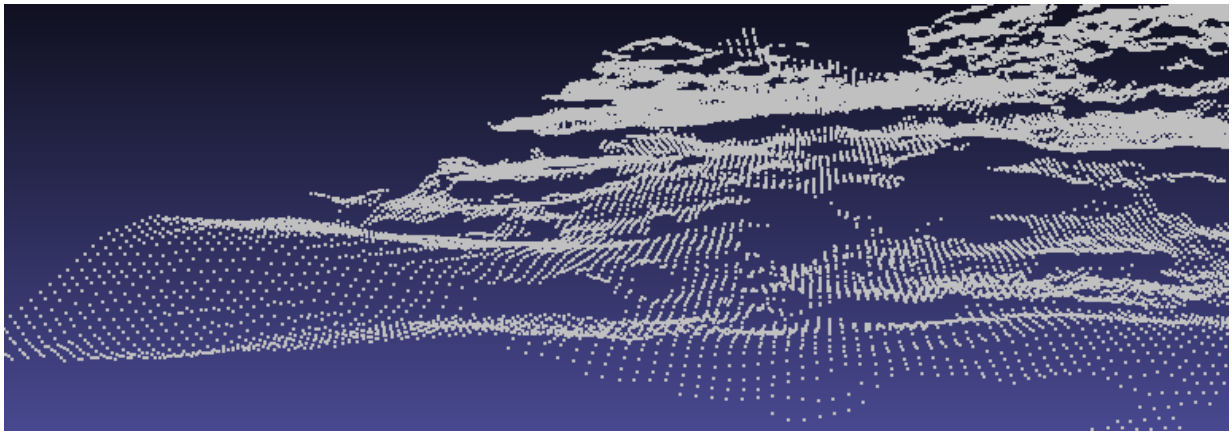

Figure 2.10: An elevation surface (346796 points) defined by a sparse 3D volume.

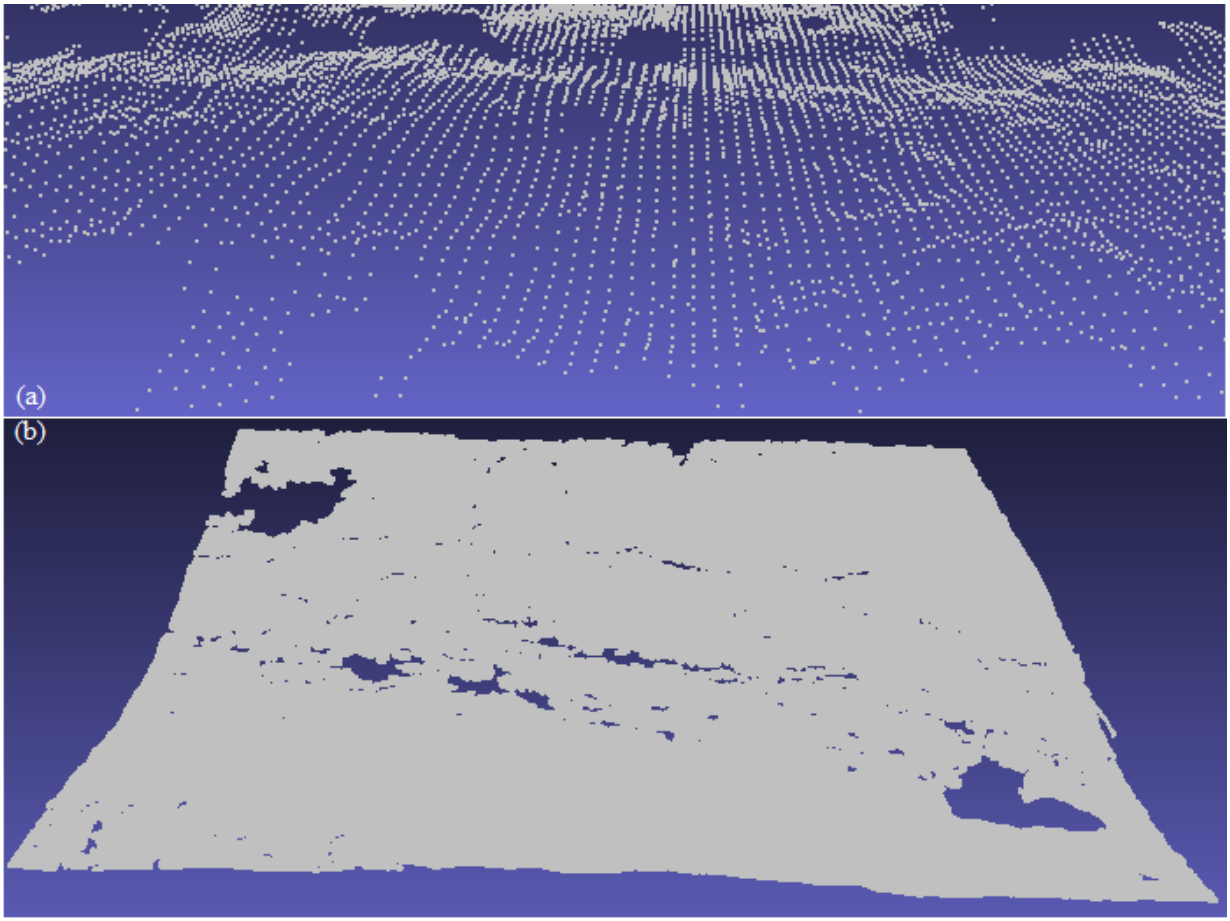

Figure 2.11: (a) An elevation surface of 257629 points; (b) An elevation surface of 886639 points.

2.3 Conclusion

In this chapter, we have studied the methods proposed in the previous works for processing the geological surfaces. From this study, we have discovered the disadvantage in Van's work, with which it can be difficult to handle huge 3D point clouds. We have reviewed a method for processing the geological data in the PhD thesis of Philippe. The obtained results are information of a seismic block structured in a sparse matrix. Based on this sparse matrix, we propose in this thesis a method to directly obtain the simplified triangulation obtained by Van.
The next chapter will present the extraction and simplification of the surface boundary.

Chapter 3

Boundary Extraction and Simplification

Contents

3.1 Introduction **17**
3.2 Related work **18**
3.2.1 Boundary detection 18
3.2.2 Boundary simplification 21
3.3 Methods for boundary extraction **22**
3.3.1 Overview 22
3.3.2 Definitions 23
3.3.3 Algorithms 25
3.3.4 Implementation 29
3.4 Methods for boundary simplification **30**
3.4.1 Background and Definitions 30
3.4.2 Algorithm 31
3.5 Results **32**
3.5.1 Boundary Extraction 32
3.5.2 Boundary simplification 32
3.6 Discussion and evaluation **33**
3.7 Conclusion **38**

3.1 Introduction

Geometric modeling and surface processing often handle surfaces defined by data in 3D volumes corresponding to a cube of voxels and which are a direct extension in 3D of pixels

and images in 2D. As a matter of fact, acquisition techniques often produce such data structures. The most popularly encountered applications are in medicine with scanner and MRI devices. Reconstructing surfaces with data coming from an automatic acquisition technique always entails the problem of mass of data. It leads to a mandatory data reduction process. Applying the process to the whole set of points induces an important risk of surface shrinking so that the initial boundary extraction is an important step permitting a simplification inside it. The global surface shape will then be better kept. It is nevertheless required to simplify the boundary, which can be done on the extracted boundary.
As we mentioned in chapter two, promising results have already been obtained [BDM05, BTD07, BTD08, Tra08, BDRT09] starting from 3D scattered points. The main issue is the huge number of data points these surfaces contain. Even with an important simplification step, memory requirements are too high. We thus decided to start our processes directly in the seismic cube (defined by Philippe Verney [Ver09] as presented in chapter two). One of the main advantage of 3D volumes is the fact that they provide evident neighboring relationships between voxels (contrary to scattered data which require an initial triangulation). 3D volumes offer a valuable opportunity to directly simplify the data onto the volume taking advantage of this natural connectivity. But applying the simplification to the whole set of points will necessarily modify and shrink the boundary, which must be avoided in order to keep the global shape of the surface. A particularity of seismic data is their poor accuracy and the fact that many data are missing so that we have to handle a sparse 3D volume, leading to specific problems justifying our current study.
We present in this chapter a new method to extract and simplify the boundary of an elevation surface given as voxels in a large 3D volume having the characteristics to be sparse since many data are missing. We first present our boundary definition based on mathematical relations between a point and its square neighborhoods. Second, we introduce algorithms to extract such a boundary. Third, we simplify this boundary. In the next section, we present works related to boundary detection for point clouds sampled from a 3D surface, and simplification algorithms for such boundaries.

3.2 Related work

3.2.1 Boundary detection

Determining the boundary of a surface is an important issue in the field of geometric modeling. Traditionally, approaches computing the boundary of a surface either use the convex hull or convex envelope of a set of 3D points. Other approaches are based on the calculation of an angle or of the distance between points and theirs neighbors. Different methods have been developed to extract the boundary [SS07, XXFJ08, Wei08, KNSS09, SKM11], although, in some cases, the computation seems too slow.
Sampath et al (2007) [SS07] suggested a method by modifying the convex hull approach to trace the boundary from airborne Lidar point clouds. Beginning with the left-most point p, the next point pN on the boundary is defined as the one holding the minimal clockwise

angle with respect to p within a neighborhood. This process is repeated from the point pN until the boundary is determined (see figure 3.1). This is a sound solution to trace the regular boundary line of the surface. However, measuring the angles in order to identify the smallest one around each point is a highly time-consuming computation.

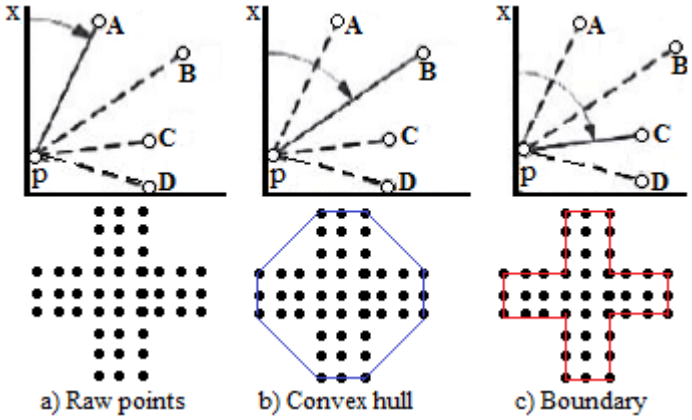

Figure 3.1: A method to extract a boundary in [SS07] (e.g. $\angle XpA$: a smallest angle)

Shen Wei (2008) [Wei08] designed an algorithm (based on the well-known "Alpha Shapes algorithm" of Edelsbrunner(1983) [EKS83]) to extract the boundary. Starting from any point p_1 within the point-set S, choose a point p_2 with $d(p_1, p_2) < 2\alpha$ (if such point exist), and draw a circle going through p_1 and p_2 with radius α (see figure 3.2a). The point p_2 is chosen if there are no other points inside the circle, and then p_1 and p_2 are defined as boundary points. They will thus be connected by a segment to obtain a boundary line. Subsequently, the process is repeated from p_2 (p_2 now becomes p_1 for the next point pair) until all points within S are checked. This method allows extracting both inner and outer boundaries of convex and concave polygon shapes, and leads to good results (see figure 3.2b). Nevertheless, the computations between each pair of points (p_1, p_2) and its neighboring points within S is repeated to find the boundary points and proves computationally costly.

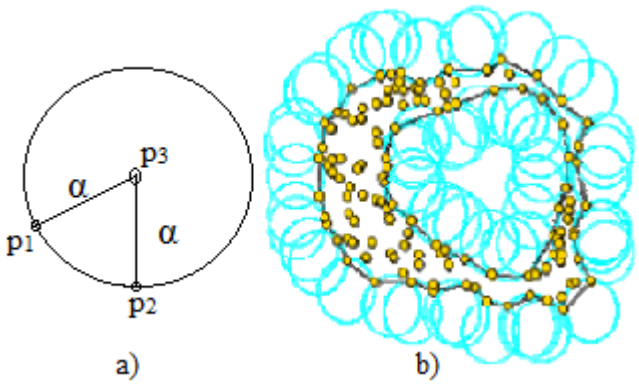

Figure 3.2: Method to extract a boundary of Shen Wei [Wei08]

A method is proposed by Xianfeng et al (2008) [XXFJ08] to trace the boundary from a point cloud Sp based on an edge ratio constrain. The main idea of the algorithm is to use the topological relationship between points, edges, triangles and other geometric features: length, angle, etc to analyze and determine the boundary of Sp. The convex hull S of Sp is computed by using a triangulation by an insertion algorithm. S allows defining an initial boundary (see figure 3.3a). S is then optimized by using one or two short edges to substitute the longer edge of each outer triangle based on an edge ratio constrain (see figure 3.3b). The process is iterated and is stopped when no line segment is longer than a given threshold. This algorithm produces a non-convex boundary. However, the calculation is repeated iteratively, to compare each edge of the triangle with a threshold line, leading to an important computing time.

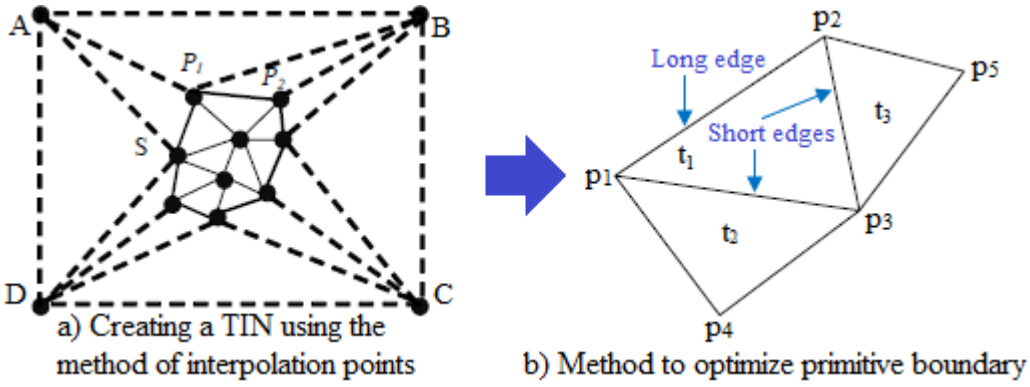

Figure 3.3: Method to extract a boundary of Xianfeng [XXFJ08]

Kalogerakis et al (2009) [KNSS09] presented an approach to determine the boundary points in 3D. At first, the minimum neighborhood for each point p_i in the initial data of point clouds is determined by finding the points closest to it, after projecting a 3D point cloud onto the local estimated tangent plane of p_i. Then, the closest points for each of six 60^o slices around p_i on this plane is taken into consideration. If there are no nearest points in two or more continuous slices around p_i, the point is recognized as a boundary point (see figure 3.4). The process is then repeated for each point of the point set, in order to find all the boundary points. This approach can be applied easily if the point clouds are distributed in the sparse areas or they are located on the side of a convex surface. However, in the case of dense data located on one side of a concave surface, such a method may fail: as any neighborhood of p_i may contain interior points, the empty slices criterion may fail and hence space has to be split with angles lesser than 60^o.
Boundary detection methods are also widely used in medical image processing. Sait et al (2011) [SKM11] described a method to detect the boundary of a set of points based on clustering. Beginning with the first point (called core point), a cluster is created as a sphere of a radius α. Next, the convex hull of this cluster is computed. The cluster is then iteratively expanded by repeating this process taking as core points those of the convex hull. Afterward, each of the convex hulls of core points is combined with the main body of the cluster. In principle, this operation corresponds to the union of two polyhedra. The

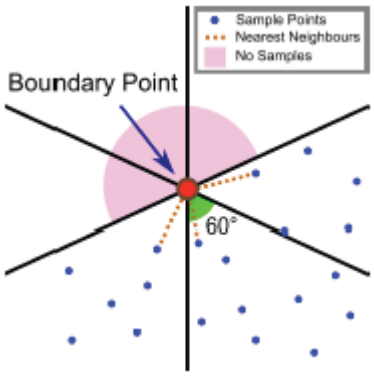

Figure 3.4: Method to extract the boundary points of Kalogerakis [KNSS09]

expansion and the combination of clusters will be stopped until no more point is found and the last boundary comprises all the outer points (see figure 3.5).

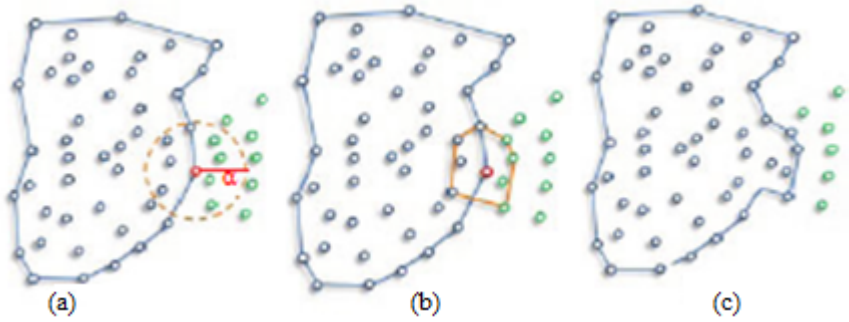

Figure 3.5: Method to extract a boundary of Sait [SKM11]

The above methods are suitable for scattered data but are time consuming. As we use voxels and their neighborhood information we can adapt these methods, to obtain faster computation.

3.2.2 Boundary simplification

Boundary simplification is an important step in surface simplification. Indeed, most surface simplification algorithms removing points or replacing clusters by representative vertices, have a shrinking effect on the surface. Therefore, detecting, simplifying, while at the same time preserving the boundary, can be considered as an essential advance over them. The different points of the boundary are considered as successive points of a polyline.
Different existing methods are available for the simplification of polylines. Let us mention the algorithm proposed by Douglas et al (1973) [DP73]. To process such polylines in 2D, the author initially connects the first point to the last point of the polyline $L(first, last)$. All the remaining points are then checked. If the perpendicular distance from each point to the line L is greater than a threshold, the farthest point F is chosen to add to simplification.

The next step consists in creating two new lines $L1(firstpoint, F)$ and $L2(F, lastpoint)$. The operation is then repeated recursively on these two lines until the number of lines remains stable (see figure 3.6).

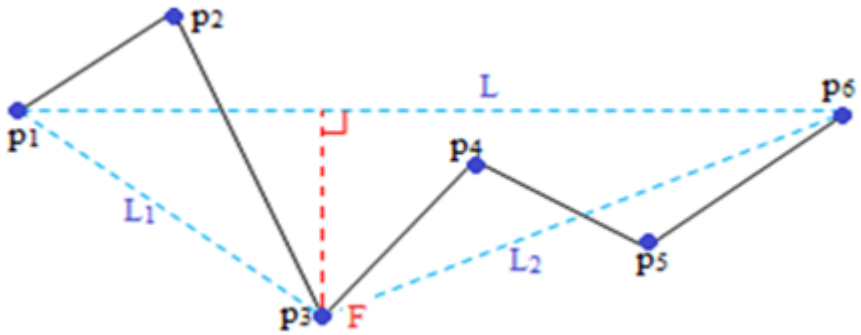

Figure 3.6: Method to simplify the boundary of Douglas [DP73]

In another algorithm, introduced by A.Garrido et al (1998) [GBGS98], the dominant point detection is based on a multiscale vision of boundaries. This algorithm has the same idea with the algorithm proposed by Meijers (2011) [Mei11]. Given three consecutive points which form a triangle $\triangle(p_{i-1}, p_i, p_{i+1})$, the weight w is the perpendicular distance from p_i to the line p_{i-1} and p_{i+1}. For each triangle T, if $w <$ threshold value, then T collapses and transforms from $\overline{p_{i-1}, p_i, p_{i+1}}$ to $\overline{p_{i-1}, p_{i+1}}$ (see figure 3.7).

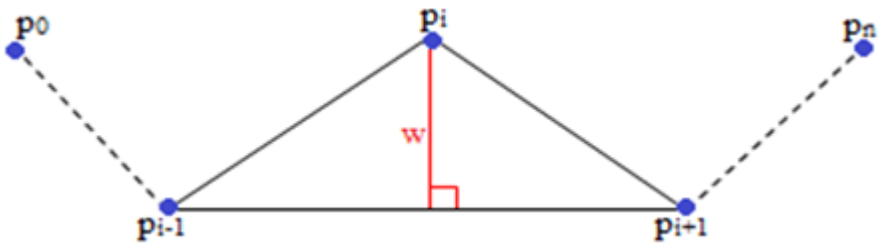

Figure 3.7: Method to simplify the boundary of Garrido and Meijers [GBGS98, Mei11]

These algorithms are implemented efficiently in regard to computing time and generally produce good results in application. However, if the number of consecutive points greater than three, are on the same line segments of the boundary, they are not completely simplified because the algorithm only process on each three points which form a triangle.

3.3 Methods for boundary extraction

3.3.1 Overview

As we mentioned in the introduction, the data are organized in a 3D volume, which allows determining easily the neighborhood of a point. In order to simplify such a surface without

any shrinking, it proves necessary to find, extract and simplify the boundary. Because the 3D point sets are sampled and originated from an elevation surface, the 3D cloud of points is first projected onto a natural 2D grid in x, y plane.

3.3.2 Definitions

3.3.2.1 Definition of a square neighborhood

Definition 3.1. *Let p be a cell of a 2D grid G, we define the k_square neighborhood of p as:*

$$SN_k(p) = \{p' \in G; \|p - p'\|_\infty \leq k\} \tag{3.1}$$

where $\|p - p'\|_\infty$ denotes the discrete infinite distance of R^2

Remark. *Given a point $p \in G$, we call k_ring neighborhood of p denoted by $RN_k(p)$ the set of p' such that $\|p - p'\|_\infty = k$. k_square neighborhood of p contains $(2k+1)^2$ points and its k_ring neighborhood contains $8k$ points. (see figure 3.8)*

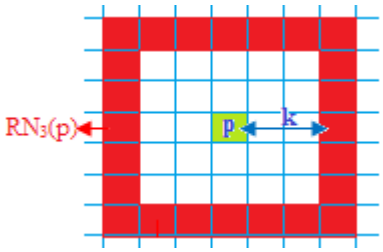

Figure 3.8: The k_ring neighborhood with $k = 3$.

Proof. As the side of the square $SN_k(p)$ is $2k+1$, we obviously have $|SN_k(p)| = (2k+1)^2$. Then, as $RN_k(p) = SN_k(p) - SN_{k-1}(p)$, we get $|RN_k(p)| = (2k+1)^2 - (2(k-1)+1)^2 = 8k$. □

We use the notations below to denote the elements in $SN_k(p)$:

- $E_k(p)$: the subset of empty cells in $SN_k(p)$.
- $NE_k(p)$: the subset of non-empty cells in $SN_k(p)$.
- $SN_k(p) = E_k(p) \cup NE_k(p)$

Different notions of boundary have been studied and precisely defined in fields of research such as differential geometry (for continuous surface studies) and topology [CM91]. The topological definition has also been extended to discrete surface (such as meshes) and has been widely used in this setting. Most boundary notions coincide with topological

definitions: a point belongs to a boundary if none of its neighborhoods are included inside the surface. In such a setting, boundary points therefore correspond either to the peripheral points of an open surface, or to the boundary of holes. In our case, as we are interested in identifying the periphery of an elevation surface in order to avoid a shrinking phenomena during simplification, we will define the concept of "exterior boundary", which will be computed according to the k_square neighborhood connectivity. It is important to recall that some points are missing, which requires specific processing.

3.3.2.2 Definition of different types of boundary

Depending on the computations we want to apply to the surface, we will use either the classical topological boundaries or our exterior boundary (see below for definition). Both contain boundary points belonging to S. Whereas the topological boundary can be used to compute and determine the boundary of holes in a surface, processing an exterior boundary line allows to avoid shrinking during the simplification of the surface. Let us first recall the classical notion of topological boundary.

Definition 3.2. *Topological boundary: A point p belongs to the topological boundary if its neighborhood (classically for 4 or 8-connectivity) contains at least one point outside S. (see figure 3.9)*

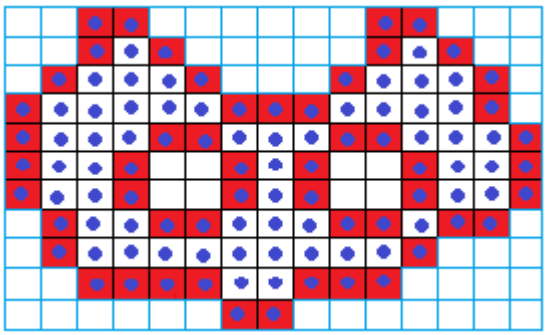

Figure 3.9: Topological boundaries of 8-connectivity

Let us now define the notion of exterior boundary.
Note: as in our case there is a one to one correspondence between points and pixels/voxels, in the sequel, we will freely speak about points or pixels/voxels. For instance, the definition of $SN_k - connexion$ of points is also valid for pixels (and conversely).

Definition 3.3. *Closed discrete $SN_k - curve$:*

- *Two points p_1 and p_2 are called SN_k-connected if $p_2 \in SN_k(p_1)$, or what is equivalent $p_1 \in SN_k(p_2)$.*
- *A closed discrete $SN_k - curve$ is a sequence of pixels where each pixel is $SN_k -$ connected with the previous and the next of the sequence.*

- *Let £ be a closed discrete SN_k − curve. The interior and exterior of £ can be defined by the classical approaches: either by means of solid angle or by the number of intersection with half-lines drawn from the various points.*

Definition 3.4. *Exterior boundary: An exterior boundary of S is a closed discrete SN_k − curve £ of points of S such that S is in the interior of £. We will denote this boundary by $EBL_k(S)$. (see figure 3.10)*

Remark. *Such an exterior boundary line is a compromise between the topological boundary (or boundary of the connected component) and the convex hull envelope, and is suitable when data are missing, as in our case. Let us also mention that such a boundary is not unique in general. Hence, different strategies can be defined in order to compute such a line.*

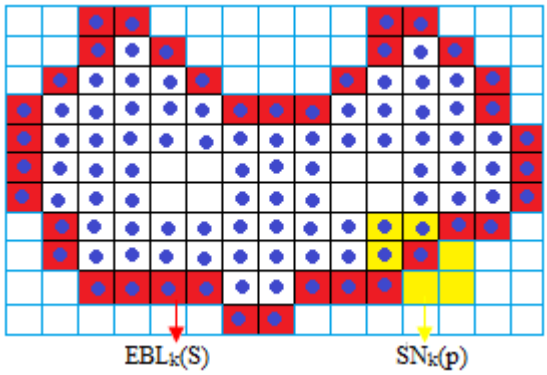

Figure 3.10: An exterior boundary with $k = 1$

3.3.3 Algorithms

Our algorithm is related to Sampath et al (2007) [SS07], but as our data are organized as pixels in a regular grid, we took an advantage of this structure: instead of computing angles, which is computationally costly, we use the pixels neighborhood information to save processing time. Moreover, our notion of exterior boundary provides a tool for extracting a simplified boundary. Therefore, we developed a method to extract such an exterior boundary starting from an initial boundary point of the surface. Our main objective was to develop an efficient algorithm and control time and space complexity as much as possible. In case the surface contains more than one SN_k connected component (called patch in the sequel), this algorithm can extract all exterior boundaries of these different connected components.
Our algorithm starts from a point located on the exterior boundary. This initial point, denoted by $pFirst$ in the sequel, is computed by taking the point of minimum x and y coordinate located on this boundary. Then, an exterior boundary is built point by point by iteratively computing next point via growth function. We introduced two different such

growth functions (based on so called "distance" or "clockwise" criteria).
Thus, a central issue in our algorithm is the definition of this growth functions. Many questions need to be carefully analyzed and answered in order to define it and determine next point on the exterior boundary: following which direction, under which conditions, following which rule? In this chapter, we suggest two alternate strategies. One of them takes the distance between two points into consideration; the other consists in attempting to reach the next point clock-wise in the shortest possible time. Both of them are computing an external boundary for SN_k (the border is followed in the clockwise direction, see figure 3.11).

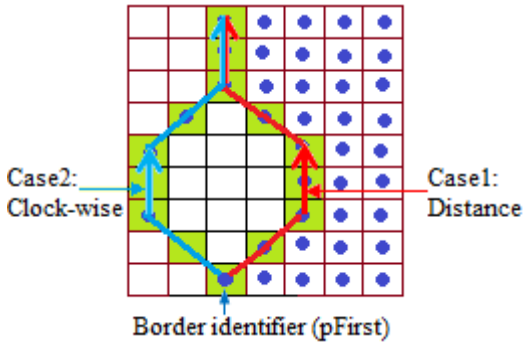

Figure 3.11: 2 cases to extract the exterior boundary

We used the notations below in our algorithm iteratively, and at any step:

- p: the current point.
- q: the point in $RN_i(p)$.
- $pPrev$: the point previously computed on the boundary.
- In the sequel, we will call half-line: a discrete half-line for 8-connectivity.
- $pPrev_k$: the intersection between $RN_k(p)$ and the half-line $(p, pPrev)$
- $pFirst$: the first boundary point used to extracting the boundary (explained previously).

3.3.3.1 Extracting boundary by distance growing

We define the first growth strategy, called distance criterion, which means that the priority is always given to the nearest point. The corresponding algorithm (Algorithm 3.1) is as follow:

Algorithm 3.1 GrowthDistance(p)

```
1: stop ← false, p = pFirst;
2: while !stop do
3:     find ← false; i=1;
4:     while i ≤ k and !find do
5:         //we enumerate RN_i(p) as depicted in figure 3.12.
6:         for any q ∈ RN_i(p) enumerated clockwise starting from pPrev_k do
7:             if one of the 4-neighborhood of q is empty then
8:                 pPrev ← p;
9:                 p ← q;
10:                find ← true;
11:            end if
12:        end for
13:    end while
14:    if p = pFirst then
15:        stop ← true;
16:    end if
17: end while
```

Remark. *At the beginning, we start from the pixel $p = pFirst$. Then an exterior boundary is built by iteratively computing next point p on the boundary. This pixel is the closest point of S encountered when cycling clockwise from $pPrev$. Therefore, this first version of our algorithm enumerates the successive ring-neighborhoods of $SN_k(p)$ starting from their intersection with the half-line $(p, pPrev)$. Therefore, this exterior boundary will be the one whose pixels are the closest one from the other. This process ends when p hits $pFirst$ (that is $p = pFirst$), which means that the exterior boundary of one connected component has been found. We apply the same method on each of them.*

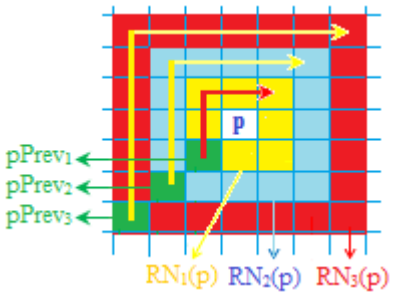

Figure 3.12: The growth algorithm based on distance.

The result of applying this algorithm is displayed on figure 3.13a. In this case, the boundary cannot connect to the outside point of the surface although the distance between the boundary points is less than or equal to k, because the priority is connecting to the first nearest distance. However, the resolution of the boundary is high.

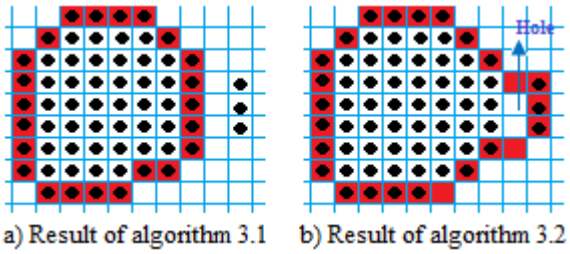

Figure 3.13: The results of two cases with $k = 2$

3.3.3.2 Extracting boundary by clockwise scanning

Our second strategy is called clockwise criterion. In this case, next point is chosen as the most clockwise point of S encountered when cycling around p starting from the direction $(p, pPrev)$. Intuitively, this point can be imagined as echo response in a radar scan. The corresponding algorithm (Algorithm 3.2) is as follow:

Algorithm 3.2 GrowthClockWise(p)

1: //We describe $SN_k(p)$ by enumerating the half-lines issued from p in a clockwise order starting from the direction $p, pPrev$ (see figure 3.14).
2: //At each step, $pPrev_k$ is the intersection between $RN_k(p)$.
3: stop $\leftarrow$ false, $p = pFirst$;
4: **while** !stop **do**
5: compute $pPrev_k$, the intersection between the half-line $(p, pPrev)$ and $SN_k(p)$;
6: find $\leftarrow$ false;
7: **for** any $q \in RN_k(p)$ enumerated starting from $pPrev_k$ and !find **do**
8: **for** i=1 to k **do**
9: $q_i \leftarrow$ intersection between $RN_i(p)$ and discrete segment (p, q).
10: **if** one of the 4-neighborhood of q_i is empty **then**
11: $pPrev \leftarrow p$;
12: $p \leftarrow q_i$;
13: find $\leftarrow$ true;
14: **end if**
15: **end for**
16: **end for**
17: **if** $p = pFirst$ **then**
18: stop $\leftarrow$ true;
19: **end if**
20: **end while**

Remark. *In this case, the same as case 3.3.3.1, we also start from the pixel $p = pFirst$. Then, the next step to find next point differently: we compute the first point of S, starting from $pPrev$ in a clockwise order; then the next point is found starting from the direction*

from p to pPrev (see figure 3.14) following a clockwise order. Then $RN_k(p)$ is enumerated circularly starting from $pPrev_k$, the intersection between the half-line $(p, pPrev)$ and $SN_k(p)$ (we denote by q the successive points of $RN_k(p)$). We determine the position of q on $RN_k(p)$ based on a conversion between two coordinate systems for $RN_k(p)$ (see 3.4 Implementation). Therefore, the next point is the first point met on the successive (p, q) half-line.

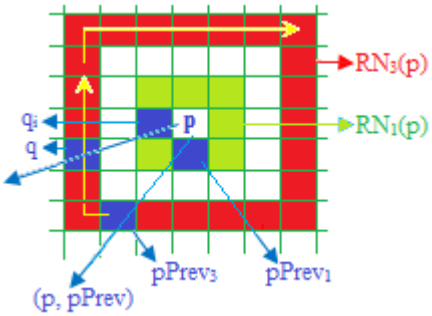

Figure 3.14: The growth algorithm based on clock-wise.

This algorithm computes an hybrid boundary: connected components of the surface normally separated in the classical 4 or 8-connectivity, can be simultaneously outlined if the gap between them is less than k. And hence, our boundary is intrinsically a simplified contour of the surface adapted to sparse data and possible (small) holes (see figure 3.13b).

3.3.4 Implementation

Let us give some hints on the implementation of the proposed algorithms. We have to enumerate the ring-neighborhoods $RN_k(p)$ starting from a point $pPrev_k$ in a clockwise order. We build a formula to compute the cartesian coordinates of q based on figure 3.15 as below:

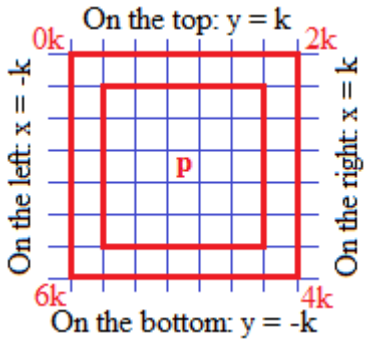

Figure 3.15: Compute the cartesian coordinates of q.

We handled this problem by a double coordinate system: cartesian on the one hand, and a cyclic numbering of $RN_k(p)$ together with the appropriate conversion functions. Besides,

the cartesian coordinates of q denoted by (x_q, y_q) are also calculated based on the cartesian coordinates of $pPrev$, denoted by (x_{pPrev}, y_{pPrev}) and the parameter k following the formula below:

$$(x_q, y_q) = \left\lfloor (x_{pPrev}, y_{pPrev}) \times \frac{k}{\|(x_{pPrev}, y_{pPrev})\|_\infty} \right\rfloor \tag{3.2}$$

And similarly:

$$(x_{q_i}, y_{q_i}) = \left\lfloor (x_q, y_q) \times \frac{m}{k} \right\rfloor ; with\, m \in \{1..k\} \tag{3.3}$$

3.4 Methods for boundary simplification

3.4.1 Background and Definitions

In this section, we introduce our method for simplifying the boundary. The exterior boundary is a closed contour ($p = pFirst$). Therefore, we cannot apply the method introduced in [DP73] to simplify a polyline directly. However, starting from the ideas of line and the polyline simplification, together with the methods presented in [GBGS98, SC06, Mei11], we combined them with our case to simplify the 3D boundary.
In general, we have a closed contour boundary extracted from a 2D set of points (x_i, y_i). As our surfaces are elevation surfaces, actually, each such point corresponds to a single 3D point (x_i, y_i, z_i). Hence, there is a one to one correspondence between the 2D boundary previously computed and the 3D boundary obtained by restoring z coordinates. We build an algorithm to simplify the 3D boundary as below:
At first, we detect the line segments in the boundary by computing the coordinates of points and the slope between points in the boundary. Then we apply the polyline simplification method on each line segment. Actually, we combine two steps: a trivial approach of line segment in 2D, and a polyline simplification method on 3D boundary.

3.4.1.1 Line segment extraction

This step is a rough approach based on 2D in order to determine the line segments in the boundary line for the next 3D polyline simplification step. Let $\{p_1, .., p_n\}$ be the set of boundary points (with $p_1 = p_n$); we denote by (x_i, y_i) the coordinates of p_i. For each line segment, we have N_{ps} representing a number of points in this segment; in the sequel, we will consider pairs of points at a distance of the line segment. For any i ∈ {1,..,n}, let SL_i be the slope of the sub segment $[p_i, p_{i+1}]$ in the line segment, that is:

$$SL_i = (y_{i+1} - y_i)/(x_{i+1} - x_i) \tag{3.4}$$

Sequentially, for any point p_i in the boundary, if they have a common coordinate, they are added to the same line segment. If not, we will check the slope SL_i of each pair of points in a sequence. If they have the same value, then they are also added to the same line segment. In fact, it is very easy to see that if points are aligned in 2D (see figure 3.16a), but they

are not necessarily in 3D (there can even be strong slopes, see figure 3.16b). Therefore, we based on these line segments (in 2D) to simplify them (in 3D) in the next step.

3.4.1.2 3D polyline simplification

Our boundary simplification (3D) is described as follows. For each line segment in the boundary, we compute the perpendicular distance H from each point p_i to the line L $\overline{p_{i-1}, p_{i+1}}$, which form a triangle $\triangle(p_{i-1},p_i,p_{i+1})$. If H is less than a threshold value t, then p_i will be removed. The next triangle is made up from $\triangle(p_{i+1}, p_{i+2}, p_{i+3})$ (see figure 3.16); Else, the next triangle starts from $\triangle(p_i, p_{i+1}, p_{i+2})$, etc. We compute H, L and A(the area of triangle) according to the formula of the distance between the 2D and the 3D points.

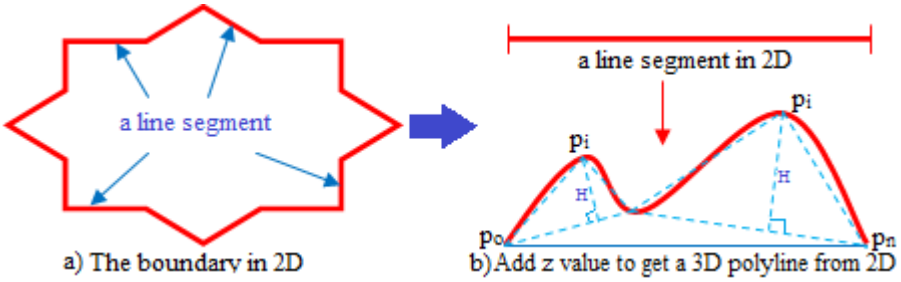

Figure 3.16: Simplify the 3D boundary

3.4.2 Algorithm

We defined an algorithm to simplify on each 3D line segment of the 3D boundary. The corresponding algorithm (Algorithm 3.3) is as follow:

Algorithm 3.3 SimplifyBoundary()

```
1: Let N_ps: the number of p_i in each line segment.
2: for each line segment do
3:     //we keep the first and the last point.
4:     for any p_i from 1 to N_ps − 1 do
5:         compute the perpendicular H;
6:         if H < threshold value then
7:             delete p_i;
8:         end if
9:     end for
10: end for
```

Remark. *The processing to simplify the 3D boundary is iteratively repeated on each line segment of these boundary as we introduced in the 3.4.1.2. During the computing and the removing of p_i, the first and the last point of each line segment are kept. Therefore, after simplification, the shape of the boundary is preserved.*

3.5 Results

3.5.1 Boundary Extraction

In this section, we present a number of results to illustrate the algorithm to extract an exterior boundary. Figure 3.20a shows the result of capturing an exterior boundary with $k = 1$ (case 3.3.3.1). We use the same data file and apply Algorithm 3.2 to get an exterior boundary with $k = 3$ (see figure 3.20b). In this case, a hole is created and the resolution of the boundary is lower than when $k = 1$. We used the same data file and applied this algorithm to get an exterior boundary with different values of k (see figure 3.22 and 3.25). Figure 3.22a: an exterior boundary with $k = 1$; Figure 3.22b: an exterior boundary with $k = 2$; Figure 3.22c: an exterior boundary with $k = 3$. Figure 21 is an exterior boundary ($k = 1$) of a geological surface. Figure 3.23(a) shown an exterior boundary with $k = 1$, extracted from the surface of 15626 points; processing time in 18 ms.
In our method, the shape of the boundary on the convex parts is not affected by the values of k, but it has affected on the concave parts of the boundary. During the boundary extracting process, the next point (on the boundary) is always scanned in a distance k from the current point p starting from the previous point $pPrev$ and following the clockwise direction. Therefore, the resolution of the boundary on a convex part is very high with any values of k (see figures 3.25 and 3.27). On the contrary, the resolution of the boundary is higher if k close to 1 (highest if $k = 1$, see figures 3.21 and 3.23a) and lower if k far from 1 (especially on the concave parts, see figure 3.27: a, b, c, d). If we choose k greater than 1, some holes could be created on the surface (the higher values of k, the bigger holes are created, see figures: 3.20b and 3.25). For this reason, our method allows to obtain different results of the boundary depending on k.
The computing time for extracting the boundary is mostly depending on the value of k. However, in some cases it also depends on the shape of the surfaces (see figure 3.18 for an illustration). As we mentioned above, the shape of the boundary does not change on the convex part of the surface. Therefore, the number of boundary points has reduced a little if we increase k (i.e. on the concave parts of the boundary). The simplification rate of the boundary points is mostly depending on the value of t (threshold). This point will be detailed in the next section.

3.5.2 Boundary simplification

The results are presented on figures 3.23, 3.24 and 3.26. They depend on the choice of the threshold value used in the Algorithm 3.3: if we chose a high value, a lot of points are removed and the resolution of the boundary after simplification will be low. Otherwise, the resolution of the boundary will be higher. Besides, the rate of reduction of points also depends on the value of threshold t. On figure 3.23(b), the number of boundary points has been reduced from 1025 to 660 points, approximate 35%; almost in the same proportion as in figure 3.24(b).
In this method, the boundary shape of the surface and the simplification rate of the bound-

ary mostly depends on the value of t. The threshold t is high or low depending on the choice of the user. For example, in figure 3.23 the initial curvature of boundary is low, so we chose $t \leq 0.4$; in figure 3.24, the initial curvature of boundary is higher and therefore we chose t with a higher value. If we increase t, the simplification rate of the boundary is high and a large number of boundary points is simplified. Figure 3.26 is an illustration: we first chose $t \leq 0.3$, the simplification rate is 63%; we then increased $t \leq 0.5$, the simplification rate rises 90% and the number of boundary points decreased from 1024 to 106 points. We have tested on many surfaces by using the different values of t, the obtained results of simplification rate are detailed in figure 3.19. However, the most important aspect of our solution is that the original shape of the boundary did not change (see figure 3.24).

3.6 Discussion and evaluation

Let us give some elements on the efficiency of our boundary extraction approach compared to existing algorithms (we will first study their respective complexities).
In the method of Sampath [SS07]: given a boundary point p, next point y on the boundary is computed as the neighbor of p of minimal angle $\angle xpy$. Therefore, each step of this method involves the computation of the neighbors of p (located within a sphere centered at p) together with their angular slope with respect to the (px) line. Hence, the total complexity of this approach for a N points boundary is $N \times k$ angle computations, where k is the average number of neighbors (see figure 3.17a).

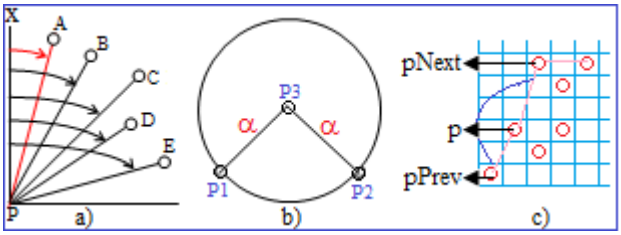

Figure 3.17: Comparison of computing strategies between the methods: Sampath (a), ShenWei (b) and Our method (c).

In the case of ShenWei method [Wei08] (based on alpha-shapes), the method tests in the worst case, $M \times (M-1)$ pairs of points, where M is the number of points of the surface S, that is $M = O(N^2)$ with the previous notations (see section 3.2: Related works for more details). For all of these pairs (p_1, p_2), the center p_3 of the circle of radius α (see figure 3.17b) and leaning on (p_1, p_2) must be computed and all points belonging to this circle must be detected, which leads to costly computations. In our method: for each boundary point p, next point $(pNext)$ is computed by enumerating the k square neighborhood clockwise starting from the direction $(p, pPrev)$, see figure 3.17c (see figure 3.14 more detail). Hence, the complexity of this step is at most $O(k^2)$; but as we stop the enumeration as soon as a neighbor is encountered, the average case is far lower. As a consequence, the complexity of our method is only $O(N \times k^2)$ where N is the number of boundary points. Therefore,

with respect to both previous methods, our approach proves more efficient. Figure 3.18 illustrates the computing time required by our method for numeric results. Figure 3.19 shows the boundary simplification rate of the surfaces.

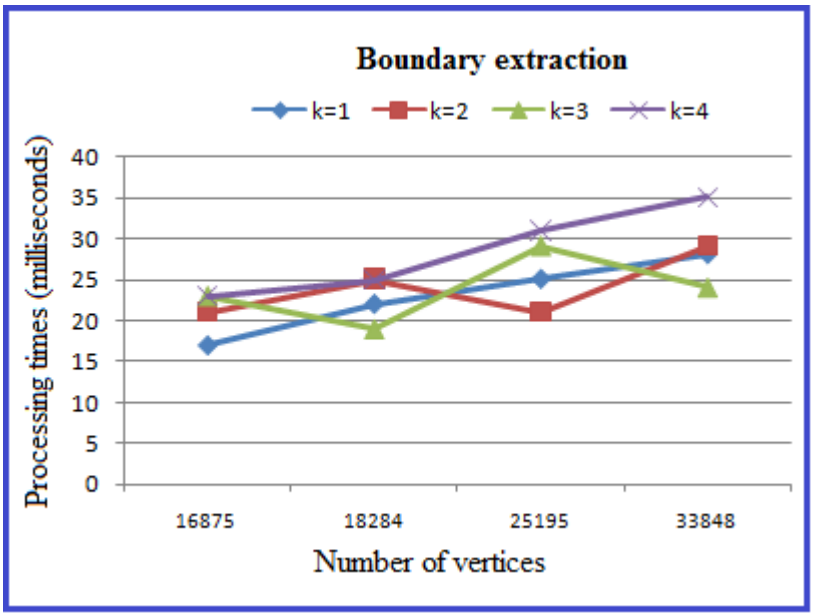

Figure 3.18: The processing time of boundary extraction usually depends on the parameter k, although sometimes it may be affected by the shape of the surface.

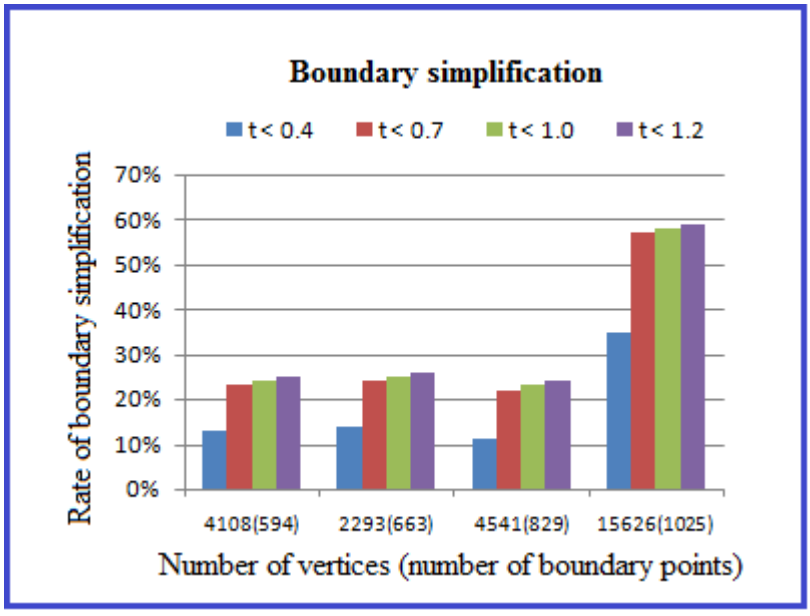

Figure 3.19: Rate of boundary simplification with different values of threshold t.

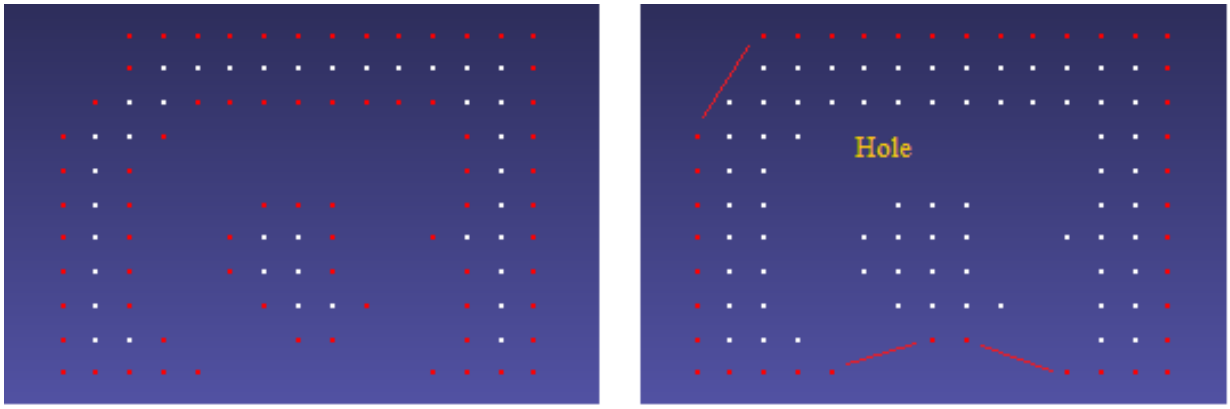

Figure 3.20: Extracting an exterior boundary of the surface with $k = 1$ and $k = 3$. The shape of the boundary does not change on the convex parts of the surface but created a hole.

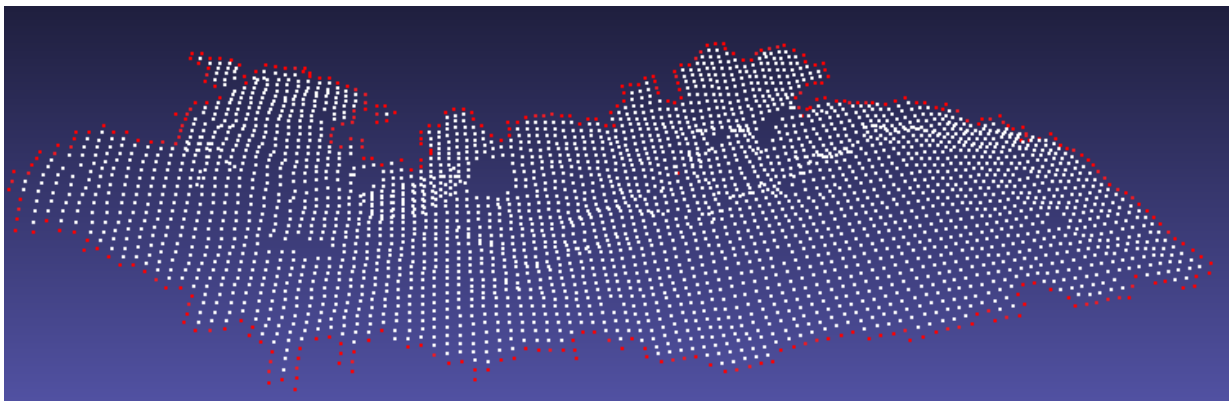

Figure 3.21: The resolution of the exterior boundary of a geological surface is highest with $k = 1$.

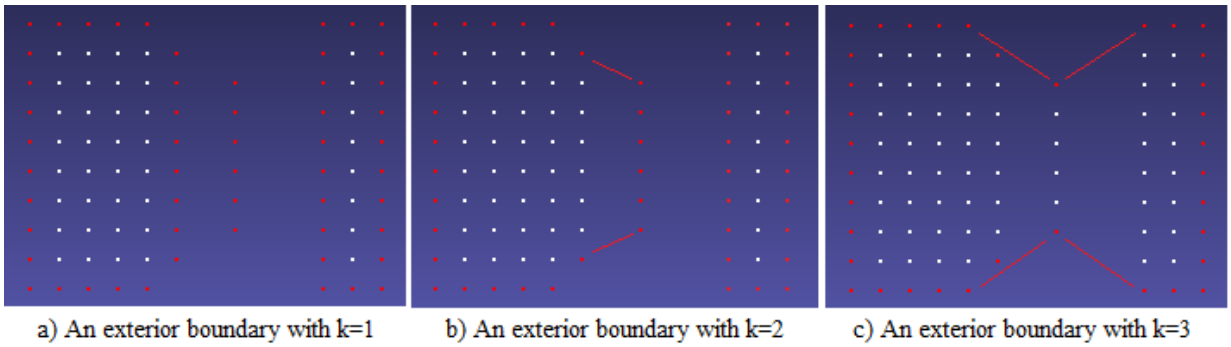

Figure 3.22: Extract an exterior boundary of the surface with different values of k. The shape of the boundary has changed on the concave parts of the surface.

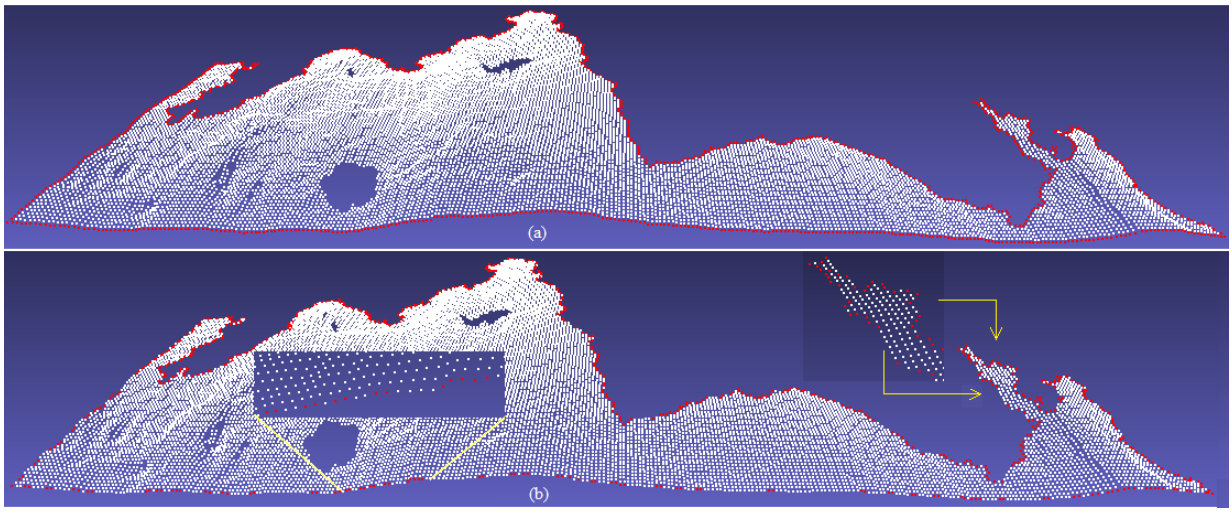

Figure 3.23: Exterior boundary in the case of a geological surface: (a) the original 15626 surface points and the extracted boundary 1025 points with $k = 1$; processing time: 18 ms. (b) Simplified boundary with threshold $t < 0.4$; number of boundary points: 660; rate of reduction: 35%; the red color points are boundary points; the white color points (on the boundary) are simplified

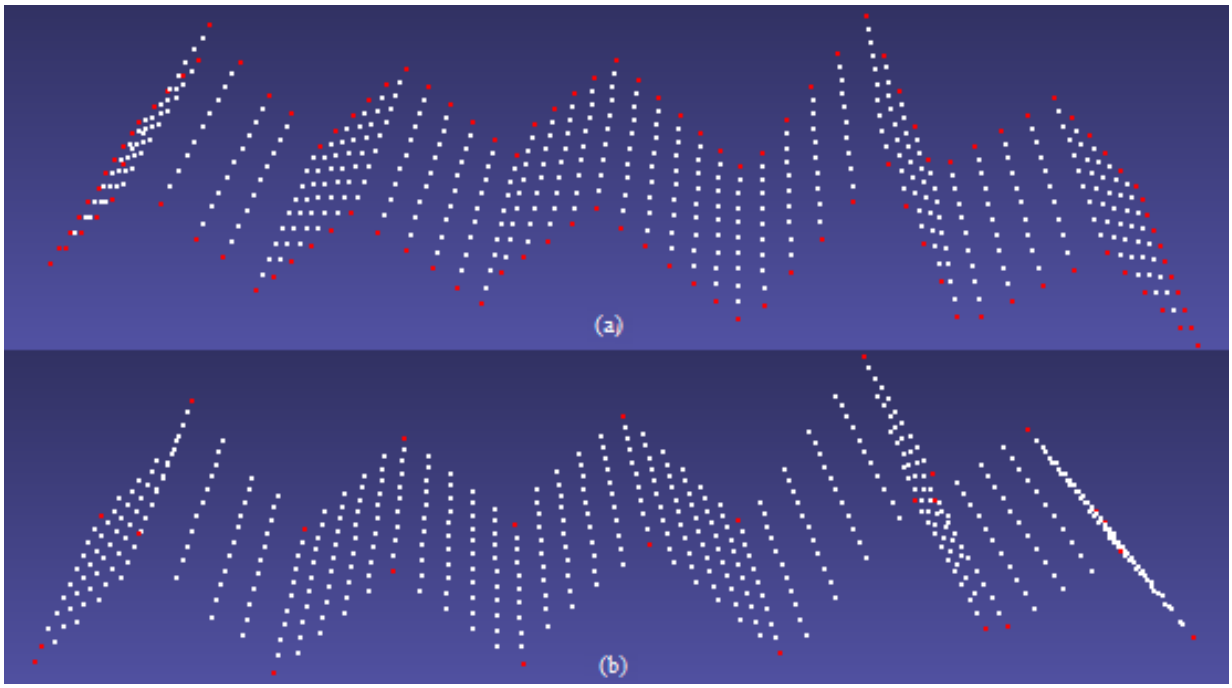

Figure 3.24: Before (a) and after (b) simplification; the boundary points reduce from 118 to 24; rate: 80%, while preserving the initial shape of this boundary (i.e. the characteristic points are preserved).

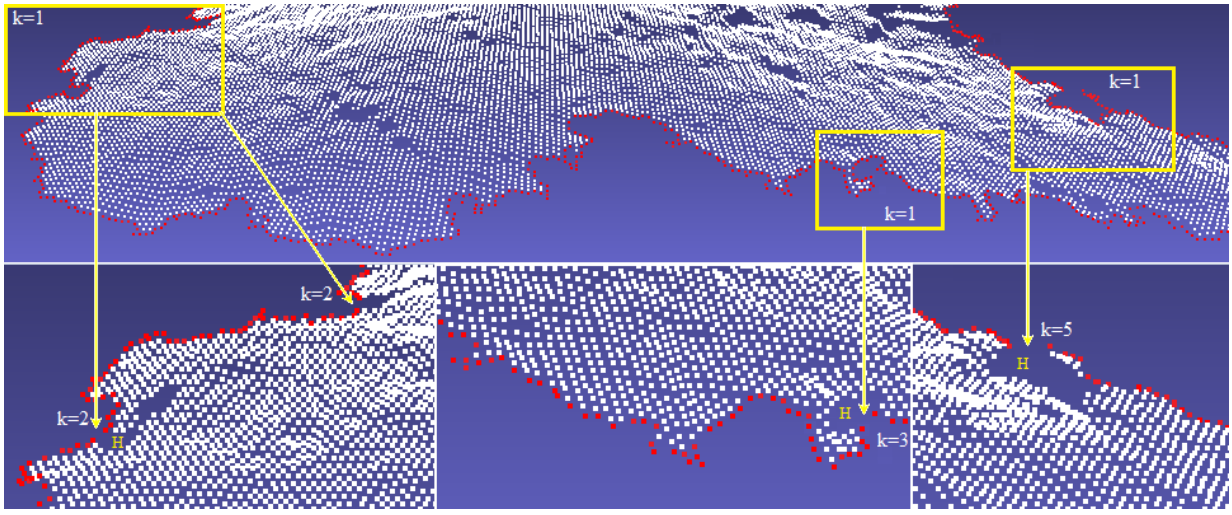

Figure 3.25: An exterior boundary of a geological surface after extracting with many values of k (if $k = 1$, the resolution of the boundary is high (highest); otherwise, the resolution of the boundary is low and some small holes H are created).

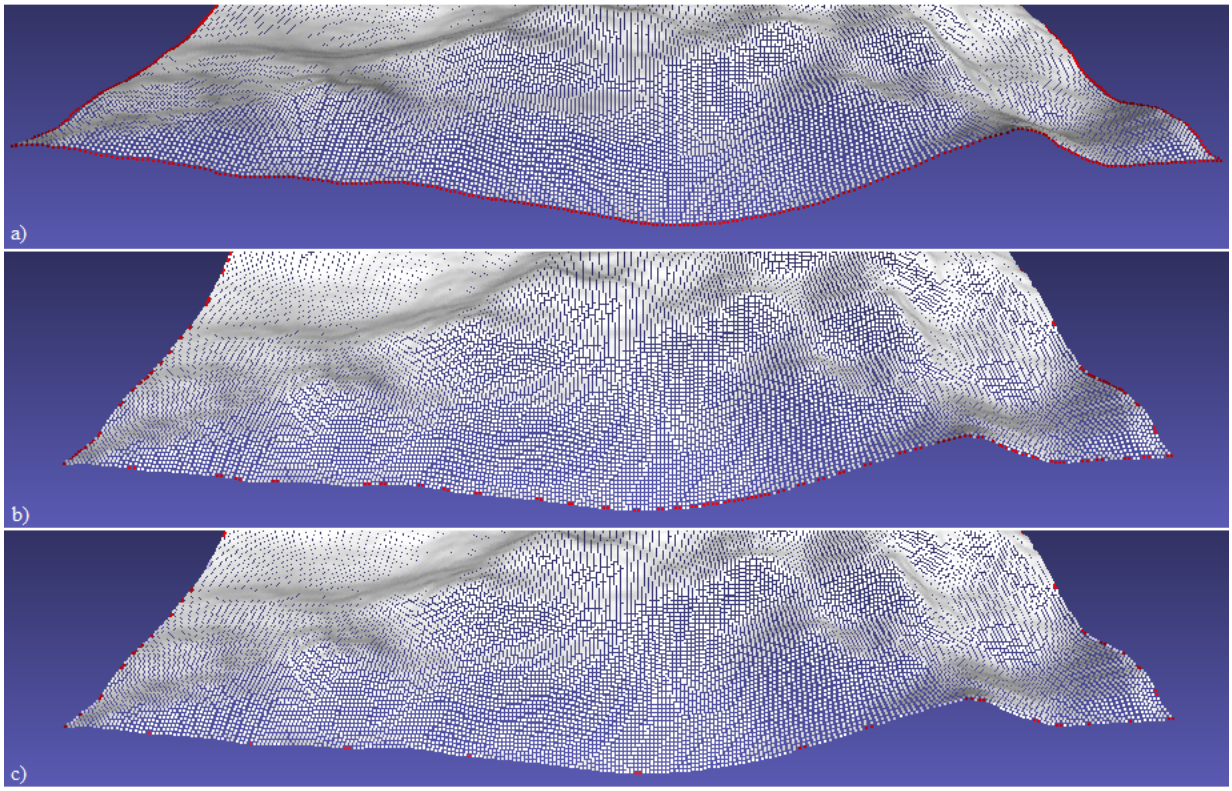

Figure 3.26: An exterior boundary of a geological surface after simplifying with many values of t: a) the original boundary (boundary points: 1024); b) after simplifying with $t = 0.3$ (boundary points: 381, simplification rate: 63%); c) after simplifying with $t = 0.5$ (boundary points: 106, simplification rate: 90%).

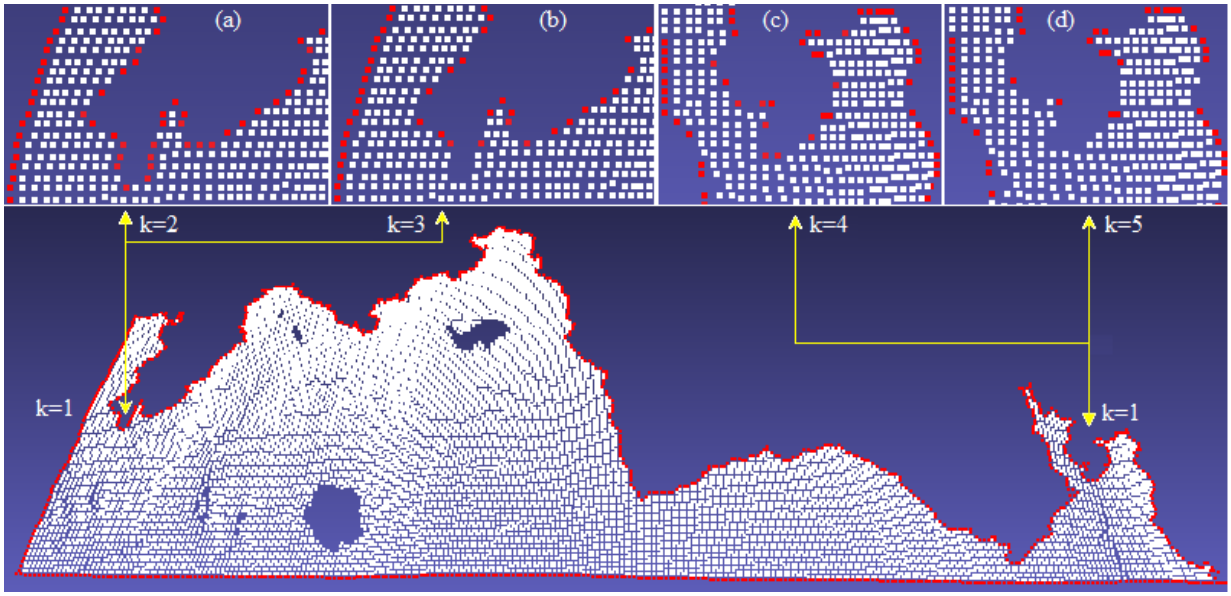

Figure 3.27: The shape of the boundary has been changed on the concave parts after extracting the boundary using many values of k. The resolution of the boundary is inversely proportional to the value of k: the smaller the value of k, the higher the resolution of the boundary; and in contrast, the more increasing values of k, the more rough boundaries we obtain.

3.7 Conclusion

In this chapter, we presented a new method to extract and simplify the boundary of an elevation surface defined into a 3D volume. This work was initially related to geological surface modeling, which leads to handling (and hence simplifying) mass of data where many information are missing, leading to a sparse 3D volume. Nevertheless, our approach is generic and only depends on the particular structure of data, which allows us to propose efficient algorithms to extract and simplify the boundary, and also to generalize the classical boundary notions. By means of the SN_k-connections we introduced, we are able to define a notion of contour (exterior boundary) outling the surface continuously despite possible small holes (that often occur in sparse or noisy data). Our method and obtained results have been published in [NBD12, Ngu12].
In the next chapter (chapter four), we continue to simplify the surface of data points inside the boundary, process which can be done without shrinking.

Chapter 4

Surface Simplification

Contents

4.1 Introduction 39
4.2 Related work 40
4.2.1 Simplification of triangular meshes 40
4.2.2 Simplification of point clouds 41
4.3 Method for simplifying the inside of a surface 44
4.3.1 Rough simplification 44
4.3.2 Elaborate simplification 45
4.4 Results 52
4.5 Discussion and evaluation 62
4.6 Conclusion 62

4.1 Introduction

Simplification of a 3D point cloud belonging to a surface is an important step in geometric modeling and surface processing. The purpose of surface simplification of a 3D point cloud is to reduce the number of points, save the memory, improve the effect of computation and optimize the processing of the geometric model. During simplification, the original shape of the surface must be kept, without shrinking or deformations.
Nowadays, the modern 3D acquisition and modeling technology allow producing a large amount of point samples from real-world objects. Different existing researches (and especially for meshes) are available for processing of the continuous surfaces, but the case of 3D point clouds simplification remains a challenging issue.
As we mentioned in the previous chapters, our problem originates in the questions of processing large 3D point clouds issued from a seismic data (themselves extracted from a 3D

sparse volume [Ver09]). The seismic acquisition does not permit to measure all the points in the 3D volume that explain the fact that the 3D volume is sparse. The 3D points are actually stored in a voxel structure in this volume (each voxel is considered as a 3D point, and has three real coordinates xyz), hence implicitly the 3D volume contains neighboring information even in a sparse context.
Most existing approaches have a common drawback: in the case of open surfaces (that is surfaces with boundaries), simplification induces a shrinking of the surface. Hence, in order to preserve the initial shape, our approach starts by an extraction and simplification step of the boundary. In chapter three, we have proposed a method for extracting and simplifying the boundary of a surface [NBD12, Ngu12]. In this chapter, we continue to process the surface by introducing a new method for simplifying the inside of this surface. To handle potentially huge clouds, our method consists of two steps: an optional initial rough simplification (basically designed to adjust the sampling rate) followed by a more elaborated simplification step. The point clouds are first projected onto a 2D grid in x, y plane to process with the first step, while the second step is directly processed in the 3D grid.
The contents in this chapter are organized as follows: firstly, we present the work related to the surface simplification of a 3D point cloud as well as a triangular mesh. Secondly, we detail our method which includes problem analysis, building the criteria and implementing the algorithms. Thirdly, the results and evaluation of our method are presented in the next sections.

4.2 Related work

The different existing methods for simplifying the surfaces which have been studied and developed are not only applied to simplify the surface of 3D point clouds, but also applied to simplify the surface of triangular mesh. Most of them (simplification methods) are dedicated to simplification of meshes [Hop94, GH97, CG09, OVBP11] and more recently (with the emergence of scanning devices), new algorithm appeared to directly simplify point clouds [PGK02, SLK05, FD07, LT07, LJ08, SF09, MWZ10, ZG10]. Among them, PCA (Principle Components Analysis) is a popular tool, a well known method [Jol02] that can be used to simplify the surface of 3D point clouds [MVF04, Tra08, YsW06, BTD07, Bel08].

4.2.1 Simplification of triangular meshes

A well known method for simplifying a mesh model is presented by Garland (1999) [Gar99]. The algorithm uses iterative contractions of vertex pairs to simplify models and maintains surface error approximations using quadric matrices. Starting from the initial model M1, a sequence of pair contraction (v_1, v_2) is applied by computing the optimal contraction target $\bar{v}$. The edge e_{v_1,v_2} will be contracted to a new position $\bar{v}$ if the quadratic error metric is lesser than a threshold. The process is repeated until the simplification goals are satisfying. The last model M2 approximates M1.

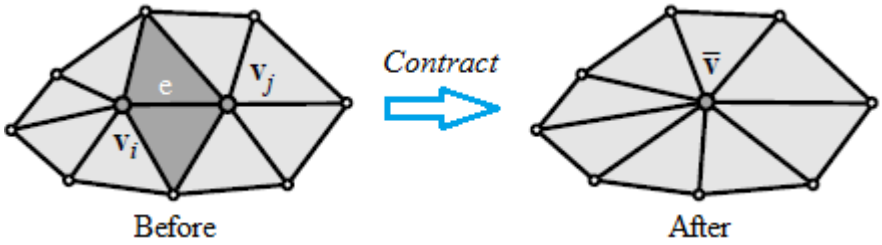

Figure 4.1: Simplification of a triangular mesh by edge-collapse [Gar99]

Figure 4.1 is an illustration (one vertex and two faces are removed from the model). In contrast (see figure 4.2), contracting of a non-edge pair will remove one vertex and join previously unconnected regions of the surface. By contracting arbitrary vertex pairs (not just edges), this is also an advantage to support non-manifold surfaces.

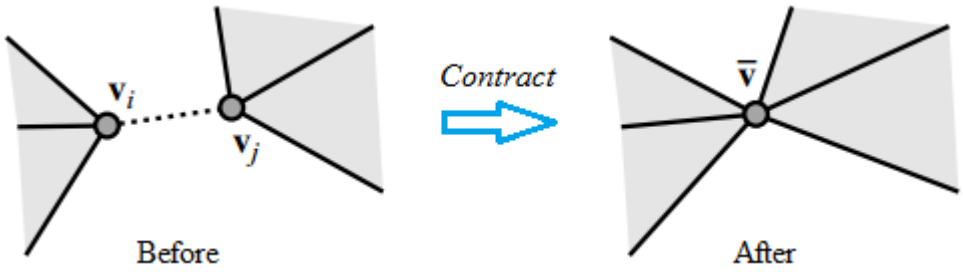

Figure 4.2: Non-edge pair (v_i, v_j) is contracted, joining previous unconnected area [Gar99]

In order to preserve the shape of the surface and optimize the placement of vertices after contraction, the quadric error matrices are also used to track the approximate error of the model. This method is time and memory demanding, but it avoids distortion of the original shape. However, evaluation of the quadratic error metric is closely related to the mesh structure (and to the face neighborhoods). Hence, it cannot easily be adapted in our setting.

4.2.2 Simplification of point clouds

In order to simplify a surface of 3D point clouds, many existing approaches are based on a preliminary clustering of points, then each cluster will be replaced by one representative point. Pauly et al (2002) [PGK02] introduced, analyzed, compared and implemented a number of methods to simplify the surface of 3D point clouds. One of these methods is called “Clustering" (see figure 4.3). The surface of 3D point clouds is clustered by splitting it into a subset of points; then, replace all points in each cluster by one representative point. This region-growing is terminated when the size of the cluster reaches the maximum bound. This method leads to simplifying the surface effectively. However, each cluster is a sphere with a radius α on the surface. Therefore, the points outside these clusters are not simplified completely after the iterative processing.

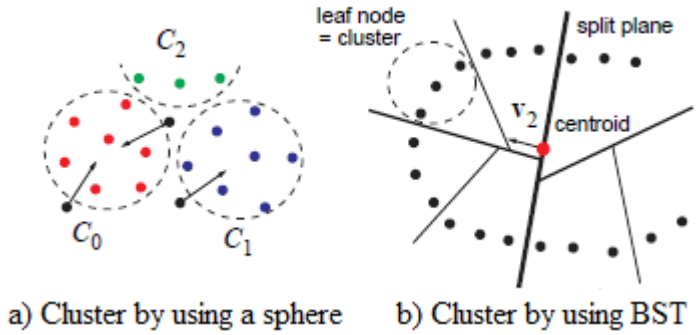

Figure 4.3: Simplification of a 3D point cloud by using cluster [PGK02]

Boris et al (2004) [MVF04] proposed a method to reconstruct and smooth a surface from noisy point clouds. At first, the authors smoothed the original point clouds to reduce the noisy points by using a robust projection procedure, while keeping the shape of the surface. During the next step, data of 3D point clouds are clustered by partitioning into a subset of clusters based on BSP (Binary Spacing Partition) tree. Then, they applied PCA to analyze, reduce the size of the original points, and determine a representative point for each cluster. In the next step, a triangular surface is obtained from the representative points of each cluster to obtain a rough surface which approximates the original surface. This rough surface is finally refined to get an optimal one. This is a complete method for surface reconstruction of a point cloud. However, the computing is complex during projecting, clustering, reducing, meshing and refining the point clouds, leading to a computation heavy and costly.
Normally, to simplify the surface of 3D point clouds, we need to compute the neighboring points for each point. The problem is how to determine the neighboring points in a local region of the surface to remove. Y.J Zhang et al (2010) [ZG10] proposed a way to define the nearest neighbouring points by using a cylinder. The points are dropped into a bounding cylinder based on the specified threshold (the radius of the cylinder); then, they are projected on the line as its center axis to simplify the points inside (see figure 4.4). This idea is used in [PGK02]. For each iterative step, there are still points outside of the cylinder; they are located between these cylinders; therefore, they are not simplified completely.
Brendan J. Frey et al (2007) [FD07] presented a method (the “affinity propagation") to cluster by passing messages between data points. This method measures the similarity of each point-pair of the input data points. Each point in a point set is assigned as a node of a network, the real-valued messages are exchanged between data points (nodes) along the edge of the network until a high-quality set of exemplars corresponds to the cluster which gradually emerge (see figure 4.5). During sending messages between the points, the messages can be combined at any state to decide which points are exemplar, and which exemplar the point belongs to. The clustering process will be terminated when all points

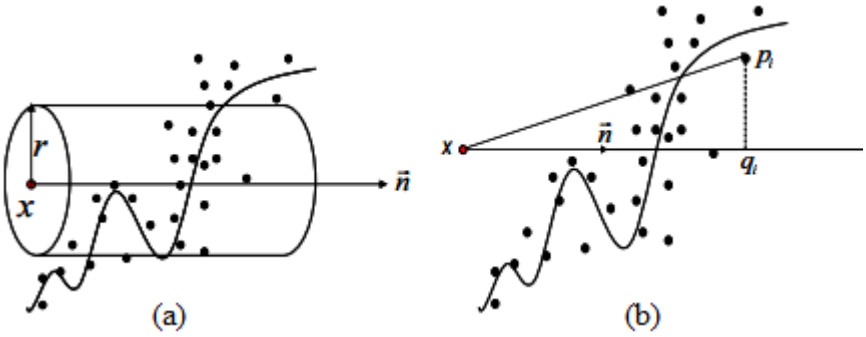

Figure 4.4: Determining of neighboring points [ZG10]

are considered and belong to their clusters. However, the cost of computing in this method is expensive because the transmission process between the points is computed recursively.

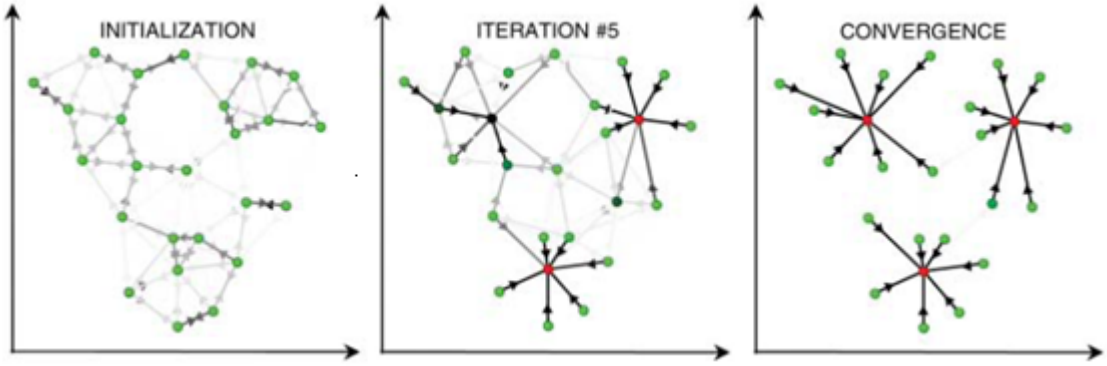

Figure 4.5: Clustering by passing messages between data points [FD07]

Jae-Young et al (2005) [SLK05] and Tamal K.Dey et al (2011) [DDW11] introduced a method by using an octree partitioning to divide the point clouds into a small subset, then process on each subset as a node of an octree on 3D space and quadtree on the 2D grid. At first, a rood node of a point cloud is divided into four in 2D or eight in 3D. Then, the child nodes are recursively divided until satisfying the condition of the threshold. After that, each node can be considered as a point during the simplification.
Rixio Morales et al (2010) [MWZ10] suggested a method to smooth and decimate the points from an unstructured point cloud based on the radial based function RBF. The method computed the distance between a central point and the nearest neighboring points in each local point set of the surface by using kd-tree nearest neighbor method. Starting from a seed point p_i, the neighboring points p_n are calculated by an Euclidean distance $\|p_i - p_n\|$ to determine the radius r. All points within r are mapped from a 3D point set to the 2D space; the point set components are mapped into each axis plane on each square matrix MxMx3 in domain N_{ix}, N_{iy}, N_{iz}. The next step is using a convolution Gaussian Kernels function $C = M \otimes G\left(\mu, \sigma^{d(k)}\right)$ for each axis N_{ij} to smooth and estimate the new

center point in each component $p'_{x,y,z}$. At the end, the 3D point sets are smoothed and simplified according to the local surface features (see figure 4.6).

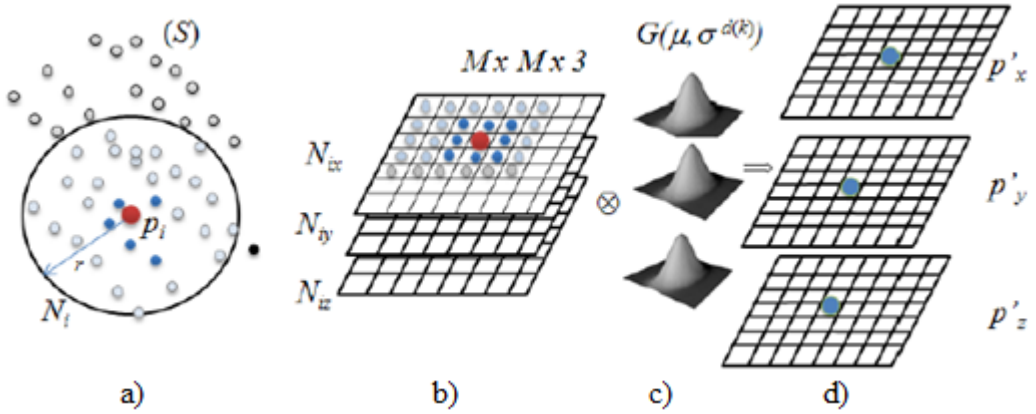

Figure 4.6: Decimation of 3D point clouds by using RBF, K-NN kernel [MWZ10]

As we have described and analyzed, the above methods are suitable for dispersive data or unorganized point clouds but lead to an expensive computation. In our work, we take advantage on the structure of voxels and their neighborhood information. We can adapt these methods to simplify the surface efficiently; preserve the shape and point distribution of the surface.

4.3 Method for simplifying the inside of a surface

Our method for simplifying the surface of 3D point clouds is based on clustering. Instead of using a sphere, a cylinder or merging of 3D points by sending messages as presented in [PGK02, FD07, ZG10], we use adaptive square cells to simplify the surface. The method is described based on both steps presented in the following section.

4.3.1 Rough simplification

4.3.1.1 Overview

Rough simplification is a preliminary step designed to handle large point clouds: points are imported in a fine regular grid and each non empty voxel is replaced by a single representative vertex. Hence, the goal of this step is merely to adjust sampling density. In this algorithm, 3D point clouds (organized in a sparse 3D regular grid) are first projected onto the 2D grid in the x, y plane. This 2D point cloud (set of non empty voxels) is subdivided according to a regular grid of size s (this size is defined by the user according to the desired final sampling rate) (see figure 4.7). Then, each non-empty cell is replaced by a single representative point: the barycenter of contained points. This step, even if

rough, can be justified in terms of resolution: it is merely a resolution adaptation (in case the resolution of the data is too high compared to the expected results). The important point in this step is that we will not simplify boundary points (as they have already been handled in the previous work, chapter three); and this step should be applied using a small cell size in order to avoid distorting the surface.

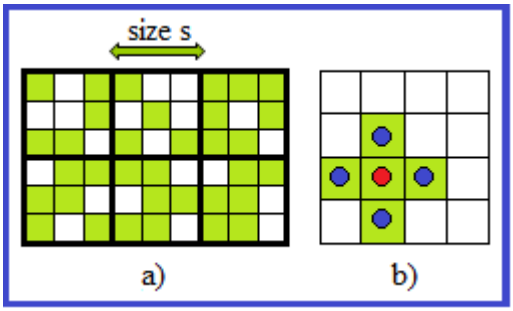

Figure 4.7: a) The size of a cell. b) The barycenter of the points (red color) in the cell.

4.3.1.2 Algorithm

As the size of the cells is small and as we want to preserve boundary points, if a cell contains boundary points, no further representative vertex will be inserted, only included boundary points are kept. Otherwise, if a cell does not contain boundary points, we compute the barycenter of the points in this cell. Based on the description above, we propose a very simple algorithm (Algorithm 4.1) with a linear complexity to roughly simplify the surface:

Algorithm 4.1 RoughSimplification(s)

```
1: for each cell S_q ∈ S do
2:     if S_q contains boundary points then
3:         keep only boundary points;
4:     else
5:         replace all points by p_q;
6:     end if
7: end for
```

4.3.2 Elaborate simplification

4.3.2.1 Overview

In this step, we focus on two main points to process the surface: *curvature* of the surface and *point density*. We process the surface directly in the 3D grid. As previously, the sparse 3D grid (equivalent to the point cloud) is divided according to a regular 3D grid C. The initial size of the cells of C is large (defined by the user) and elaborate simplification will

further subdivide cells of C according to density and curvature criteria. If cells contains boundary points, they are processed based on the combination between *boundary density* and *local curvature* in these cells. Otherwise, subdivision is based on *local curvature* within each cell and adapted to the *size of neighboring cells*. After simplification, the distribution of points has to vary continuously; it must be constrained regularly from the exterior boundary to the inside of the surface. This constraint is introduced to avoid creating bad triangles (in the sense of Delaunay triangulation) in a further meshing step, detailed in chapter five.

4.3.2.2 Analysis

Obviously, our rough preliminary simplification is too basic to reach high simplification rates. It is useful only to adjust the resolution or as a first decimation for huge point clouds (for which a more elaborate simplification cannot be applied directly because of time and space complexity issues). Hence, this preliminary step is optional.
In the case of complex surfaces with a high curvature, simplification must be based both on density and curvature criteria. For this reason, we develop an advanced algorithm to simplify the surface more elaborately. This algorithm is based on an octree subdivision (see figure 4.8) of the surface adapted to its curvature, point density and to the border density. We will combine two subdivision criteria to simplify the surface: subdivision according to the boundary density and subdivision according to the curvature.

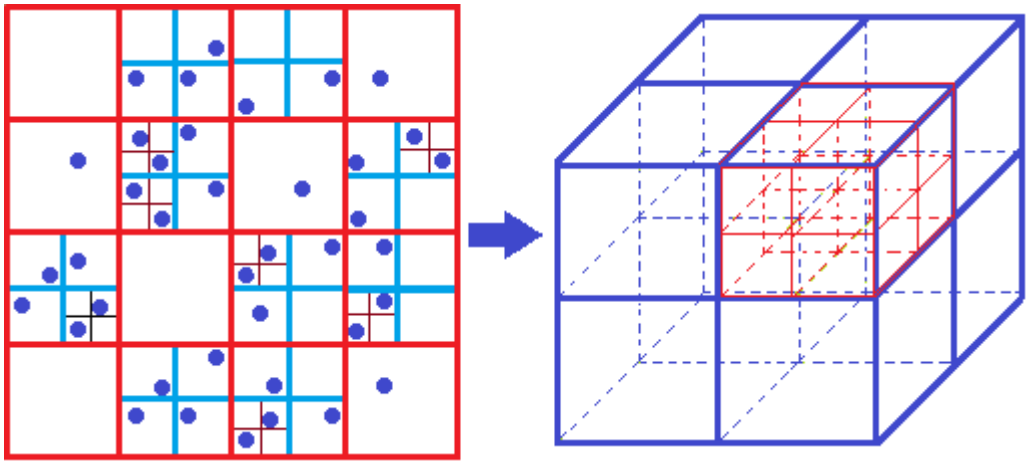

Figure 4.8: 2D illustration of the octree subdivision, we actually handle in the 3D structure.

4.3.2.3 Subdivision according to the boundary density

An important issue is that point density should vary smoothly (in order to preserve the shape of triangles in a further meshing step of the next chapter, chapter five). It must be constrained continuously on the surface and propagate regularly from the boundary to the

inside of the surface. In order to subdivide cells according to the boundary density, we have to build a subdivision criterion. At first, we analyze the density of boundary points (number of boundary points in a cell) and their distribution. Our criterion is based on the size of a cell, the number of boundary points and the distance between them.

4.3.2.3.1 Notation and formula construction

In order to construct the formula for our computations, we introduce the following notations:

- d_{max}: the maximum distance between two boundary points in C_q,
- L_s: the level of subdivision of a cell (see figure 4.9),
- s': the size of the smaller cells after each subdivision of C_q: $s' = \frac{s}{2^{L_s}}$.

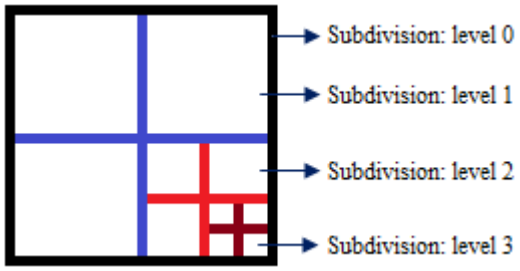

Figure 4.9: The level of subdivision in a cell.

In our context, data points are organized based on a 3D grid structure, each point in a cell has three real xyz coordinates and in the sequel, we will use the Euclidean distance to compute the distance between points. Hence the maximum distance between boundary points in a cell is given by:

$$d_{max} = \max_{i,j \in (1..N_{bp});\, i \neq j} (\|p_i - p_j\|) \tag{4.1}$$

4.3.2.3.2 Boundary density criteria

Subdivision according to boundary density is performed from cells containing boundary points (called first ring) towards the surface interior (ring by ring, starting from the boundary). In the sequel, we will denote by r_i the i^{th} ring of cells based on the 8-connectivity (hence, r_1 is the set of boundary cells). There is a relationship between the density of points and the distance between them in a cell. Obviously, as the density of boundary points in a cell increase, the distance between them will decrease. The formula: $D(density) = N_p(number\,of\,points)/V(volume)$ can be applied to compute the density of points on a volume. In our case, we focused on the number of boundary points N_{bp} in a cell and its size s to calculate point density PD of that cell ($PD = N_{bp}/s$). Hence our criterion is based on PD and d_{max}:

$$(PD > threshold_{pd})\, and\, (d_{max} > threshold_d) \tag{4.2}$$

In order to preserve the shape of the surface for a further triangular meshing step, the size of cells must vary smoothly. Therefore, for boundary cells (also called first ring cells), we state a specific subdivision criterion: if a cell C_q (containing boundary points) satisfies the first condition (4.2), then we check the size of C_q. If the size is less than or equal than a threshold, we keep only boundary points; else, we keep boundary points and the barycenter of inner points in that cell. Otherwise, C_q is subdivided (as an octree).
Starting from the second ring (which contains inner points of the surface), we subdivide cells both according to the local curvature and previous ring cell sizes. Cells are processed ring by ring from the outside to the inside of the surface. The cell size in ring r_i is subdivided according to the sizes of neighboring cells of ring r_{i-1} (the outside adjacent ring of r_i). It means that, if an inner cell satisfies the curvature criterion, we subdivide it according to the average subdivision level of all nearest neighboring cells (see figure 4.10). Let $C_q \in r_i$ and let $\{C_1^{i-1}, \ldots, C_m^{i-1}\}$ be the set of neighboring cells in r_{i-1}, the subdivision level of C_q is computed as:

$$size(C_q) = \frac{1}{m} \sum_{j=1}^{m} size(C_j^{i-1}) \tag{4.3}$$

In the end, the cell size varies smoothly; and if the curvature inside a cell is low, all points in this cell are replaced by one representative point. In next section, we build a flatness criteria in order to subdivide cells according to their curvature.

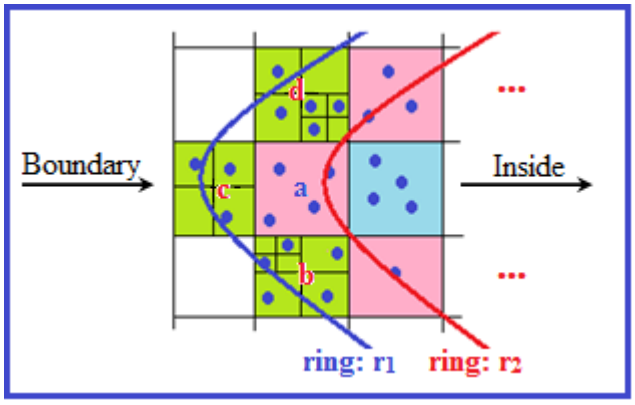

Figure 4.10: Computing the average subdivision level of neighboring cells (cell a is computed based on cells b, c and d).

4.3.2.4 Subdivision according to the curvature

Our goal is to preserve the shape of the surface after simplification. In this section we process the cells containing inner points, from the second ring to the inside of surface. For each cell we apply a principal component analysis (PCA) to estimate the average local curvature of the surface. We thus define a flatness criterion and subdivide cells accordingly.

4.3.2.4.1 PCA flatness criteria

PCA can be used as a useful statistical method to analyze data. This is a technique that can be applied to simplify a surface of 3D point clouds (see [PGK02, MVF04, BTD07, LT07, ZAMZ11]). In order to estimate the curvature or the flatness of a cell, we compute the *PCA* of the vertices of the cell. The eigenvalues of the corresponding covariance matrix provide a curvature information and we define accordingly a flatness criterion. Cells that do not meet this flatness criterion are subdivided until either their size is lesser than a threshold or they satisfy the criterion.
We use the formula below to compute the covariance matrix (C_v) for each cell:

$$C_v = \frac{1}{N_{cq}} \sum_{i=1}^{N_{cq}} (p_i - \bar{p}) (p_i - \bar{p})^t ; \tag{4.4}$$

Where:

- $\bar{p}$: barycenter of points in C_q,
- λ_i, v_i: the i^{th} eigenvalue and i^{th} eigenvector of C_v.

The eigenvectors of C_v provide information about the principal directions of a point set. More precisely, the eigenvectors provide main axes of the cloud, while eigenvalues provide its stretching along the corresponding axes. Hence, the eigenvector associated to the smallest eigenvalue provides an average normal vector while both other eigenvalues are related to principal curvatures.
Following the above analysis and applying the ideas introduced in [PGK02, MVF04, Bel08, ZAMZ11], let us sort eigenvalues: $\lambda_0 \leq \lambda_1 \leq \lambda_2$. If the value of λ_0 is very small or even equal 0, that means all the points in a cell are approximately on a plane (it satisfied the flatness criteria). In such a case, the average normal vector on a local surface within a cell can be determined based on the direction of v_0. The **flatness criterion** "∂" below is considered as a condition to further subdivide cells (and hence to control the simplification of the surface):

$$\partial = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{4.5}$$

For each point on the local surface, if their normal vectors are distributed isotropically, these points will lie on the same plane. This solution is given by Hugues Hoppe [HDD+92] to compute the orientation of the tangent plane (see figure 4.11a): for each data point p_i, a tangent plane is computed by least-squares approximation based on *PCA* of the k nearest neighbor of p_i.
In our case, we use the flatness criteria (4.5) to estimate the local curvature in a cell. The minimum value of ∂ equal 0, while its maximum value equal 1/3, and our flatness criteria are based on the range of these values. (see figure 4.11)

The curvature in a cell is first determined by computing ∂. Then, ∂ is compared with a threshold value from the user. If $\partial \leq threshold_\partial$, we replace all points in this cell

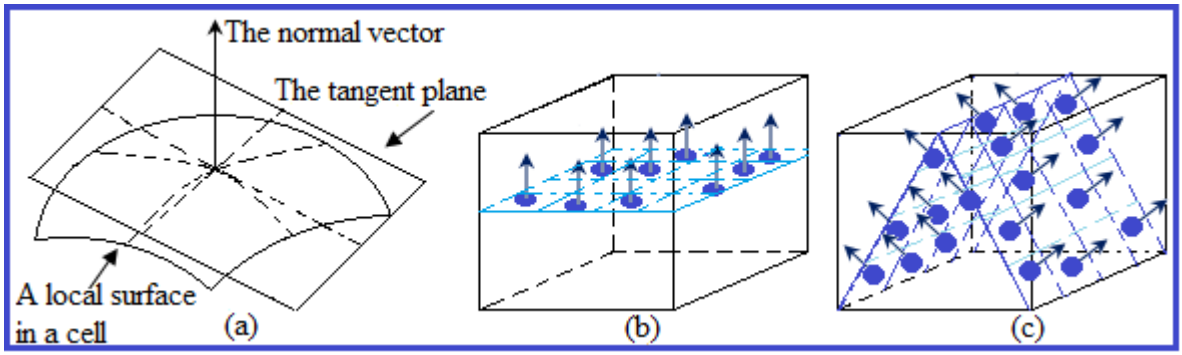

Figure 4.11: Estimation of the curvature in a cell: (a) Computing the orientation of points; (b) The points are approximately on a plane within a cell (λ_0 is very small, λ_1 and λ_2 are large); (c) λ_0 is large or ($\lambda_0 \simeq \lambda_1 \simeq \lambda_2 \simeq 1$) or ($\partial \simeq 1/3$) $\Rightarrow$ this cell is subdivided.

by one representative point. This way can simplify the surface efficiently and the ratio of simplification is very high (if the points in that cell are approximately on a plane). However, the density of points could vary irregularly after a large number of points have been removed. For this reason, we have to combine with the computation of point density and size of cells to constrain the distribution of points on the surface to be as regular as possible.

4.3.2.5 Algorithms

According to the previous analysis, we now define our simplification algorithm. Our algorithm covers cells ring by ring (starting from boundary cells to the inside of the surface), each ring is processed clockwise direction (see figure 4.12).

We start from the first ring, blue color (i.e. the ring of boundary points). In this ring, we begin with the left-most cell (1) and follow the clockwise direction to compute, subdivide and simplify each cell. From the second ring (yellow color), we also begin with the left-most cell (2) and so forth for following rings (third - green, fourth - pink, etc). The algorithms below are used to simplify the surface: Algorithm 4.2 is used to process the cells containing boundary points in the first ring. Algorithm 4.3 is used to process the cells containing inner points from the second ring to the inside of surface.

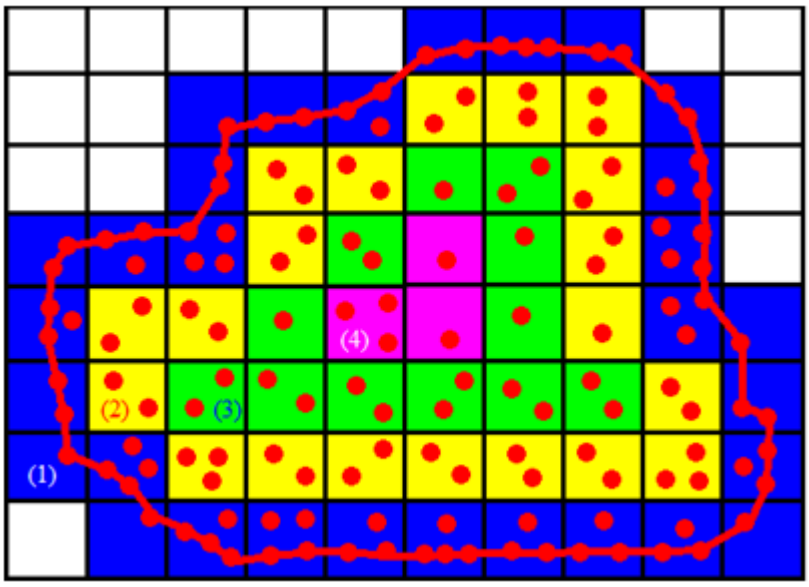

Figure 4.12: Illustration of the elaborate algorithm.

Algorithm 4.2 SimplifyBoundaryCells(s)

$N_{bp} = 0$, $L_s = 0$; //start from the left-most cell, follow the clockwise direcrion.
for each boundary cell C_q(size s) $\in S$ **do**
    compute N_{bp}, d_{max};
    if C_q satisfy the density criteria(4.2) **then**
        if size $s \leq threshold_s$ **then**
            keep only boundary points;
        else
            replace all points by boundary points and the barycenter of inner points;
        end if
    else //subdivide C_q by L_s.
        $L_s = L_s + 1$;
        $s' = s/(pow(2, Ls))$;
        for each $C_q(s') \in C_q(s)$ **do**
            if $C_q(s')$ contains boundary points **then**
                SimplifyBoundaryCells(s');
            else
                SimplifyInnerCells(s');
            end if
        end for
    end if
end for

For each inner cell, we compute the curvature criterion (4.5). If it satisfies the threshold, we first subdivide this cell based on formula (4.3); then, replace all points in each sub-cell by their barycenter. Otherwise, we subdivide this cell and repeat the process until all conditions of the criterion are satisfied.

Algorithm 4.3 SimplifyInnerCells(s)

```
1: L_s = 0; //start from the left-most cell, follow the clockwise direcrion.
2: for each inner cell C_q (size s) ∈ S do
3:     compute the covariance matrix of points in C_q;
4:     if C_q satisfy the flatness criteria(4.5) then
5:         subdivide C_q based on (4.3);
6:         replace all points by the barycenter in each sub-cell;
7:     else //subdivide C_q by L_s.
8:         L_s = L_s + 1;
9:         s' = s/(pow(2, Ls));
10:        SimplifyInnerCells(s');
11:    end if
12: end for
```

In this step, our computing experiences have seen that the processing time mostly depends on values of ∂; before and after combining with step one (rough simplification) (see table 4.3), and less depends on s (size of a cell). Normally, the number of points in a cluster (using PCA) is around from 30 points [MD04, WCZ+08, MWZ10]. In our case, the curvature within a cell of a geologic surface is low and the 3D points are sparse. Therefore, we choose $s \leq 10$ (that is initial cells containing at most 100 voxels) and many values of ∂ to implement. As a result, the time is affected if the number of points in a cell greater than 36 or ∂ close to 0 and before combining with step one. We keep the boundary and combine two steps (rough and elaborate) to simplify a surface; thus, the surface is simplified completely, the initial shape of the surface is preserved, and the time is controlled.

4.4 Results

In this section, we present some of our results. In order to evaluate the quality of the surfaces generated by our methods, we used the Metro tool [CRS98] implemented in Meshlab [Cou13] to compute the approximation error of the surfaces before and after simplification. We compare the quality between $S1$ (the input surface of 3D point clouds) and $S2$ (the output surface after simplifying $S1$) by computing approximation errors between them. We measure both the maximum error $\Delta_{max}(S1, S2)$, i.e. the two-sided Hausdorff distance, and the mean error $\Delta_{avg}(S1, S2)$, i.e. the area-weighted integral of the distance from point to surface. This method and tool are also detailed and used in Pauly [PGK02].
For step one (rough simplification), the computations are very fast. The algorithm has been tested on many surfaces with different numbers of points to compare the running time, the

simplification rate and the approximation error with the cluster method [PGK02] implemented in Meshlab [Cou13] and named "Merge Close Vertices" inside the "filter clean". The cluster method and our rough simplification are not equivalent because in the rough method, the boundary points are kept (after the boundary simplification detailed in chapter 3), which is not the case in the cluster method. The results are presented in table 4.1: the computing time of our method is faster than the cluster method, while the simplification rate is slightly lower (depending on the initial shape of the input surface) because we keep the boundary points. For this reason, the approximation errors between the input surfaces and the output surfaces of our method are lower than those with the cluster method (see table 4.1).

Input points	Size of bounding box (x,y,z)	Rough method				Cluster method			
		p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}	p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}
15626	303,110,28.5	1194(92.3)	26	0.017	0.0003	475(97)	163	0.019	0.0005
32402	382,161,73.3	1881(94)	36	0.029	0.0013	1297(96)	317	0.032	0.0015
60511	435,294,105.3	2942(95)	47	0.020	0.0005	1909(97)	609	0.032	0.0007
68956	586,411,88.9	3695(95)	53	0.023	0.0009	2758(96)	698	0.039	0.0018
148317	862,405,83.6	6368(96)	98	0.016	0.0006	4675(97)	1496	0.017	0.0006
346796	1162,652,132.7	13030(96)	206	0.032	0.0004	11068(97)	3556	0.037	0.0007
664582	1162,847,144.7	26583(96)	377	0.025	0.0003	19872(97)	7116	0.060	0.0004

Table 4.1: Comparison between the rough method and the cluster method. We use the same size of a neighboring distance between the points (cell size: s = cluster size = 6), and run them on the same a computer. (p.output: output points; s.rate: simplification rate)

Input points	Rough method: $s = 6$				Rough method: $s = 8$			
	Output points (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}	Output points (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}
15626	1194(92.3)	26	0.017	0.0003	1103(93.5)	18	0.020	0.0007
32402	1881(94)	36	0.029	0.0013	1491(95)	26	0.045	0.0014
60511	2942(95)	47	0.020	0.0005	2299(96)	37	0.043	0.0011
68956	3695(95)	53	0.023	0.0009	3086(95.5)	49	0.028	0.0012
148317	6368(96)	98	0.016	0.0006	5100(96.5)	88	0.023	0.0010
346796	13030(96)	206	0.032	0.0004	12125(96.5)	191	0.034	0.0006
664582	26583(96)	377	0.025	0.0003	22595(96.6)	325	0.046	0.0005

Table 4.2: Comparison of the rough method: we use the different cell-sizes and run them on the same a computer (s.rate: simplification rate). The results with cell size (s=3) can be found in table 4.3

In this rough simplification, the simplification rate is controlled by the cell size. In our method, although the boundary points are kept to preserve the shape of the surface, this approach does not take into account the curvature of the surface and hence is too rough to be applied for receiving accurate simplified surfaces. If we use a larger cell size to simplify, the received results are not accurate (see figure 4.13 and table 4.2), even if they remain comparable with the cluster method. In parallel, the larger the cell size, the lower the computing time. According to us, this step can only be applied to simplify a simple surface of 3D points or to adjust the resolution of a 3D point cloud by using a small cell size. In the cluster method, all points of the surface (boundary points and inner points) are simplified, i.e. each cluster is replaced by one representative point. Therefore, the simplification rate is higher, but the shape of the output surface is not well preserved (as compared in figure 4.14). We now analyze our obtained results before and after combining our two steps (rough and elaborate) to simplify completely the surface.

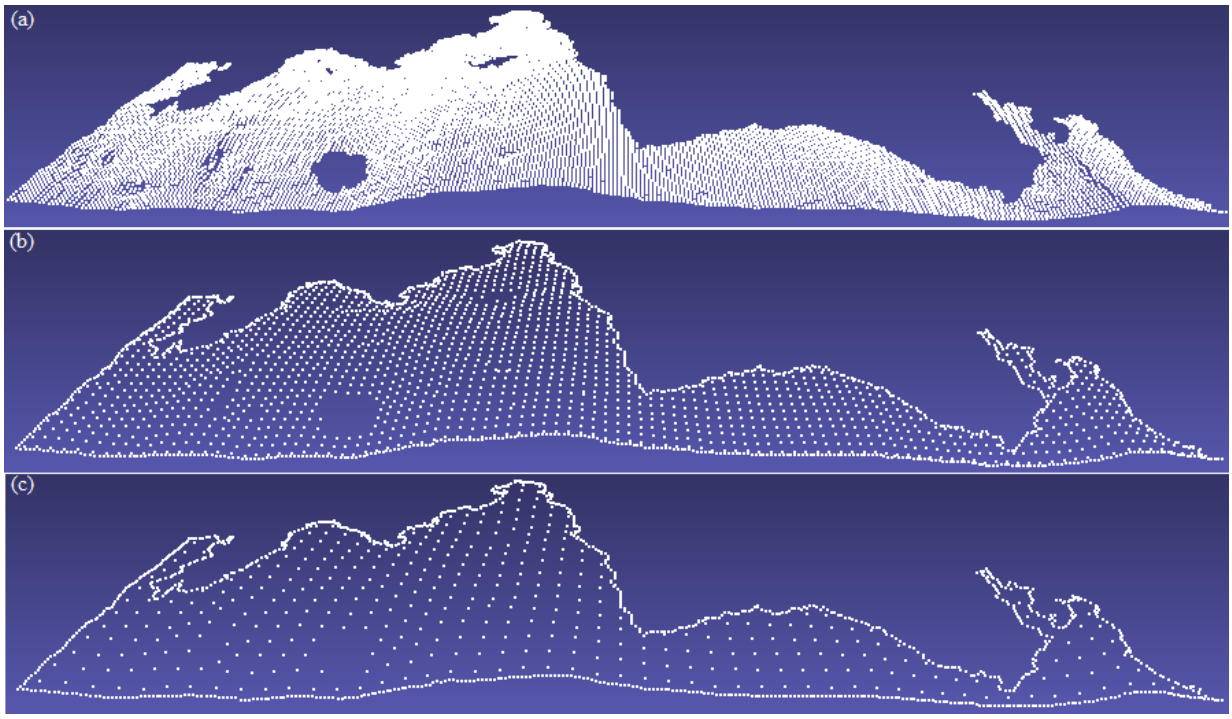

Figure 4.13: Rough simplification: the shape of the initial surface is not preserved and received results are not accurate using a large cell size. a) a geological surface; b) after simplifying with cell size: $s = 3$ (Δ_{max}: 0.006, Δ_{avg}: 0.0002); c) after simplifying with cell size: $s = 6$ (Δ_{max}: 0.017, Δ_{avg}: 0.0003).

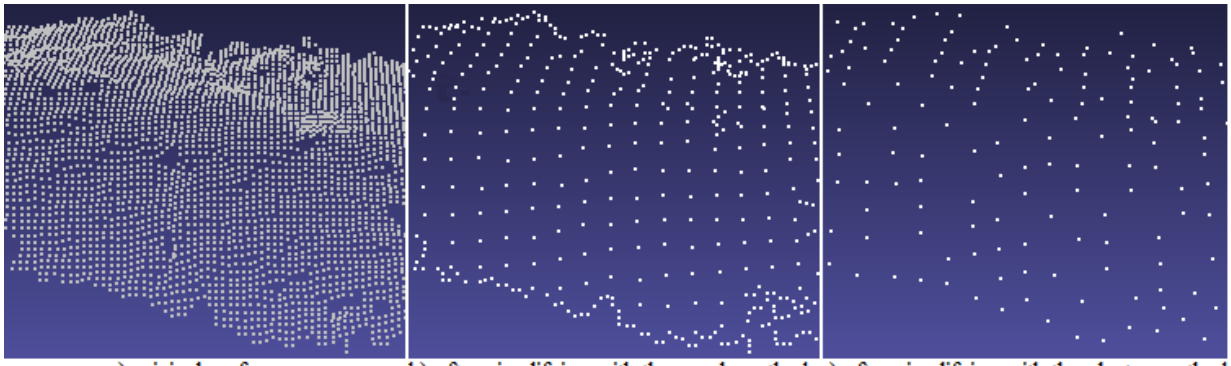

Figure 4.14: Shape comparison by computing the approximation error of the surface after simplifying with the same size of neighboring distance: b) using the rough method (Δ_{max}: 0.0142, Δ_{avg}: 0.0004); c) using the cluster method (Δ_{max}: 0.0296, Δ_{avg}: 0.0009).

In step two (elaborate simplification), we have tested our approach on different surfaces used in table 4.1 (with the same input points) to compare the results of the rough method and the elaborate method with the different parameters. The results are detailed in table 4.3. In the elaborate method, we provide the different values of ∂ in order to show that: if the value of ∂ is close to 0 (a lot of cells are subdivided), the obtained surface is smooth, close to the initial surface (small simplification rate) and the processing time is low; otherwise, if the value of ∂ is close to 1/3, the obtained surface is far from the original one (higher simplification rate) and the running time is higher.

The second step of our procedure permits to receive surfaces close to the initial surfaces (small errors). The final result does not depends on the rough simplification, but the associated computing time is closely related with this initial simplification. In Table 4.3, Time1 is the time required to simplify the surface without applying the rough simplification since Time2 corresponds to the time required when combining both steps. The time is also affected by the initial shape of the surface. If the curvature of the surface is high or the initial shape of the surface is complex, it leads to a higher processing time for that surface. One can also notice that most of the computing time is required to obtain the last percents of simplification while preserving satisfactory errors. Preserving a regular distribution of points evidently constraints the simplification rates compared to the cluster method. Moreover, we have obtained output surfaces preserving the initial geometry of the surface (as compared in figures 4.15, 4.17, 4.18). Figure 4.16 shows the result of point distribution constrained from the boundary to the inside of the surface. As a result, a good triangular surface can be obtained in a further meshing step.

Input points	Rough Method				Elaborate method					
	p.output (s.rate%)	time (ms)	Δ_{max}	Δ_{avg}	Values of ∂	Time1 (ms)	Time2 (ms)	p.output (s.rate%)	Δ_{max}	Δ_{avg}
15626	2344(85)	33	0.006	0.0002	$\partial \leq 0.03$	856	574	1571(90)	0.013	0.0004
					$\partial \leq 0.12$	851	568	1123(92.8)	0.020	0.0006
					$\partial \leq 0.20$	842	567	1071(93.2)	0.022	0.0007
32402	7285(77.5)	37	0.005	0.0003	$\partial \leq 0.03$	1384	1186	5818(82)	0.009	0.0004
					$\partial \leq 0.12$	1380	1179	5509(83)	0.010	0.0005
					$\partial \leq 0.20$	1371	1178	5185(84)	0.013	0.0005
60511	12668(79)	57	0.005	0.0003	$\partial \leq 0.03$	5271	3231	9879(84)	0.007	0.0004
					$\partial \leq 0.12$	5026	2958	9377(84.5)	0.010	0.0004
					$\partial \leq 0.20$	3776	2910	6786(89)	0.031	0.0009
68956	18681(73)	70	0.006	0.0003	$\partial \leq 0.03$	5940	4031	15858(77)	0.007	0.0003
					$\partial \leq 0.12$	5883	3993	14481(79)	0.009	0.0003
					$\partial \leq 0.20$	5764	3898	12413(82)	0.010	0.0004
148317	28137(81)	128	0.003	0.0001	$\partial \leq 0.03$	22106	14194	21122(86)	0.004	0.0003
					$\partial \leq 0.12$	21167	13825	20820(86)	0.005	0.0003
					$\partial \leq 0.20$	15896	12079	18916(87)	0.024	0.0004
346796	72089(79)	264	0.003	0.0001	$\partial \leq 0.03$	114795	111362	56448(84)	0.004	0.0002
					$\partial \leq 0.12$	111289	107309	52187(85)	0.005	0.0003
					$\partial \leq 0.20$	110623	101544	50112(86)	0.032	0.0005
664582	76702(88.5)	387	0.019	0.0002	$\partial \leq 0.03$	910160	170636	75112(88.7)	0.020	0.0003
					$\partial \leq 0.12$	909040	170553	73105(89)	0.022	0.0004
					$\partial \leq 0.20$	909002	170550	63136(90.5)	0.040	0.0006

Table 4.3: Comparison between the rough method (cell size $s = 3$) and the elaborate method. Time1: the computing time by using only step2 (initial cell size $s = 8$); Time2: the total computing time by using both steps (rough first: cell size $s = 3$; then elaborate: initial cell size $s = 8$); p.output: output points, s.rate: simplification rate.

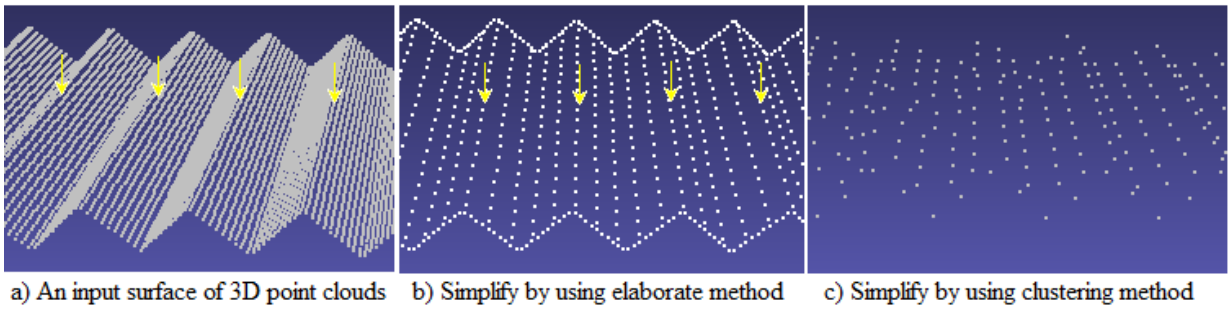

Figure 4.15: Comparison of the shape of the surface between the two methods by using the same size of neighboring distance: a) an input surface of 23559 points; b) after using the elaborate method, remaining points: 2305, the approximation error between (a) and (b) is Δ_{max}: 0.007, Δ_{avg}: 0.0007; c) after using the cluster method, remaining points: 801, the approximation error between (a) and (c) is Δ_{max}: 0.015, Δ_{avg}: 0.003.

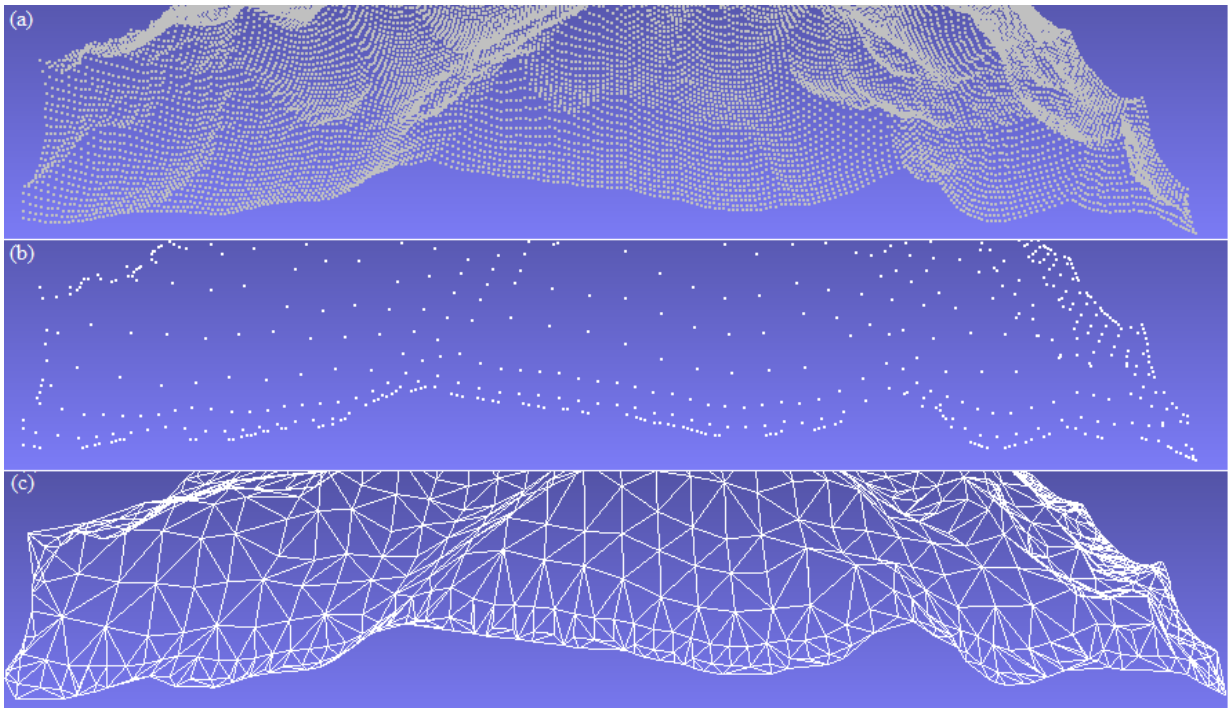

Figure 4.16: a) Input surface with 66049 points; b) After simplifying by using the elaborate method (s=8, $\partial \leq 0.09$, remaining points: 1840), the points are constrained from the boundary to the inside; c) A good triangular surface can be obtained in a further meshing step (the approximation error between (a) and (c) is Δ_{max}: 0.018; Δ_{avg}: 0.002)

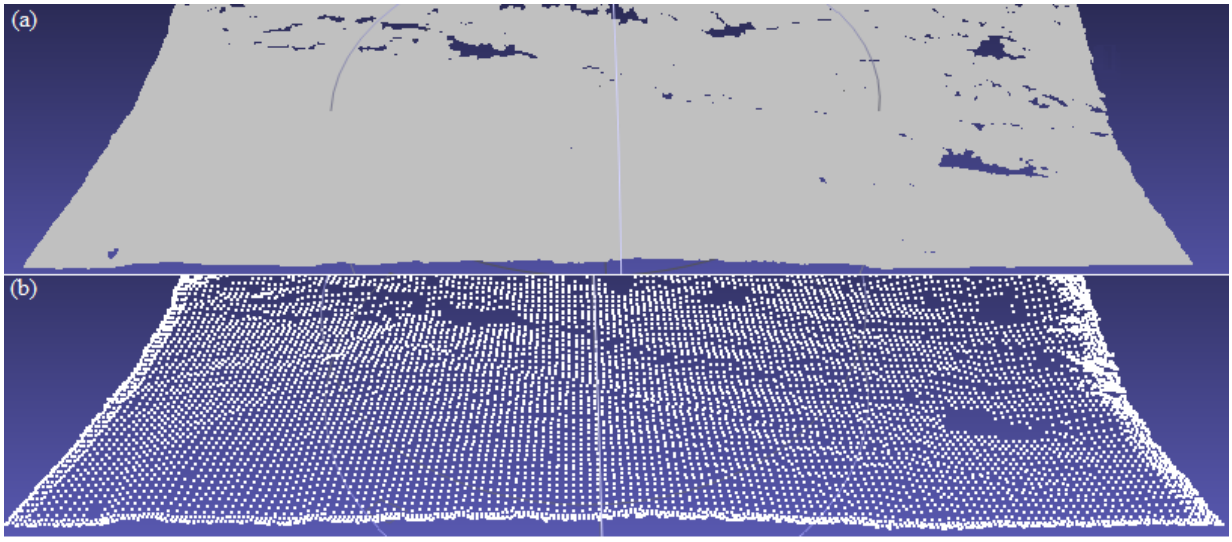

Figure 4.17: Comparison of the shape of a geological surface: a) input surface of 664582 points; b) after simplifying by using the elaborate method (s=8, $\partial \leq 0.15$), the simplification rate: 89%; the approximation error between (a) and (b) Δ_{max}: 0.0248; Δ_{avg}: 0.0004.

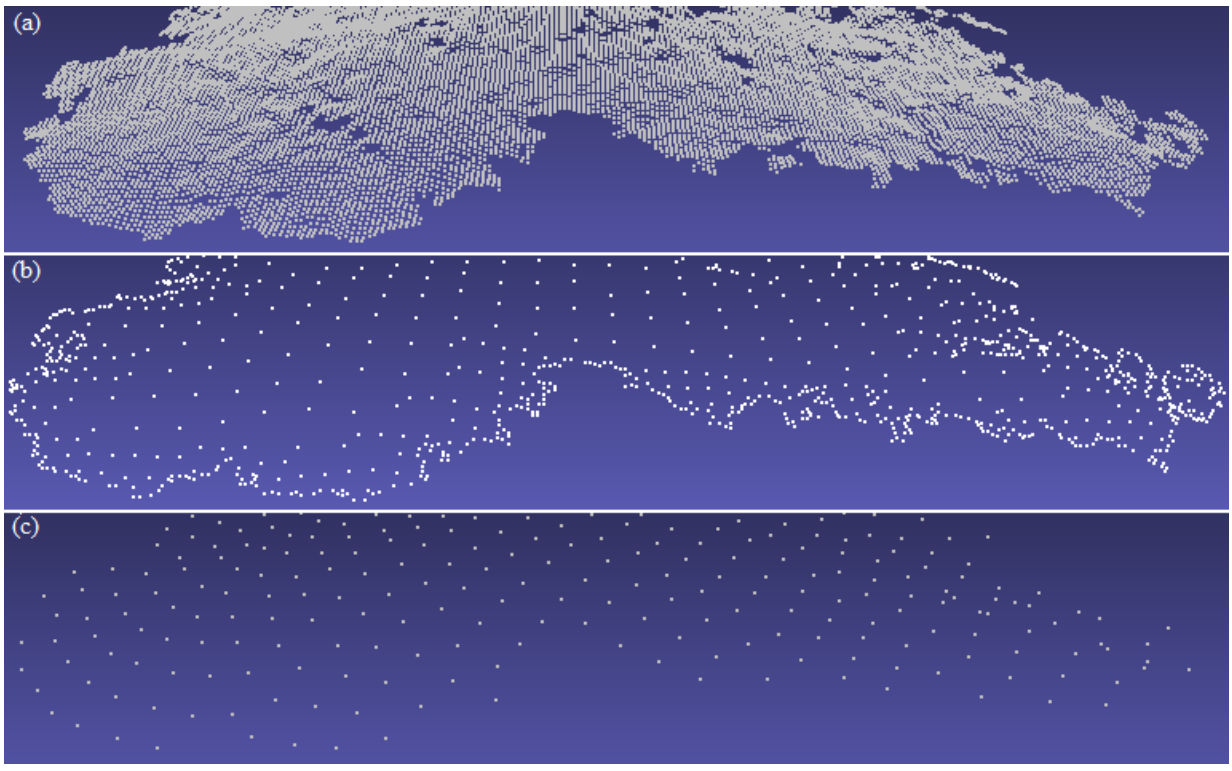

Figure 4.18: Comparison of the approximation errors: a) input surface with 2136 kb; b) after simplifying by using the elaborate method, s=8, $\partial \leq 0.12$, the remain data: 309 kb; the approximation errors between (a) and (b) are Δ_{max}: 0.015; Δ_{avg}: 0.0005; c) after simplifying by using the cluster method, cluster size = 8, the remain data: 46 kb; the approximation errors between (a) and (c) are Δ_{max}: 0.034; Δ_{avg}: 0.0014.

Input points	Cluster method				Global method				
	p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}	Value of ∂	p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}
15626	438(97.2)	133	0.025	0.0007	$\partial \leq 0.03$	1250(92)	415	0.014	0.0005
					$\partial \leq 0.12$	1093(93)	312	0.021	0.0006
					$\partial \leq 0.20$	1077(93.1)	305	0.022	0.0007
32402	1135(96.5)	303	0.098	0.0024	$\partial \leq 0.03$	5369(83.4)	994	0.011	0.0005
					$\partial \leq 0.12$	5221(83.8)	883	0.012	0.0005
					$\partial \leq 0.20$	5100(84.3)	847	0.014	0.0006
60511	1634(97.3)	477	0.043	0.0014	$\partial \leq 0.03$	9246(85)	2500	0.011	0.0005
					$\partial \leq 0.12$	8747(85.5)	2216	0.013	0.0006
					$\partial \leq 0.20$	5786(90)	2001	0.032	0.0010
68956	2414(96.5)	544	0.057	0.0013	$\partial \leq 0.03$	15567(77.5)	4318	0.008	0.0003
					$\partial \leq 0.12$	14558(79)	4049	0.009	0.0003
					$\partial \leq 0.20$	12531(82)	3715	0.010	0.0004
148317	4153(97.2)	1149	0.024	0.0007	$\partial \leq 0.03$	20290(86.3)	14194	0.006	0.0003
					$\partial \leq 0.12$	19022(87)	13825	0.007	0.0003
					$\partial \leq 0.20$	17196(88.4)	12079	0.025	0.0004
346796	9364(97.3)	2766	0.084	0.0015	$\partial \leq 0.03$	54073(84.4)	97309	0.005	0.0003
					$\partial \leq 0.12$	51157(85.3)	87138	0.006	0.0004
					$\partial \leq 0.20$	49073(86)	82538	0.033	0.0005
664582	17280(97.4)	5793	0.099	0.0014	$\partial \leq 0.03$	73104(89)	170636	0.022	0.0004
					$\partial \leq 0.12$	69781(89.5)	170553	0.024	0.0005
					$\partial \leq 0.20$	59812(91)	170550	0.042	0.0007

Table 4.4: Comparison of the best results between the cluster method (cluster size s=8) and the global method (cell size: rough s=3; elaborate s=10). We run on the same a computer. p.output: output points; s.rate: simplification rate.

In table 4.4, we also use the same input surfaces implemented in table 4.1 of the cluster method in order to compare with our global method (combination of the two steps), entering for both what we consider as the best input parameters. In the cluster method, we use a cluster size = 8. In the global method, we first use a cell size: $s = 3$ with the rough method; we then use a cell size: $s = 10$ with the elaborate method and different values of ∂ (as presented in table 4.3). The results show that the processing time of the global method is higher, the simplification rate is lower, but the approximation errors are largely better. Table 4.4 emphasizes that the two methods are difficult to compare. It is the reason why we try to decrease the quality of our results in order to be able to make a comparison with the cluster method. We use the rough simplifications studied in Table 4.2 and then apply our elaborate simplification. The results are gathered in Table 4.5 and 4.6 and have to be compare with the left side of Table 4.4. The results evidently depends on each surface but the tendency is clear. We can obtain higher simplification rate, computing times closer to the cluster method while receiving better errors.

Input points	Rough Method				Global method				
	p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}	Values of ∂	Time (ms)	p.output (s.rate%)	Δ_{max}	Δ_{avg}
15626	1194(92.3)	26	0.017	0.0003	$\partial \leq 0.03$	159	1089(93)	0.018	0.0004
					$\partial \leq 0.12$	142	1015(93.5)	0.020	0.0005
					$\partial \leq 0.20$	138	938(94)	0.026	0.0007
32402	1881(94)	36	0.029	0.0013	$\partial \leq 0.03$	489	1879(94.2)	0.029	0.0013
					$\partial \leq 0.12$	425	1717(94.7)	0.031	0.0014
					$\partial \leq 0.20$	399	1620(95)	0.032	0.0014
60511	2942(95)	47	0.020	0.0005	$\partial \leq 0.03$	1844	2723(95.5)	0.022	0.0006
					$\partial \leq 0.12$	1721	2601(95.7)	0.023	0.0006
					$\partial \leq 0.20$	1685	2360(96.1)	0.033	0.0010
68956	3695(95)	53	0.023	0.0009	$\partial \leq 0.03$	3518	3103(95.5)	0.024	0.0009
					$\partial \leq 0.12$	3429	2758(96)	0.026	0.0010
					$\partial \leq 0.20$	3231	2552(96.3)	0.029	0.0011
148317	6368(96)	98	0.016	0.0006	$\partial \leq 0.03$	12692	6132(96)	0.017	0.0006
					$\partial \leq 0.12$	11587	5636(96.2)	0.020	0.0007
					$\partial \leq 0.20$	10852	5191(96.5)	0.026	0.0008
346796	13030(96)	206	0.032	0.0004	$\partial \leq 0.03$	70019	12831(96.3)	0.033	0.0004
					$\partial \leq 0.12$	62453	12131(96.5)	0.034	0.0004
					$\partial \leq 0.20$	58931	11444(96.7)	0.036	0.0005
664582	26583(96)	377	0.025	0.0003	$\partial \leq 0.03$	108811	25254(96.2)	0.027	0.0004
					$\partial \leq 0.12$	95671	23924(96.4)	0.028	0.0005
					$\partial \leq 0.20$	89367	21931(96.7)	0.043	0.0007

Table 4.5: Comparison between the rough method (cell size $s = 6$) and the global method. In the global method, we use both steps to simplify the surface (rough first: cell size $s = 6$; then elaborate: initial cell size $s = 10$); p.output: output points, s.rate: simplification rate.

Input points	Rough Method				Global method				
	p.output (s.rate%)	Time (ms)	Δ_{max}	Δ_{avg}	Values of ∂	Time (ms)	p.output (s.rate%)	Δ_{max}	Δ_{avg}
15626	1103(93.2)	18	0.020	0.0007	$\partial \leq 0.03$	145	1012(93.5)	0.020	0.0005
					$\partial \leq 0.12$	138	985(93.7)	0.023	0.0006
					$\partial \leq 0.20$	135	938(94)	0.027	0.0007
32402	1491(95)	26	0.045	0.0014	$\partial \leq 0.03$	420	1482(95.2)	0.046	0.0014
					$\partial \leq 0.12$	379	1471(95.3)	0.046	0.0014
					$\partial \leq 0.20$	333	1325(95.6)	0.048	0.0015
60511	2289(96)	37	0.043	0.0011	$\partial \leq 0.03$	1796	2231(96.2)	0.044	0.0011
					$\partial \leq 0.12$	1678	2210(96.3)	0.045	0.0011
					$\partial \leq 0.20$	1586	2117(96.5)	0.051	0.0012
68956	3086(95.5)	49	0.028	0.0012	$\partial \leq 0.03$	3452	3034(95.6)	0.028	0.0012
					$\partial \leq 0.12$	3371	2869(96.1)	0.029	0.0013
					$\partial \leq 0.20$	3120	2413(96.5)	0.033	0.0014
148317	5100(96.5)	88	0.023	0.0010	$\partial \leq 0.03$	11987	5042(96.6)	0.023	0.0010
					$\partial \leq 0.12$	11218	4746(96.8)	0.024	0.0011
					$\partial \leq 0.20$	10452	4449(97)	0.028	0.0012
346796	12125(96.5)	191	0.034	0.0006	$\partial \leq 0.03$	63120	12120(96.5)	0.034	0.0006
					$\partial \leq 0.12$	59921	11791(96.6)	0.036	0.0007
					$\partial \leq 0.20$	57821	10403(97)	0.038	0.0008
664582	22595(96.6)	325	0.046	0.0005	$\partial \leq 0.03$	9756	21266(96.8)	0.047	0.0005
					$\partial \leq 0.12$	8891	20602(96.9)	0.047	0.0006
					$\partial \leq 0.20$	8895	19937(97)	0.049	0.0008

Table 4.6: Comparison between the rough method (cell size $s = 8$) and the global method. In the global method, we use both steps to simplify the surface (rough first: cell size $s = 8$; then elaborate: initial cell size $s = 10$); p.output: output points, s.rate: simplification rate.

In conclusion, the advantages of the rough and the cluster methods are the running times and the simplification rates which can be reached, but the approximation errors are important. In the rough simplification, we have a better control of the global shape. In our elaborate method, the most important point is to preserve the characteristics of the original surface while reaching low approximations errors. The counterpart is higher computing times but experiments show that a compromise can be received, if required, in order to quickly simplify surfaces with good accuracies.

4.5 Discussion and evaluation

Our method has two advantages compared to existing methods. First, we use a cell to gather and compute the points in a local neighborhood to simplify the surface. By using a cell, there are no outside points between the cells; only one loop is used to consider all points of the surface. On the contrary, the other methods [PGK02, ZG10] use a sphere or a cylinder (both are the same) to compute the neighboring points within a threshold value of a radius r (see figure 4.19). Therefore, after each iterative operation, they have to process the points outside of these sphere/cylinder. The second advantage is that searching to compute a neighboring point within a cell is faster than within a sphere [Smi90, DE96]. Our approach also takes advantage of the fact that our data are already organized in a sparse numeric volume, and hence we don't need to lose time and memory space to build accelerating data structure for k_neighbors computation (such as kd-trees or octrees).

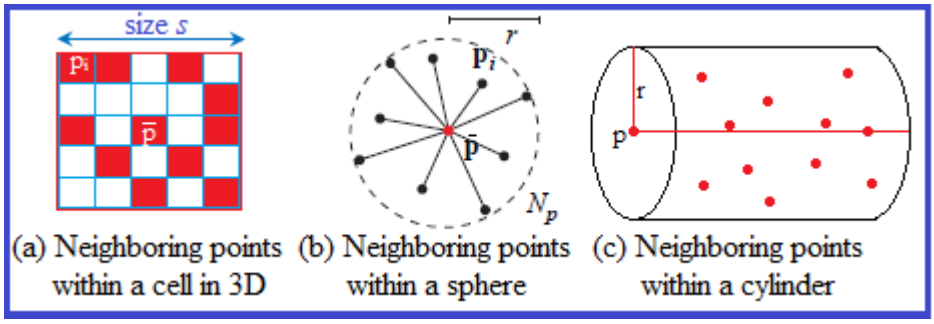

Figure 4.19: Determining of neighboring points.

4.6 Conclusion

In this chapter, we have presented a method to simplify an elevation surface defined by a 3D point cloud. The work is connected from the previous work in chapter three. The surface simplification of 3D point clouds using PCA can normally yield an expensive computation. In our case, the input data are stored in the 3D grid volume, implicitely containing the neighborhood information for each point. We have taken this advantage; combined two steps for rough and elaborate simplification; and two ways of subdivision by using a cell to grow and simplify the surface. The output surface preserves the initial shape of the input surface; the point density and the point distribution are kept regularly, constrained from the boundary to the inside of surface; and the running time is controlled. A good distribution of points is an advantage to obtain a good triangulation of the point clouds. Our method and obtained results in this chapter have been published in [VSAM13].
The obtention of a good triangulation will be presented in the next chapter.

Chapter 5

Surface Triangulation

Contents

5.1 Introduction **63**
5.2 Related work **65**
5.2.1 The methods in 2D 65
5.2.2 The methods in 3D 67
5.3 Methods for triangulating the surface **70**
5.3.1 Overview 70
5.3.2 Concepts, notation and definition 71
5.3.3 Building a seed triangle 73
5.3.4 Searching conditions in one side of an edge 75
5.3.5 Delaunay Criterion 75
5.3.6 Neighboring points search based on the voxel tracing 77
5.3.7 Triangulating a surface 82
5.3.8 Processing the outside triangles on the boundary 83
5.4 Implement **84**
5.5 Results **86**
5.6 Discussion and evaluation **89**
5.7 Conclusion **90**

5.1 Introduction

The last step of our modeling process consists in reconstructing a surface (actually, in our case, a triangular mesh) from the 3D point cloud. Reconstructing the surface of 3D point clouds is a reconstruction from a cloud of 3D points to a triangular mesh. This process

approximates a discrete point cloud by a continuous/smooth surface (depending on the input data and the applications of users). Surface triangulation of a 3D object is a fast and efficient way to reconstruct such an approximating surface in the 3D space. Therefore, meshing a 3D point cloud has been studied widely in the field of geometric modeling and has become a necessary ingredient in many researches of computer graphics and their application [BPK+07].

The goal is to reconstruct a surface of triangular mesh from a given 3D point cloud, such that the triangular mesh must approximate the original shape of the input surface. In other words, the surface reconstruction can be defined and solved explicitly as follows: the point cloud in sampled from an initial surface S, we reconstruct a surface S' from S and compare S' to the initial surface S, such that the points belong to S lie on or close to S'; the topological shape of S' approximates the geometric model of S.

Various methods are available for generating a triangular mesh from a 3D point cloud [Hop94, Ede01, DG03, KL06, Ma11], they can be applied on both open and close surfaces. There are mainly two kinds of approaches: the first one focuses on geometrical considerations [BMR+99, ACK01, OBS05]; the second makes use of implicit surfaces [Hop94, ZG04, KBH06]. The computing time of algorithms and the quality of generated triangular meshes are two essential objectives that the researchers are facing. The effectiveness of the method for meshing a triangular surface of a 3D point cloud depends not only on the types of the surface, the input data structure, but also on the characteristics of this data.

Our purpose is to build an optimal geological triangulated surface in order to get the best simulation of the oil reservoir. As we presented in the previous chapters, the problem comes from the seismic data, with a very large number of 3D points (can reach several millions of points). Therefore, the time necessary to triangulate these data points with "classical" methods can be very high (and hence unacceptable in terms of time and memory). The proposed method for simplifying the surface in chapter four is an important step for the further meshing step, indeed, simplification produces smaller data sets whose density varies regularly over the surface. In this chapter, we propose a method to triangulate this surface by using a fast search algorithm based on the 2D Delaunay triangulation.

We start from the first Delaunay triangle created by the first Delaunay boundary edge and its neighboring points in a local region. The next adjacent triangle T' is then created between an edge e_i of the first triangle T and a neighboring point based on the voxel traversal search within a square box, on one side of e_i. At each step, the new edges of the created triangle T' are inserted in a pool and the process is iterated starting from this triangle T' by removing edge e_i from this pool, until empty the pool. In the implement, we add an introduction and execution of a method for searching a neighboring point based on computing the compactness of a triangle in order to compare with our method. The obtained results show that the our method is very fast, due to the efficiency of our voxel traversal algorithm within a local neighborhood [AW87]. Our method preserves the characteristics of the output surface which approximate the initial shape of the input surface.

The following sections in this chapter are organized as follows: we first present the existing methods for triangulating a surface of 2D and 3D point clouds. Then, our method is pre-

sented in detail in the next section (including an overview analysis, methods for searching a neighboring point, triangulating a surface and processing the outside triangles on the boundary). The last section is our results, discussion and evaluation.

5.2 Related work

In this section, we summarize the techniques for triangulating a surface of point clouds on both 2D and 3D. Various methods for meshing a triangular surface of a 3D point set have been proposed in [Hop94, BMR+99, Ede01, LTW04, YLL+07, BTD08, Ma11]. Other existing approaches for building a triangular mesh of 2D points have been suggested in [FP93, Slo93, Kle97, YZY+06, DZ08]. Most of these methods are classified into three following categories:
Contour tracing approach (or called implicit surface approach) is a method which produces approximating rather than interpolating surface. Most contour tracing algorithms divides space into cubes or tetrahedrons, evaluate the scalar function at the vertices of these volume elements; and for each element, from the values at its vertices, infer a linear approximation to the surface [DWLT90, Hop94]. For surface reconstruction, implicit surface provides a compact representation of a surface that can allow smooth approximation to the data points [FXC07, Cha07]. The main advantage is the ability to test if a point is inside or outside of the surface based on a scalar function as the zero set of a function for defining an implicit surface.
Region growing approach always starts from a seed triangle in a local region of a point set; then considers a new point in the existing region boundary to connect and create a new triangle. The process will stop until all points have been touched. Unlike the implicit surface method, the region growing approach takes every point of a point cloud as the reconstructed triangle mesh vertex (interpolating all points in the point cloud). Therefore, they will keep the most details of the original surface of physical objects and the reconstructed surfaces are expected to be more accurate [Ma11].
Sculpting-based approach is also called Delaunay-based approach. In other words, the Delaunay-based approaches sculpt the Delaunay triangulation of the sample points; and Delaunay triangulation is used in sculpting based methods.
In the next section, we review some existing methods for meshing a surface with 2D and 3D triangulations. Each algorithm intends to build a mesh from a point cloud with some given properties.

5.2.1 The methods in 2D

2D triangulation is known as a method designed to handle planar triangulations. It is based on a triangulation data structure in 2D and embedded in a plane. The plane of the triangulation may also be embedded in a higher dimensional space. In order to preserve the initial geometry of the surface as well as the topology of a triangular mesh, the existing methods aim to preserve the important properties of a triangular mesh (i.e. the triangles

are "as equilateral as possible"). They are studied to find the best triangulation, where "best" is evaluated according to some specific quality measures and criteria such as Delaunay triangle, maximum angle, minimum angle, maximum edge length, or total edge length of a triangle on the surface [BP00].
Fang et al (1993) [FP93] proposed a method for triangulating the 2D data points based on a uniform grid structure. At first, 2D data points are structured in a 2D uniform grid G; then, the authors built an algorithm for a planar triangulation consist of three steps. In the first step, they find the first point p_1 in the middle of grid G; then, a closest point p_2 to p_1 is chosen and the first edge $e(p_1, p_2)$ is then created. In the second step, they find a point p_3 such that the angle $\angle(p_1p_3p_2)$ is a maximum one and the triangle $\triangle(p_1, p_2, p_3)$ satisfies the Delaunay criterion. After creating the first triangle T_1, they insert all edges of T_1 into the edge-list. In the last step, the process is started from each edge of T_1 to create T_2 and repeated the second step until empty the edge-list. This method triangulates the convex hull of the surface (see figure 5.1).

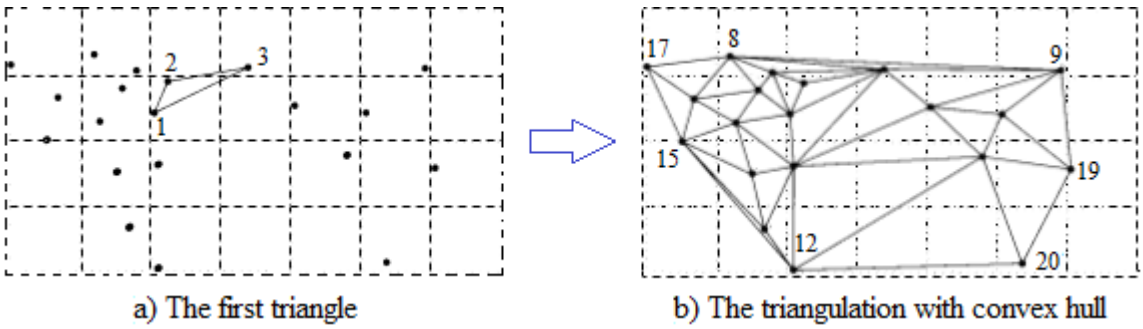

Figure 5.1: 2D Delaunay triangulation by using a uniform grid [FP93].

As we known, Delaunay triangulation DT is an elementary method applied to build a triangular surface [BP00, Ede01, Mau02]. We based on the properties of the DT as a criterion for triangulating the surface in order to preserve the shape of a triangular mesh. However, in some cases, we cannot apply absolutely the properties of conforming DT; even we may violate some these properties, but it is not affected to the global triangulation of the surface. At that time, the method that can be applied is called CDT (Constrained Delaunay Triangulation) [Slo93, Kle97, YZY+06]. The CDT involves a set of edges and points while still maintaining most of the favorable properties of DT.
Domiter et al (2008) [DZ08] introduced a method for 2D constrained Delaunay triangulation built upon sets of points and constraining edges. The method uses a sweep-line paradigm combined with a Lawson's legalization (see figure 5.2). The sweep-line moves following the vertical axis. The hitting point p_i (p_{11}) is projected on the advancing front of a line segment e_i (p_8, p_9). A new triangle between p_i and two endpoints of e_i is formed (p_9, p_{11}, p_8) if it satisfied the criterion of an empty circle passed through these three points; otherwise, swap an edge in order to preserve the Delaunay triangle's properties (see figure 5.2d). After that the advancing front is updated and the process is repeated until finishing

the surface triangulation. This algorithm simultaneously triangulates points and constraining edges. Thus avoid consuming location of those triangles containing constraining edges, and improving the processing time. However, during the triangulation of a surface, the method has to check the Delaunay criterion for flipping an edge in order to obtain an optimal triangular surface.

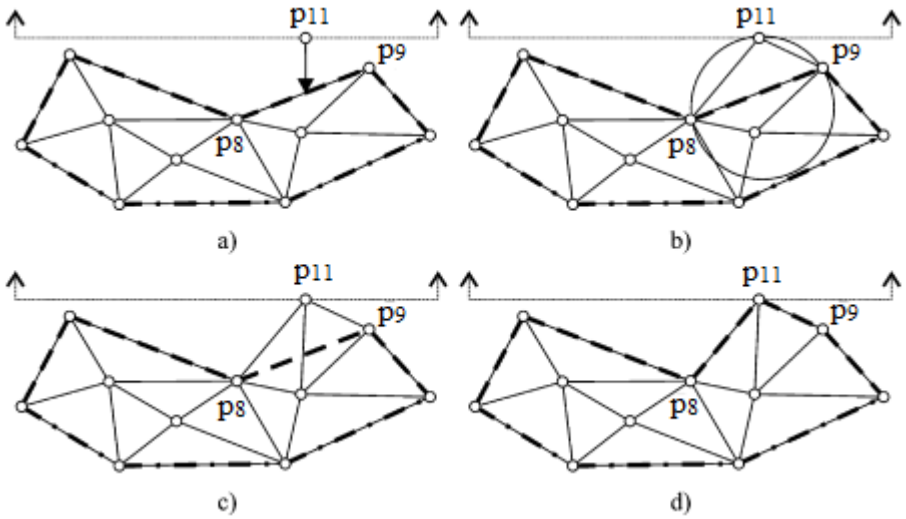

Figure 5.2: Sweep-line *DT* triangulation by using an advancing front [DZ08].

5.2.2 The methods in 3D

In 3D, many methods based on the computation of an approximate Delaunay mesh. At first, a tetrahedral volume is computed from the data point set. Then, it removes the convex hull parts to extract the original shape. Other approaches computes an implicit surface or based on the region growing technique.
Kazhdan et al (2006) [KBH06] suggested a method for surface reconstruction from the oriented points of 3D point clouds based on the Poisson formulation. The method falls into the category of contour tracing approach using an implicit function. The goal is to reconstruct a watertight triangulated approximation to the surface by computing approximately of an indicator function from the samples. The authors compute a 3D indicator function χ (defined as 1 at points inside the model, and 0 at points outside: see figure 5.3), and then obtain the reconstructed surface by extracting an appropriate isosurface. The method considers all points at once, without restoring the heuristic spatial partitioning or blending. Therefore, this is a global solution for the Poisson reconstruction and viewed as an advantage of this method. Besides, it can create a very smooth, robust surface, and adapt to the noisy data. Nevertheless, in some cases the limitation of this method is connecting the regions without data points.

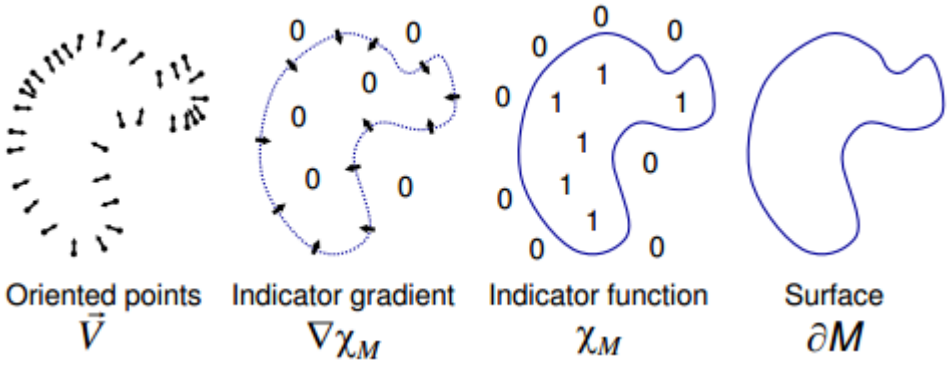

Figure 5.3: Intuitive illustration of Poisson reconstruction in 2D [KBH06].

The proposed method of Bernardini et al [BMR+99] (called Ball Pivoting Algorithm: BPA) is considered as an example of the region growing technique. In this method, an essential point with ball pivoting is that it is an approximation of the alpha shapes of the point cloud (see figure 5.4). For each three points in a 3D point set, they will form a triangle if a ball of a user (with a specified radius ρ) touches them without containing any other points. Starting from a seed triangle, the ball pivots around an edge while keeping in contact with the edge's endpoints until it touches another point, and forms a new triangle. The process continues by rolling the ball over a set of points until all reachable edges have been tried. For each forming a new triangle, this new triangle becomes a seed triangle for the next operating until all points have been processed. This method can process a large number of a 3D point cloud, while using a small amount of required memory, and the processing time is efficient. The mesh generating can fill the small holes that are smaller than the ball radius. This method provides topologically correct mesh, model's scenes of any geometric type and size. However, one of the disadvantages is that the ball radius ρ does not adapt to local point density; many holes may be generated during the meshing; and the quality of the mesh depends on the radius ρ defined by user.

Hong-Wei et al (2004) [LTW04] proposed a method to generate a triangular mesh based on an intrinsic property of a 3D point cloud. The algorithm is called IPD (intrinsic property driven) and fall into the category of region growing approaches. Starting from a seed triangle; a new triangle is created from each edge (called active edges: e_{ij} in a queue) of the seed triangle, by selecting a new point k in the influence region of e_{ij} to form a new triangle (i, j, k). In each iterative operating, the newly reconstructed edges are added to the queue of e_{ij}, and k is selected from the weight of minimal length criterion. Besides, the normal vector of each triangle is also checked to determine and adjust its direction. The process is repeated until the queue of e_{ij} is empty. The advantage of this method is that using the weight of the minimal length criterion adapted to the local sampling density for choosing point k. Therefore, this way overcomes the user's specified ball-radius in the BPA method [BMR+99]. This method guarantees the constructed surface approximated the ini-

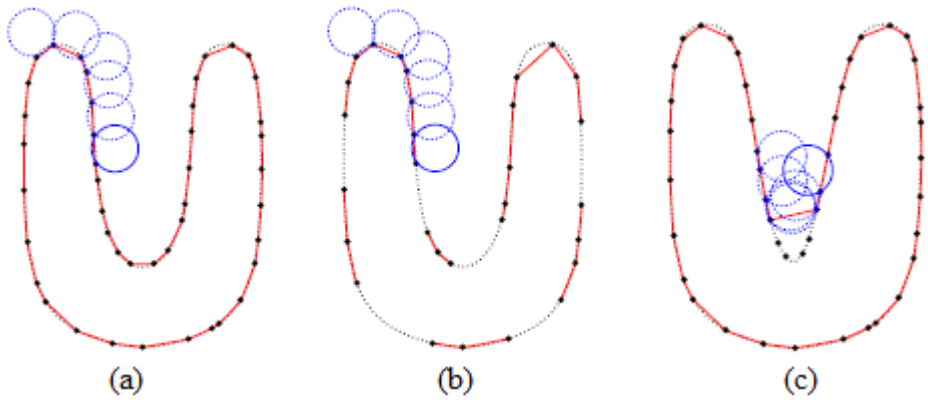

Figure 5.4: A 2D illustration of BPA algorithm [BMR+99]: a) a reconstructed curve connects the points using a circle of radius ρ pivots; b) the sampling density is too low to create the edges with the user-specified radius ρ; c) the user-specified radius ρ is too large to reconstruct some high curvature regions.

tial surface, with small topological errors. However, each time to create a new triangle, the computation is complex. For each active-edge e_{ij}, all boundary faces in 3D (surround e_{ij}) are first computed to determine the influence region. Then, all neighboring points of e_{ij} are checked based on the harmonic map function [EDD+95] to select a new point and minimize distortion of the surface (see figure 5.5). Therefore, the method leads to a costly computation.

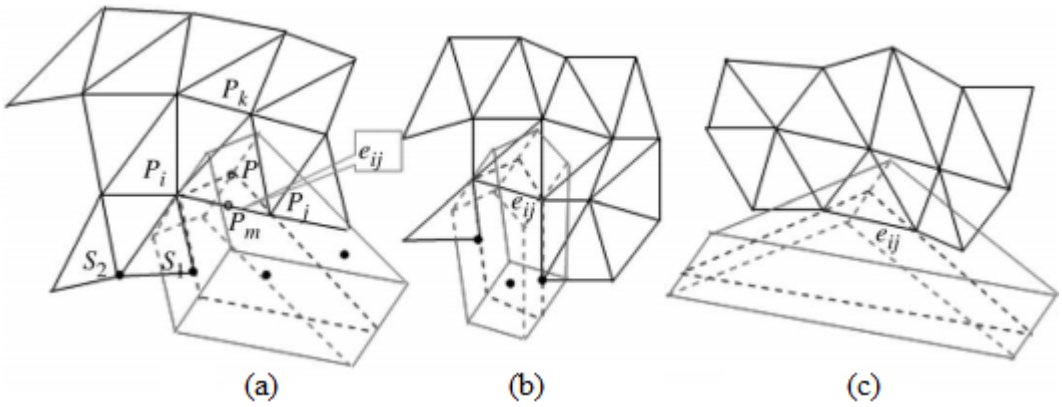

Figure 5.5: Reconstructing a triangular mesh based on IPD [LTW04].

Zoltan et al (2009) [MRB09] presented a method for a fast surface reconstruction of a large noisy point cloud. The method based on the category of surface growing used a robust triangulation algorithm to recreate the underlying surface's geometric properties. Starting from the first triangle FT, the next triangle NT is created by connecting from each edge of FT to a neighboring point. After that, the process is repeated from NT until all points are checked and no more valid triangles can be created. This method works directly on a

3D point cloud and uses the nearest neighbor search based kd-tree. The search for a neighboring point of each point p is selected in a sphere with radius $r = \mu.d_0$ that adapts to the local point density (d_0 is the distance of p to its closest neighbor, and μ is a user-specified constant). Therefore, the lookup of edges is optimized. For each point, the triangulation always grows to the front; thus, it can save the memory because no need to store all the triangles in the entire data set. However, to process the noisy point, a principal component analysis is used to estimate the triangle normal vectors. The covariance matrix is computed by adding a weight of the Gaussian function: $\xi_i = exp^{-\left(d_i^2/\mu^2\right)}$ for point p_i of the neighborhood. As a result, the extra step is more expensive in computation.
The above methods can be applied to triangulate a 2D or 3D point cloud, with or without organization. They lead to a very good triangular mesh, which approximates and closes to the original model of the input data. As we mentioned in the previous chapters, our data are organized in the 3D grid in voxels. Therefore, we took this advantage for building a method to obtain a faster computation. We detail our method in the next section.

5.3 Methods for triangulating the surface

5.3.1 Overview

In this section, we describe our method for triangulating an elevation surface structured in a 3D grid, based on the 2D Delaunay triangulation of the projected point cloud. As presented in chapter four, we have simplified this surface. After simplification, the point distribution is constrained (with respect to its density) from the boundary to the inside of the surface. Moreover, a large number of points has been removed while keeping the initial shape of the surface. This is one of the important steps for further triangular surface processing, because part of computation in the algorithm depends on the number of input data points. Starting from the properties of 2D Delaunay triangle, we applied a fast voxel traversal search [AW87] to develop our method. At first, the 3D cloud of points is projected onto a natural 2D grid in the x, y plane (see figure 5.6). Then, we triangulate the surface (actually, we compute a Delaunay triangulation of the 2D point cloud taking advantage of its regular structure). The main novelty of our approach is that the neighboring points are searched in a square box supported by the edge e_i under consideration.

Therefore, our method consists of three steps. In the first step, we create a first Delaunay triangle. Starting from the first Delaunay boundary edge, we find and connect to a neighboring point to create a first triangle. In the second step, we create the next triangle, adjacent to the first triangle based on a criterion of the Delaunay triangle. In the third step, we triangulate the surface by repeating the process from the next triangle. After creating the next triangle, this triangle will become a first triangle for the next iteration.

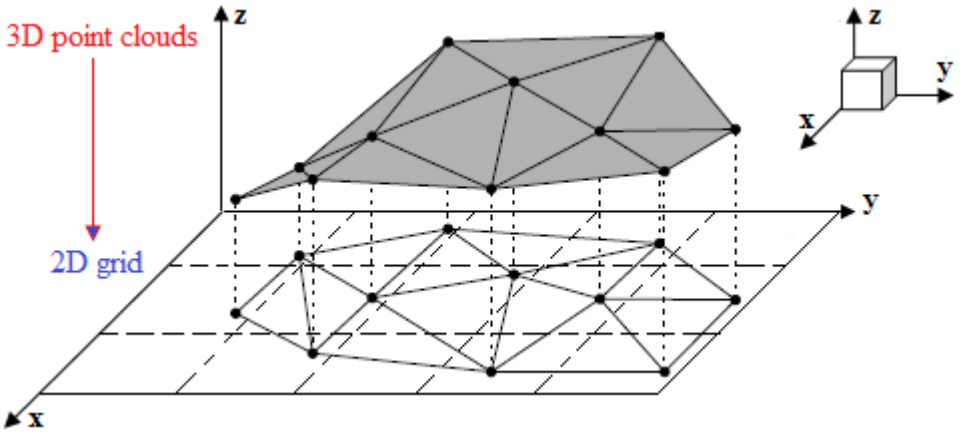

Figure 5.6: Triangulation of a 3D point cloud by projecting it onto the 2D grid.

5.3.2 Concepts, notation and definition

In order to clarify the context, let us introduce some concepts and notations used in the sequel. In chapter three, we have determined the boundary of the surface. It is a polyline connecting the boundary edges $e(v_i, v_{i+1})$ with the numbering of the polyline. As we also mentioned in our context, each point belongs to exactly one pixel. Therefore, we will freely use either the continuous or discrete notations: (see figure 5.7)

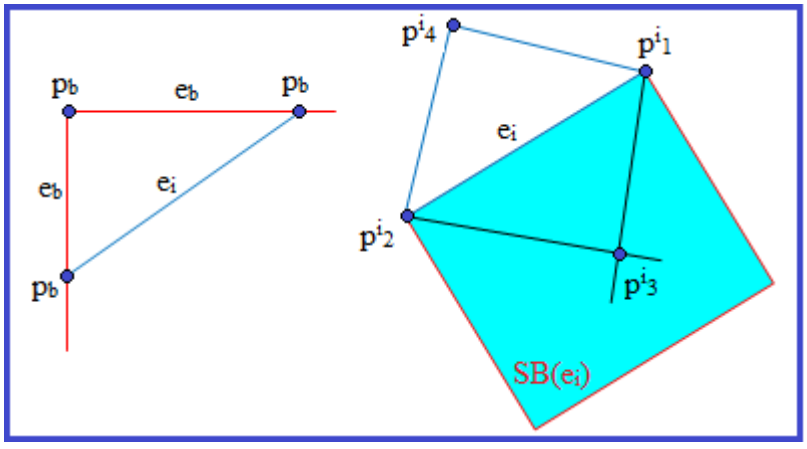

Figure 5.7: Some concepts and notations used in the surface triangulation

- Given an edge $e = (p_1, p_2)$, there exist exactly two pixels v_1, v_2 such that $p_i \in v_i$ $(i = 1, 2)$.
- p_b: a boundary point.
- e_b: a boundary edge, $e_b = (p_{b1}, p_{b2})$.
- The continuous size of e is, as usual given by $\|p_2 - p_1\|$, we will denote it by $\|e\|$.

- The discrete size of e (denoted by $|e|$) as the discrete distance between v_1 and v_2 ($\|v_2 - v_1\|_\infty$ in chapter 3).

- Given a triangle (p_2^i, p_1^i, p_3^i) and the edge $e_i = (p_1^i, p_2^i)$, a point p_4^i is called opposite to p_3^i with respect to e_i if it belong to a vertex of a triangle $\triangle(p_1^i, p_2^i, p_4^i)$ and opposite to a neighboring point p_3^i by edge e_i (see figure 5.7).

- $k = |e_i|$: the discrete size of e_i.

- $SB(e_i)$: a square box, supported by e_i.

As we mentioned in the related work, in the special case of the surface: if we cannot apply the properties of a Delaunay triangle to triangulate the surface, we can apply the case of constrained Delaunay triangulation CDT. Let us introduce the definition of a visible vertex [LL86] and thereafter we apply to define the CDT in our case:
Given a set of point together with a set of edges E, a CDT is the triangulation of the point set closest from the Delaunay triangulation but containing the edges of E.

Definition 5.1. *Constrained Delaunay boundary edge $CDBE$:*

- *Visible vertex: Two points (p, p') are Delaunay visible from each other if the segment between the two points intersects no edges (see figure 5.8a, where p and p' are not visible together).*

- *Constrained Delaunay Criterion: The circumcircle C of a given three points of a $\triangle(p_1, p_2, p_3)$ does not contain any points visible from p_1, p_3 or p_2 (see figure 5.8b). This definition has to be compared with the Delaunay criterion: the circumcircle C of any triangle $\triangle(p_1, p_2, p_3)$ is empty.*

- *Constrained Delaunay boundary edge ($CDBE$): A boundary edge $e_b(p_{b1}, p_{b2})$ is a $CDBE$ if and only if both p_{b1}, p_{b2} are boundary points, and there exist an inner point p such that the circumcircle of $\triangle(p_{b1}, p_{b2}, p)$ does not contain any inner points but may contain the outer points that are not visible from p. (see figure 5.8c)*

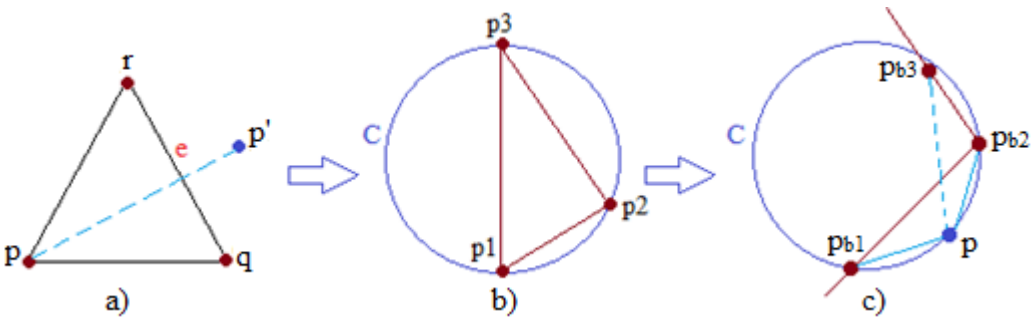

Figure 5.8: A case of $CDBE$

Our exterior boundary is defined in chapter three, each point is obtained as the first and closest clockwise neighbor of the previous one. Therefore, all boundary edges on the boundary of the convex hull of the surface are Delaunay edges (as proved in [LS80]). However, there are some boundary edges on the boundary in the concave parts of the surface which can be non Delaunay. An illustration is described in figure 5.8c, whereas p_{b1}, p_{b2}, p_{b3} are boundary points, but the boundary edge $e_b(p_{b1}, p_{b2})$ is a non Delaunay (it is a case of $CDBE$). In the section "Building a seed triangle", we will detail how to find the first edge to triangulate the surface. An overview of our method is presented in figure 5.9.

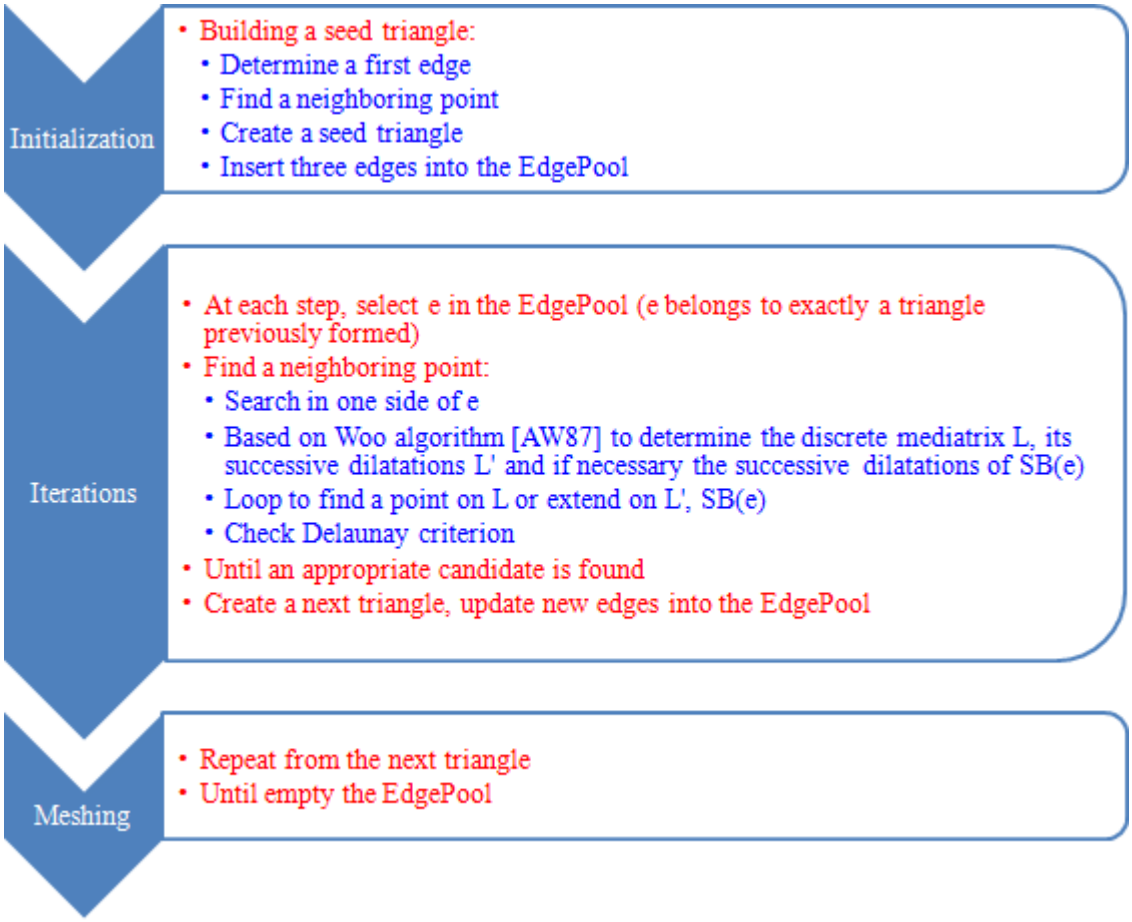

Figure 5.9: The general diagram of our method

5.3.3 Building a seed triangle

In this section, we describe our method for building a first triangle. The existing methods for creating a first (seed) triangle are introduced in [FP93, LTW04, Kle97, BMR+99]. Normally, most of these methods always start from the first point p_i (it can be a random choice, a middle point or a point with a special value of its coordinates in a point cloud);

then a point p_j close to p_i is chosen in order to create the first edge e_{ij}. The next step consists in finding a neighboring point p_k such that p_k satisfies the condition of a minimum distance to e_{ij} or an empty circle which passed through three points p_i, p_j, p_k. This triangle $\triangle(p_i, p_j, p_k)$ is then considered as a seed triangle.
In the algorithm of Boris et al (2004) [MVF04] for triangulating a surface, they proposed a method to create a seed triangle as follows: they first select a point $p \in \Re$ ($\Re$: set of 3D points) and determine its nearest neighbor $q \in \Re$ (these two points defined the first border edge, pq). Then, they find the point $r \in K(p)$ (where $K(p)$: k-nearest neighbors of p) that maximizes the angle $\angle prq$. If the triangle $\triangle(p, q, r)$ satisfies the condition of a Delaunay triangle, it is considered as a seed triangle.
In our case, we start from the left-most boundary point p_{b1}; next, we find a closest neighboring boundary point p_{b2} on the boundary. If the boundary edge $e_b(p_{b1}, p_{b2})$ is not a Delaunay edge, it is a case of $CDBE$. Then, this boundary edge $e_b(p_{b1}, p_{b2})$ is determined as a first edge. The next step, we find a neighboring point p_3^i of e_b such that p_3^i must satisfy the Delaunay criterion. Therefore, the triangle $\triangle(p_{b1}, p_{b2}, p_3^i)$ is our seed triangle T_{first}. In the next section, we give a proof that a Delaunay edge has always been existed in our data:

Property: Let p_1 is a point belonging to a set of vertices V ($p_1 \in V$). Let p_2 be the closest neighbor of p_1. The edge p_1p_2 is a Delaunay edge.

Proof. We orient edge p_1p_2 from p_1 to p_2 (see figure 5.10a); H is a half space on the right side of p_1p_2. The vertices in H are sorted by an increasing order of angles ($v_i \rightarrow \alpha_i$). We choose v_i with the largest angle $\widehat{p_1\alpha_i p_2}$ at p_3 such that the intersection between circum$(p_1p_2p_3)$ and H is empty. As p_2 is closest from p_1, $\|p_1p_3\| \geq \|p_1p_2\|$; and the arc$(p_1, p_2, p_3)$ is larger than the half circle $\Leftrightarrow \widehat{p_1p_2p_3} < 90^o$. If there is a point p_4 belong to circum$(p_1p_2p_3)$, on the left of p_1p_2 (see figure 5.10b); then, $\|p_1p_4\| < \|p_1p_2\|$. This is impossible because p_2 is the closest point to p_1. Hence, there is no such point p_4; and p_1p_2 is a Delaunay edge. □

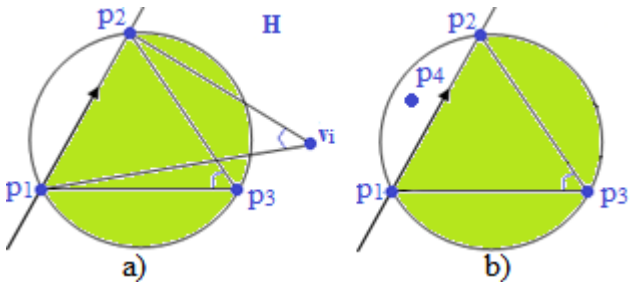

Figure 5.10: Determination of the first Delaunay edge.

After building the seed triangle T_{first}, we insert its three edges into the EdgePool for the next iteration step. The next section, we describe the condition of non-intersection for generated triangles in the triangular mesh.

5.3.4 Searching conditions in one side of an edge

For each edge e_i of the previous triangle (e.g. $\triangle(p_1^i, p_2^i, p_4^i)$), we need to find a neighboring point p_3^i to form a new adjacent triangle $\triangle(p_2^i, p_1^i, p_3^i)$ (see figure 5.11). In order to avoid creating intersecting triangles, we require the following conditions for p_3^i. The first condition is that the three points (p_1^i, p_2^i, p_3^i) are not aligned. The second condition is that p_3^i and p_4^i must be on either side of e_i. More precisely, we require that p_3^i and p_4^i satisfy the following relation:

$$\left[(p_4^i - p_2^i) \times N\right] . \left[(p_3^i - p_1^i) \times N\right] < 0 \tag{5.1}$$

Where:

- $N = Oz$: normal to the triangles $\triangle(p_1^i, p_2^i, p_4^i)$ and $\triangle(p_2^i, p_1^i, p_3^i)$ (actually our triangles are 2D, but we map them to the Oxy plane of R^3 for further computations).
- $\times$: the cross product operation.

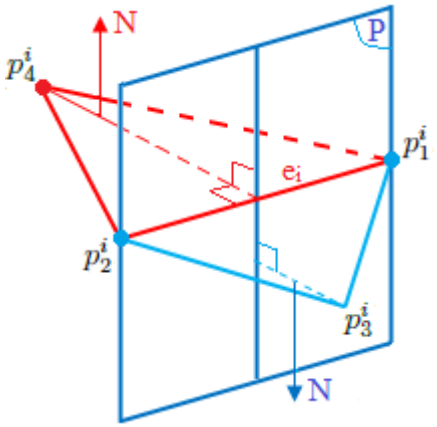

Figure 5.11: Condition of non-intersection for generated triangles.

5.3.5 Delaunay Criterion

In this section, we describe a fast way to compute the Delaunay criterion (testing the emptiness of a circumcircle). For each e_i, we find an optimal neighboring point p_3^i in one side of $e_i = (p_1^i, p_2^i)$ to form a triangle $T(p_1^i, p_2^i, p_3^i)$. Then, we check the emptiness of the circumcircle C going through p_1^i, p_2^i, p_3^i. If C is empty, then T satisfies the criterion of the Delaunay triangle. In order to check the criterion for T, we compute the following steps:

- Compute the length of the three sides: a, b, c of the triangle T;
- Compute the radius r of the circumcircle of T: $r = (a * b * c)/4 * area(T)$;
- Compute p_c, the circumcenter of the triangle T based on the equation of the perpendicular bisectors between a pair of edges of T;

- Based on r and p_c, we loop over the points in the bounding box of C to check if one of them (let's call it p) belongs to C.

We call Circumcircle(p_1^i, p_2^i, p_3^i, p) the function testing wether $p \in Circum(p_1^i, p_2^i, p_3^i)$ or not. This predicate function is computed by evaluating a test on the determinant of following 4x4 matrix:

$$Circumcircle(p_1^i, p_2^i, p_3^i, p) = \left(Det \begin{bmatrix} x_{p_1^i} & y_{p_1^i} & x_{p_1^i}^2 + y_{p_1^i}^2 & 1 \\ x_{p_2^i} & y_{p_2^i} & x_{p_2^i}^2 + y_{p_2^i}^2 & 1 \\ x_{p_3^i} & y_{p_3^i} & x_{p_3^i}^2 + y_{p_3^i}^2 & 1 \\ x_p & y_p & x_p^2 + y_p^2 & 1 \end{bmatrix} > 0 \right) \tag{5.2}$$

The repeated computation of a determinant of 4×4 matrix in a huge amount of 3D point clouds is very slow (expensive). Therefore, observing that the previous determinant can be decomposed into constant terms (depending only on p_1^i, p_2^i, p_3^i) and combined according to the point p, we pre-compute the following four sub-determinants:

$$SubDet1 = \begin{bmatrix} y_{p_1^i} & x_{p_1^i}^2 + y_{p_1^i}^2 & 1 \\ y_{p_2^i} & x_{p_2^i}^2 + y_{p_2^i}^2 & 1 \\ y_{p_3^i} & x_{p_3^i}^2 + y_{p_3^i}^2 & 1 \end{bmatrix} \tag{5.3}$$

$$SubDet2 = \begin{bmatrix} x_{p_1^i} & x_{p_1^i}^2 + y_{p_1^i}^2 & 1 \\ x_{p_2^i} & x_{p_2^i}^2 + y_{p_2^i}^2 & 1 \\ x_{p_3^i} & x_{p_3^i}^2 + y_{p_3^i}^2 & 1 \end{bmatrix} \tag{5.4}$$

$$SubDet3 = \begin{bmatrix} x_{p_1^i} & y_{p_1^i} & 1 \\ x_{p_2^i} & y_{p_2^i} & 1 \\ x_{p_3^i} & y_{p_3^i} & 1 \end{bmatrix} \tag{5.5}$$

$$SubDet4 = \begin{bmatrix} x_{p_1^i} & y_{p_1^i} & x_{p_1^i}^2 + y_{p_1^i}^2 \\ x_{p_2^i} & y_{p_2^i} & x_{p_2^i}^2 + y_{p_2^i}^2 \\ x_{p_3^i} & y_{p_3^i} & x_{p_3^i}^2 + y_{p_3^i}^2 \end{bmatrix} \tag{5.6}$$

The 4×4 determinant and associate test are then computed as follows:

$$Circumcircle(p_1^i, p_2^i, p_3^i, p) = [x_p \times SubDet1 - y_p \times SubDet2 + (x_p^2 + y_p^2) \times SubDet3 - SubDet4] > 0 \tag{5.7}$$

Therefore, we use the algorithm (Algorithm 5.1) to check a points p belong to C or not:

Algorithm 5.1 Circumcircle(p_2^i, p_1^i, p_3^i)

compute: a, b and c, three sides of $\triangle(p_2^i, p_1^i, p_3^i)$
compute: r, radius of circumcirle;
compute: p_c, the circumcenter;
compute: $SubDet1, SubDet2, SubDet3, SubDet4$ extending from equation (5.2);
for each point p in bounding box of C of $\triangle(p_2^i, p_1^i, p_3^i)$ **do**
    if satisfies equation (5.7) **then**
        $p \in Circumcircle(p_2^i, p_1^i, p_3^i)$;
    else
        $p \notin Circumcircle(p_2^i, p_1^i, p_3^i)$;
    end if
end for

5.3.6 Neighboring points search based on the voxel tracing

Our goal is to triangulate a surface such that all its triangles are as equilateral as possible. Therefore, the neighboring point p_3^i will be searched as close as possible to the position of the third vertex of the equilateral triangle lying on e_i. We apply the voxel traversal algorithm of Woo introduced in [AW87] to search a neighboring point p_3^i of e_i within a square box $SB(e_i)$, size k (we denote by k the discrete size of e_i (that is, as explained previously, the discrete distance between the pixels containing the extremities of e_i): see figure 5.12).

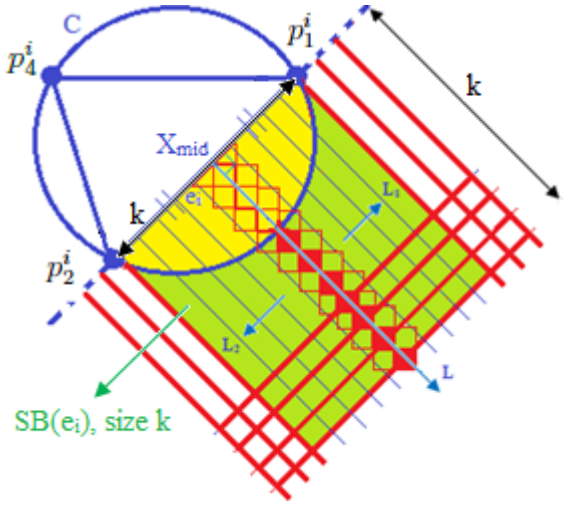

Figure 5.12: The voxels traversal search

We use the following notations (illustrated on figure 5.12) to denote and compute in our algorithms:

- Woo algorithm will be described in a very generic way (with indices i ranging in $\{1..n\}$ n being the dimension). In our case, $n = 2$.
- $X[i]$ with $i \in \{1..n\}$: coordinates of the current voxel.
- L: the discrete mediatrix of e_i up to a distance k of X_{mid} (the voxel in the middle position of edge e_i).
- L': the successive dilatations of L.
- $Step[i]$ with $i \in \{1..n\}$: for each coordinate, direction of L (+1 if the positive direction along the i^{th} axis, -1 otherwise) will indicate the coordinates of next point along each axis.
- $tMax[i]$ with $i \in \{1..n\}$: time at which the ray crosses the output plane along the i^{th} axis.
- $tDelta[i]$ with $i \in \{1..n\}$: time to cross a voxel along the i^{th} axis.

For each edge e_i, we use the Woo algorithm [AW87] to determine L under the form:

$$U + t.V \quad (t \in \mathbb{R}^+) \tag{5.8}$$

where:

- U: the initial point of L; in our case, $X_{mid}(x_{mid}, y_{mid})$;
- V: the direction of L (orthogonal to e_i and oriented opposite to (p_2^i, p_4^i).
- t: the time.

The next section, we will compute the discrete mediatrix L and the successive dilatations L' in order to find an optimal neighboring point p_3^i of e_i.

5.3.6.1 Computing the discrete mediatrix L

We execute Woo algorithm to compute the discrete mediatrix up to a distance k of X_{mid} (and starting from this point). At each step, given the current voxel $X(x, y)$, we decide whether to increase the coordinates of x or y or both x, y to find the next voxel. This set of voxels is stored in a list.
Initialization. We compute:

- X_{mid}: the initial voxel of L.
- V: we have $e_i = p_1^i - p_2^i = (x, y)$, let us set $v = (-y, x)$ the orthogonal direction and $v' = p_4^i - p_2^i$ and edge of the previous triangle. If $v \times v' > 0$, then we set $V = -v$ otherwise, $V = v$ (that is V is the direction of the half-line orthogonal to e_i and opposite with respect to the previous triangle (p_1^i, p_2^i, p_4^i)).

- For this voxel, we compute $tMax[i]$; $tDelta[i]$ and $Step[i]$ (which are static and will be constant along the algorithm).

Iterations. At each step:

- Let i be the axis such that tMax[i] is minimal
- Update X (by adding $Step[i]$ to its i^{th} coordinate)
- Update $tMax$

We use the algorithm (Algorithm 5.2) to compute the discrete mediatrix L as follow:

Algorithm 5.2 WooLine(e_i, k, rl)

```
1: //pointL: the endvoxel of L (size k from X_mid)
2: compute: X_mid, V;
3: compute: pointL;
4: compute: tMax[i], tDelta[i], Step[i];
5: repeat //start at X = X_mid
6:     rl.insert(X); //save current voxel
7:     for each dimension i do
8:         tMax[i]+ = tDelta[i];
9:         X[i]+ = Step[i];
10:    end for //X is next crossed voxel, that is our new voxel
11: until X = pointL
12: rl.insert(pointL);
```

Remark. *We start from the first current voxel $X = X_{mid}$. For each dimension (here $i = 1, 2$), we update $tMax[i]$ by adding the value of $tDelta[i]$. Therefore, the current voxel X is updated by adding $Step[i]$ to become the next voxel; then the next voxel is the new current voxel for the next iteration and inserted into the list rl. The iteration stop when the current voxel X hits the endvoxel $pointL$; and the last voxel $pointL$ is inserted into rl.*

For example, the iteration of Woo algorithm is described in figure 5.13. In this case, we can see in ray_1: $tMax[2] < tMax[1]$, that mean the next voxel will be computed by increasing the y coordinate of the current voxel $X(x, y+1)$. Similarly, in ray_2: $tMax[2] > tMax[1]$, we increase the coordinate x, so the next voxel will be $X(x+1, y)$. In the case of ray_0: $tMax[1] = tMax[2]$, it mean that the ray L crosses the output plane at the same intersection point of x and y coordinates; hence, the next voxel will be $X(x+1, y+1)$.

5.3.6.2 Computing the successive dilatations L'

All points lying on the discrete mediatrix of e_i are tested with the algorithm determined in section 5.3.5 in order to determine if they satisfy the Delaunay criterion. If none of

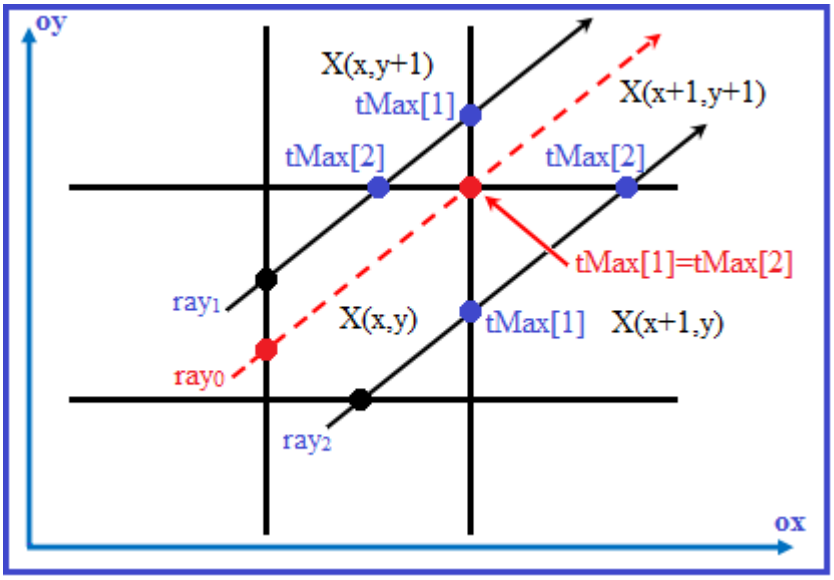

Figure 5.13: Iteration of Woo algorithm

them coincides, we compute the successive dilatations L' of the discrete line L and test the vertices they may contain. This dilatation is constituted of two discrete lines (called L_1 and L_2). The vertices on L_1 and L_2 are determined based on the neighboring vertices of each current vertex with its x, y coordinates $X(x, y)$ on L as follows: (see figure 5.14)

- In line L_1, the neighboring vertices $N_1(X(x, y))$ for each current vertex $X(x, y) \in L$ are determined:
  $N_1(X(x, y)) = \{X(x + 1, y), X(x + 1, y + 1), X(x, y + 1)\}$

- In line L_2, the neighboring vertices $N_2(X(x, y))$ for each current vertex $X(x, y) \in L$ are determined:
  $N_2(X(x, y)) = \{X(x - 1, y), X(x - 1, y - 1), X(x, y - 1)\}$

The successive dilatations $(L_1 \cup L_2)$ are explored until $SB(e_i)$ has been fully explored. In this case, if there is no point in $SB(e_i)$, we will explore the successive neighborhoods of $SB(e_i)$ (in one side of e_i, start from voxel $pointL+1$) until finding an appropriate point. In the next section, we will present our algorithm for searching a point based Woo algorithm.

5.3.6.3 Algorithm based on the voxel traversal search

In this section, we describe our algorithm based on the voxel traversal search. We note l, the list containing voxels crossed with the ray line L; similarly l_1, l_2 for L_1 and L_2 respectively. The algorithm includes the following two steps:

- First step: we compute the discrete mediatrix of e_i by calling the function $WooLine(e_i, k, rl)$ which return the list of voxels stored in rl; then, the list rl is assigned to l. For each

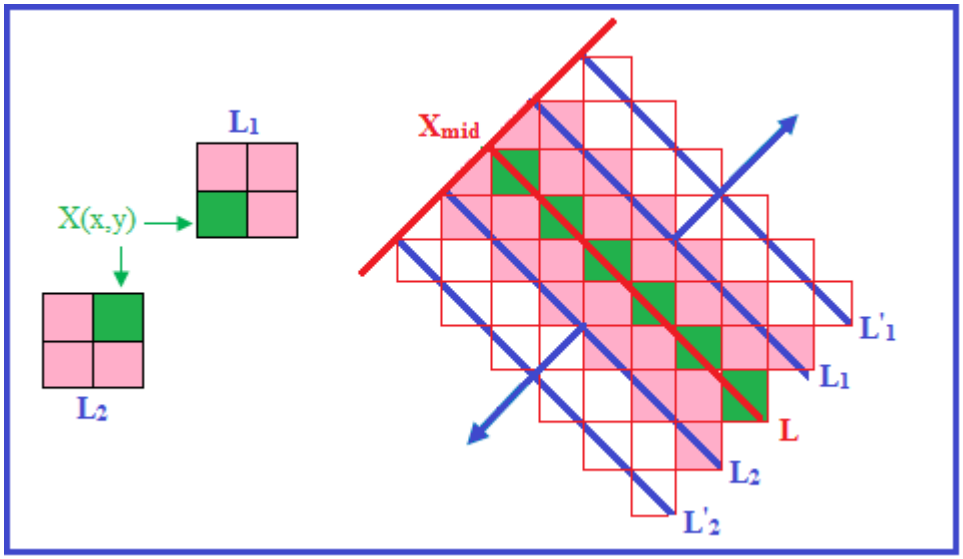

Figure 5.14: Computing the successive dilatations of the discrete mediatrix of e_i.

non-empty voxel $v \in l$ (i.e. a neighboring point: p_3^i), if it satisfies the Delaunay criterion ($Circumcircle(p_2^i, p_1^i, p_3^i)$), we stop searching. Otherwise, we go to step 2.

- Second step: we repeat the process on the neighboring voxels of l on l_1 and l_2 successively. If there is no point p_3^i on the square box of edge e_i, we enlarge the searching by iterative dilatations of $SB(e_i)$. The process will stop when a neighboring point p_3^i satisfying the Delaunay criterion is found.

The algorithm below (Algorithm 5.3) is used to find this neighboring point p_3^i based on the voxel traversal search. In this algorithm, each non-empty voxel $v \in SB(e_i)$ is considered as a neighboring point p_3^i.

Algorithm 5.3 SearchNeighbors(e_i, p_3^i)

```
1: find ← false, k=1;
2: while !find do
3:     WooLine(e_i, k, rl); //Step1: compute L.
4:     l_1 = l_2 = l ← rl;
5:     for each voxel v ∈ l do
6:         if v != empty and Circumcircle(e_i, v) == true then
7:             find ← true;
8:         else
9:             mark_v = true;
10:        end if
11:    end for
12:    while inside the SB(e_i) do  //Step2: compute L'
13:        compute l'_1, the neighbor voxels of l_1
14:        for each voxel v ∈ l'_1 do
15:            if v != empty and Circumcircle(e_i, v) == true then
16:                find ← true;
17:            else
18:                mark_v = true;
19:            end if
20:        end for
21:        compute l'_2, the neighbor voxels of l_2
22:        for each voxel v ∈ l'_2 do
23:            if v != empty and Circumcircle(e_i, v) == true then
24:                find ← true;
25:            else
26:                mark_v = true;
27:            end if
28:        end for
29:        l_1 ← l'_1; l_2 ← l'_2;
30:    end while
31:    dilatation from SB(e_i);
32: end while
```

5.3.7 Triangulating a surface

Following the previous analysis, we present in this section our algorithm for constructing a triangular mesh of an elevation surface defined by a sparse 3D grid. The corresponding algorithm (Algorithm 5.4) is used to triangulate a surface S as follow:

Algorithm 5.4 MeshGenerating(S)

start from the first Delaunay boundary edge;
create the first Delaunay triangle T_{first};
insert all edges of T_{first} into a list EdgePool;
while EdgePool != empty **do**
    for each edge e_i(endpoints: p_1^i, p_2^i) **do**
        if e_i != visited **then**
            SearchNeighbors(e_i, p_3^i);
            $T_{next} = \triangle(p_1^i, p_2^i, p_3^i)$;
            update the new edges of T_{next} into the EdgePool;
            e_i = visited;
            $T_{first} \leftarrow T_{next}$;
        end if
    end for
end while

Remark. *We start form the first boundary Delaunay edge to create the first Delaunay triangle T_{first}. Then, put all edges of T_{first} into a list EdgePool. Then, for each edge in the EdgePool, we choose a neighboring point p_3^i by calling the function SearchNeighbors(e_i, p_3^i) to create a new adjacent triangle T_{next}. Then, we update the EdgePool by inserting the new edges of T_{next}. After that, the triangulating process is repeated from T_{next} by assigning ($T_{first} \leftarrow T_{next}$) for the next iteration, until finishing the triangulating process (i.e. the EdgePool is empty).*

5.3.8 Processing the outside triangles on the boundary

The Delaunay triangulation previously presented produces a meshing of the convex hull of the initial set of points [FP93] (see figure 5.15a). Many triangles lie outside the boundary computed in chapter three. Hence, after triangulating the surface, it generates many undesired triangles which are outside the exterior boundary (see figure 5.15b). For this reason, we propose a method to delete the outside triangles on the boundary.

In order to determine the triangles outside the boundary computed at chapter three, we apply the ideas of [Kle97, DZ08], based on computing the angle α between each pair of consecutive edges on the boundary (e.g. two boundary edges: $e_{b1}(p_{b1}, p_{b2})$ and $e_{b2}(p_{b2}, p_{b3})$; angle α at p_{b2}). If $\alpha < \pi$, the triangle $\triangle(p_{b1}, p_{b2}, p_{b3})$ is considered as an outside triangle. This method of course requires computing before meshing, the boundary of the point set. In our case, the boundary has already been determined in chapter three. Therefore, in order to remove outer triangles, we just need to follow the boundary in the clockwise direction and delete outer triangles accordingly. Our method is described as follows:
We first determine the boundary triangles according to their vertices. For each triangle, if its three vertices are boundary points, it is considered as a boundary triangle. Following

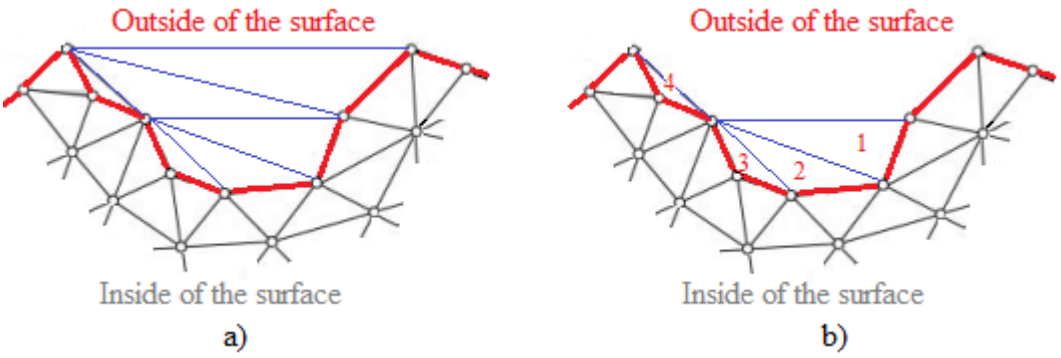

Figure 5.15: a) The convex hull of a triangular surface; b) The outside triangles of the boundary.

the boundary clockwise, we then check and delete the outward boundary triangles. A boundary triangle is an outward boundary triangle if the dot product between two vectors of this triangle (an inward normal vector of an edge and a vector of an outside edge) is negative (see figure 5.16).

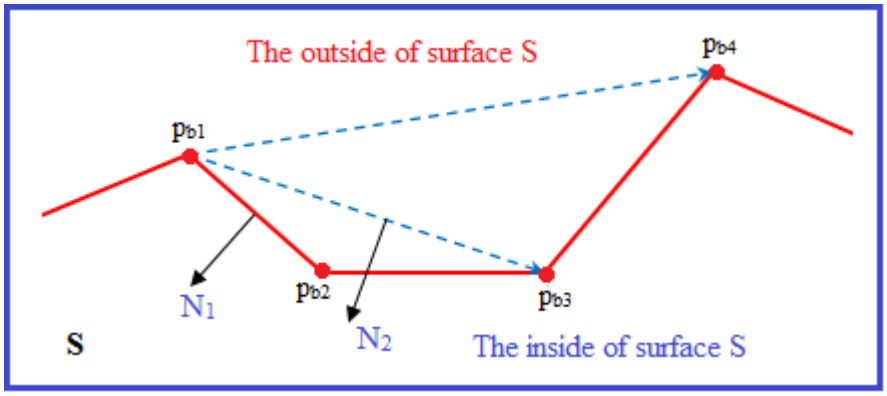

Figure 5.16: Determination of an outward triangle on the boundary.

Figure 5.16 is an illustration. The red line is a boundary line of surface S; $p_{b1}, p_{b2}, p_{b3}, p_{b4}$ are boundary points. We check if $((p_{b3} - p_{b1}).dot(N_1) < 0)$ then the triangle $\triangle(p_{b1}, p_{b2}, p_{b3})$ is an outward triangle of S; similarly, if $((p_{b4} - p_{b1}).dot(N_2) < 0)$ then the triangle $\triangle(p_{b1}, p_{b3}, p_{b4})$ is also an outward triangle of S.

5.4 Implement

In this section, we implement our proposed method for triangulating an elevation surface of 3D point clouds based on the voxel traversal search. Our algorithms are programmed and integrated in Meshlab [Cou13] as a plug-in. Therefore, we can test the running time

between our method and the existing method in the Meshlab (ball pivoting). However, it is difficult to evaluate exactly the comparison of the time between the methods (that depends on the similarity of algorithms and data structure between these methods). For this reason, we introduce and execute our adding method for finding a neighboring point p_3^i of e_i which can be applied in our context to compare exactly the processing time with the voxel traversal search. This method is based on computing the compactness of a triangle introduced in [Gue97, FB99, Tra08].
With the same idea (as we presented in the method voxel traversal search) in order to find the best neighboring point for creating a triangle as equilateral as possible, we used the formula of Guéziec [Gue97] for measuring the quality of triangles. We compute the compactness of a triangle in among triangles $\triangle(p_1^i, p_2^i, p_l)$ where $(l = 1..v_i)$ with v_i is the number of neighboring points in one side of $e_i(p_1^i, p_2^i)$ (see figure 5.17a). The formula is as follows:

$$C = \frac{4\sqrt{3}A}{l_0^2 + l_1^2 + l_2^2} \tag{5.9}$$

Where C is denoted the compactness of triangle T; A is area of T; l_0, l_1 and l_2 are respectively the three lengths of each edge of T. Similarly, $Pascal\ J.Frey$ [FB99] has also used a formula below to prove the shape quality of a planar triangle:

$$Q_K = \alpha \frac{S_K}{h_{max} P_K} \tag{5.10}$$

Where:

- Q_K: the quality of a triangle K.
- $\alpha = 2\sqrt{3}$: a normalization coefficient; $Q_K = 1$ for an equilateral triangle [GB97].
- S_K: area of triangle K.
- h_{max}: the length of the longest edge of triangle K.
- P_K: the half-perimeter of triangle K.

Instead of computing the discrete mediatrix L, its successive dilatations L' and if necessary the successive dilatations of $SB(e_i)$ to find the best neighboring point p_3^i (as we computed in the voxel traversal search); we now compute all triangles between $e_i(p_2^i, p_1^i)$ and its neighboring points p_l $(l = 1..v_i$, with v_i, the number of neighboring points: see figure 5.17a) based on equation 5.9 to find a point p_l of a compactness triangle within $SB(e_i)$. If this point is found, we then check the Delaunay criterion of triangle $\triangle(p_2^i, p_1^i, p_l)$; thereafter, we follow exactly the successive steps as presented in our voxel traversal search for triangulating the surface.
We implemented both methods (our voxel traversal search and computing the compactness) to test and compare the processing time between them. The obtained results are detailed in the next section.

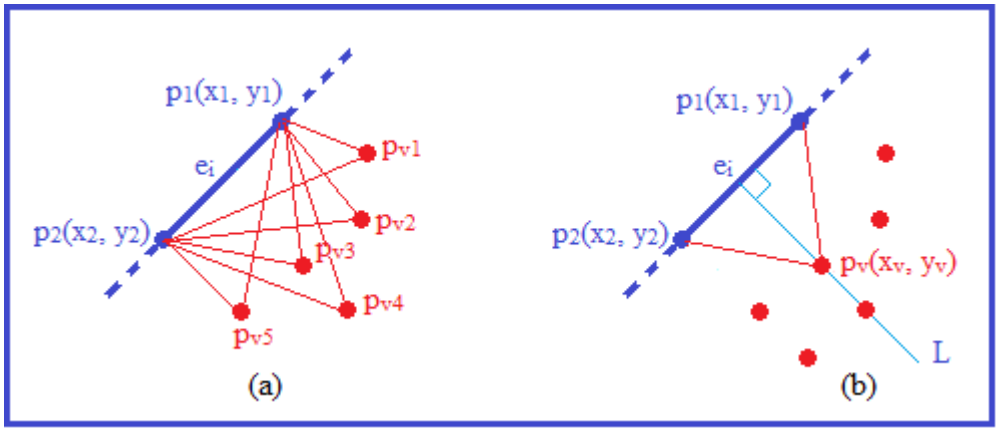

Figure 5.17: Searching a neighboring point: a) computing the compactness; b) voxel traversal search.

5.5 Results

In this section, we present some results of our method for triangulating an elevation surface. We compare the processing times between our method: "Voxel Traversal Search" (VTS), "Computing The Compactness" (CTC) and the existing method in Meshlab [Cou13]: "Ball Pivoting" (BP). We also compare the shape of the triangulated surfaces after previous simplifying step with our elaborate method (as presented in chapter 4). The processing time of our method is faster and as we will see, the initial shape of the surface is well preserved.

After triangulating the surfaces and processing their boundaries with our method, the total processing times are presented in table 5.1. In this table, we used the same input surfaces to test with the three methods. Both our neighboring searched methods (VTS and CTC) are integrated and tested (one by one) in the same algorithm for 2D Delaunay triangulation, on the same a computer. They are only different from their ways to find a neighboring point. The processing time is a bit different between them (see table 5.1). We will explain the reason why in the section 5.6 (Discussion and evaluation). The BP method generated a very good mesh, but the processing time is higher than our methods. If the number of input points increases, the running time to triangulate the surface is far from our method (see figure 5.18).

Input points	CTC		VTS		BP	
	Output faces	Time (ms)	Output faces	Time (ms)	Output faces	Time (ms)
15626	29871	14631	29871	13360	29990	18984
32402	61073	30911	61073	28984	58030	67871
60511	115828	68797	115828	63223	110982	225037
68956	125463	112117	125463	93246	118273	268276
98231	181851	194682	181851	162451	174582	859341
148317	266187	246435	266187	218713	267842	2017632
886639	1618624	1473188	1618624	1307466	1601158	11951403

Table 5.1: Comparison of the processing times between the methods. We use the same input data points and run on the same a computer (Intel 2CoreDue, 2GB of Ram).

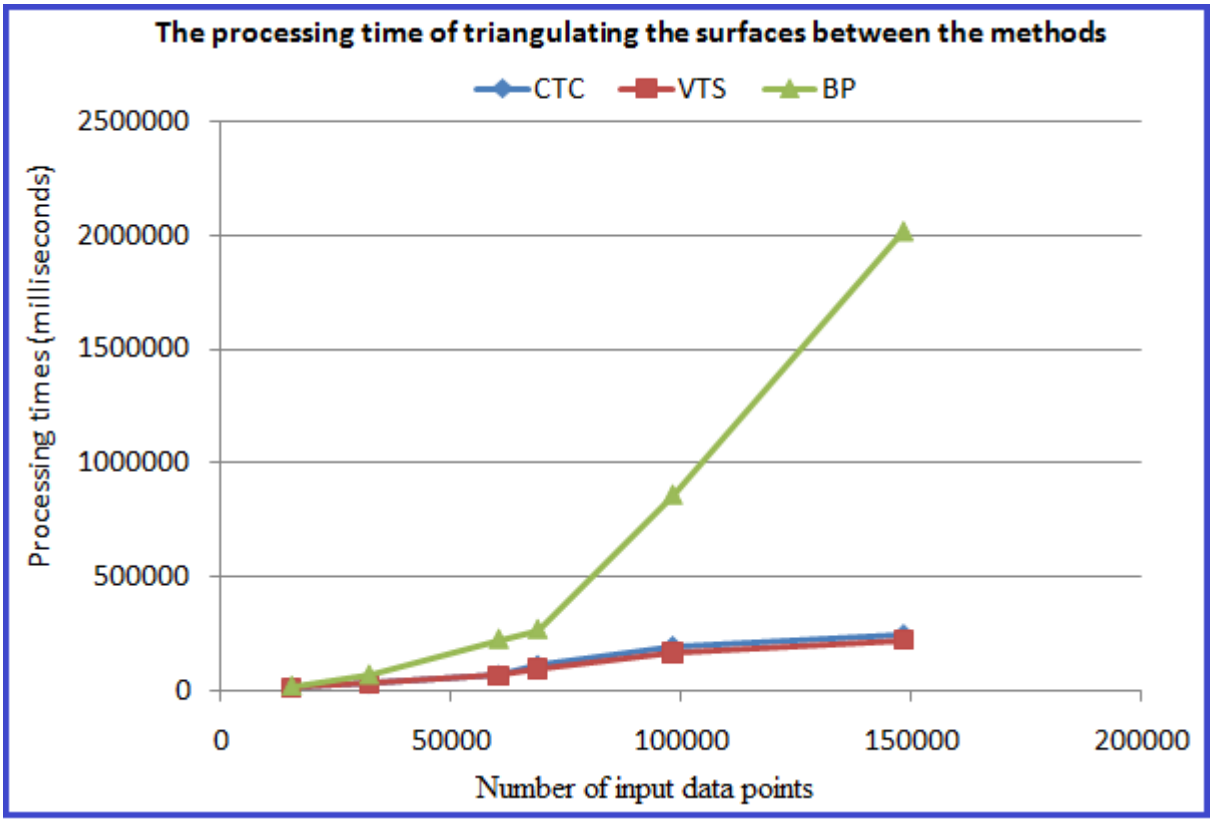

Figure 5.18: Comparison of the processing times between the methods: Computing the compactness (CTC); Voxel traversal search (VTS) and Ball pivoting (BP). On this graph, we do not plot the last example with 886639 points because it is too far from other examples and therefore spoils the graph.

We have processed the triangles on the concave parts of the boundary by removing some outward triangular faces of surface S (see figure 5.19). The exterior boundary of S and surface S have been processed in the previous chapters (chapter 3 and chapter 4). Therefore, the number of boundary points and the density of them depend on the previous processing and initial shape of the surface. For these reasons, the number of triangular faces is a bit different from the classical triangulation algorithms (in 2D Delaunay triangulation: the number of triangular faces is at most $2n-2-b$ triangles; where n is the number of vertices and b is the number of vertices on the convex hull [Wik13]).

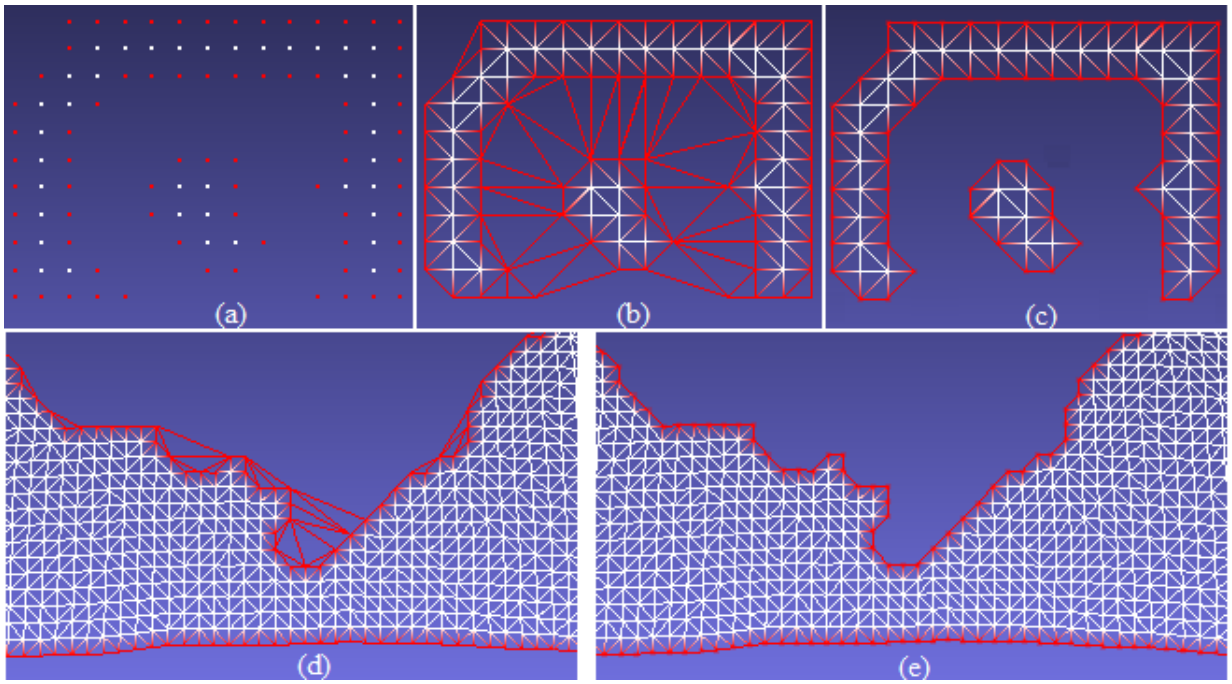

Figure 5.19: Processing the triangular faces on the boundary; (a) an input surface of 3D point clouds with boundary points (red color); (b) a triangular surface; (c) after removing the outside triangles; similarity, (d) before and (e) after deleting the outward triangles on the boundary.

In order to evaluate the quality of our triangulated surface (generated by our method) with respect to the initial shape of that surface, we compute the distance between two sampled-points surfaces ($S1$: the input surface of 3D point clouds; and $S2$: the output surface after simplifying and triangulating $S1$, as described in the section 4.4 of chapter 4). After triangulating the surfaces, the initial shape (e.g. curvature, bend, ridge, valley, groove) of these surfaces is well preserved (see figure 5.20 and 5.21).

In conclusion, the initial shape of the BP method is well preserved after triangulating while consuming the running time for the surface triangulation. Our method VTS obtained the good results on both processing time and approximation errors.

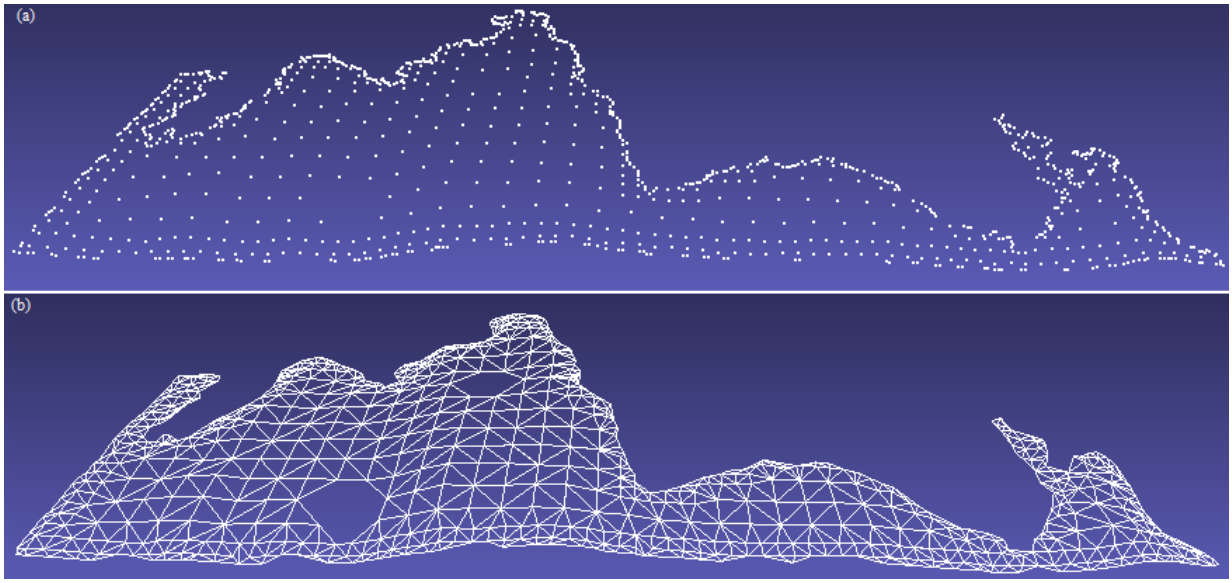

Figure 5.20: a) A geological surface of 3D point clouds (232 kb). b) After simplifying by using the elaborate method (cell size = 8, $\partial \leq 0.12$) and triangulating, the size of surface: 18 kb; the approximation error between (a) and (b) is Δ_{max}: 0.020; Δ_{avg}: 0.0006; the triangular faces vary in density from the boundary to the inside of the surface.

5.6 Discussion and evaluation

In this section, let us give some discussions and evaluations about the processing time of our method. We triangulate the surface based on the 2D Delaunay triangulation following the category of region-growing. Our method has two advantages compared to existing methods. The first and most important point is that we designed the square box and our particular search in order to speed up the Delaunay triangulation. The neighboring point p_3^i is always searched in one side of an edge e_i with two endpoints (p_1^i, p_2^i) within a square box $SB(e_i)$. In the method of computing the compactness (CTC), for each edge e_i, we have to compute all neighboring points q_i (with i, the number of neighboring points of e_i) to find the best point for creating the compactness triangle. Therefore, the complexity of this computation is always at most $N \times i$ (where N is the number of points of the surface) for iteration. In our method: voxel traversal search (VTS), we based on the Woo algorithm [AW87] in order to save the searched time for a neighboring point p_3^i. We aim to find the point that is close to the position of the third point of an equilateral triangle created by e_i. This point is always located on the non-empty voxel crossed on the ray L (if it existed) because the direction of L orthogonal to e_i, at the middle of e_i. If there is no point on L, we dilate from L until finding an appropriate point. For this reason, we do not need to check all neighboring points as computed in the method CTC. Comparing to the method CTC, the processing time of our method is faster (as presented in the table 5.1), but no guarantee exists for the worst case if an important dilatation is required.
The second advantage is that our surface is triangulated after a simplification step (in

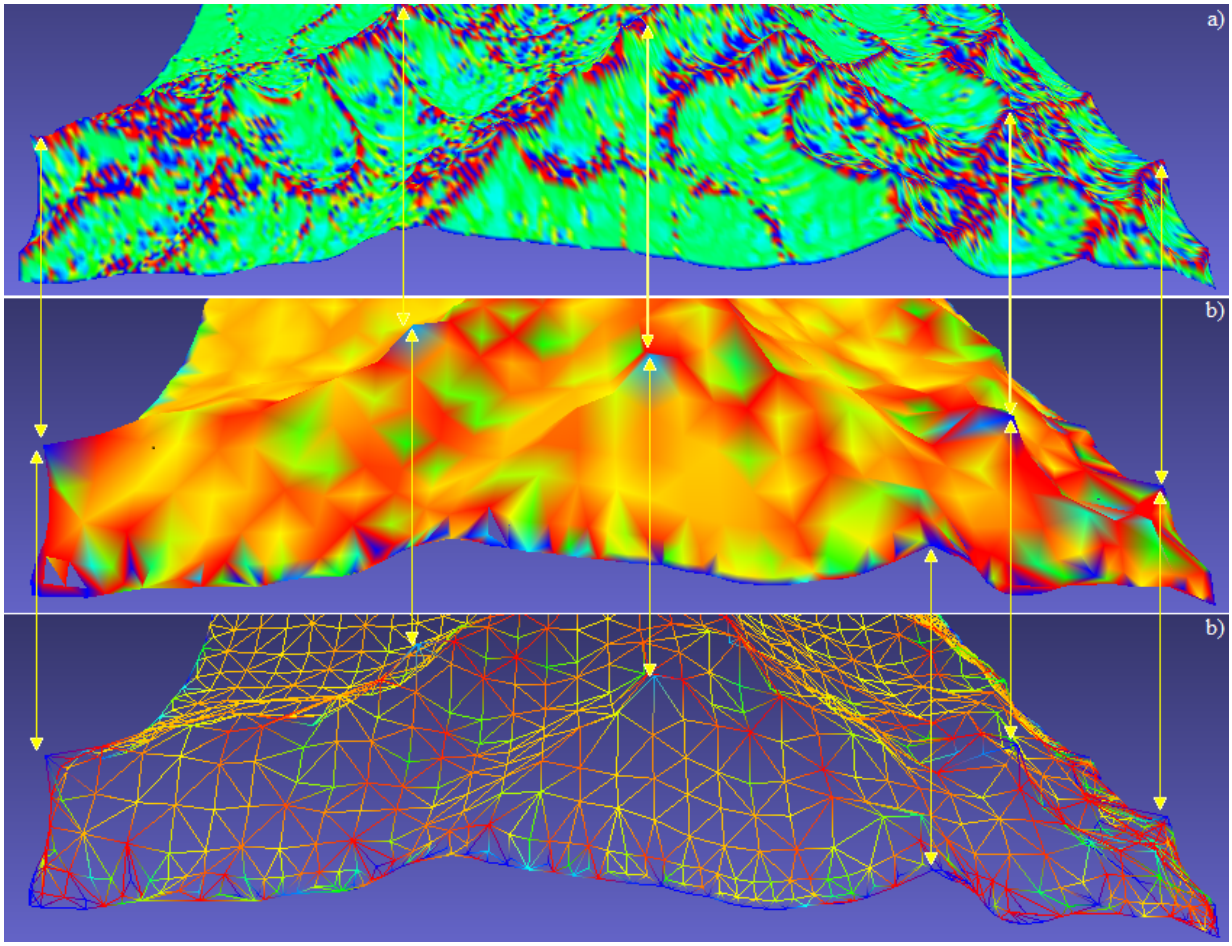

Figure 5.21: a) The input surface of 3D point clouds with 2629 kb. b) After simplifying by using the elaborate method (cell size = 8, $\partial \leq 0.09$) and triangulating, the size of surface: 68 kb; the approximation error between (a) and (b) is Δ_{max}: 0.018; Δ_{avg}: 0.002; the characteristics of the surface are well preserved.

chapter four). After simplifying, a large number of points of the surface has been simplified. Therefore, for each edge e_i, there is only a few neighboring points p_3^i in one side of e_i within $SB(e_i)$. For these advantages, the processing time of our method is very fast, as well as the initial shape of the surfaces is well preserved.

5.7 Conclusion

In this chapter, we presented our method for constructing a triangular mesh of the surface of 3D point clouds defined in the 3D grid. We applied the ideas of the methods introduced in [AW87, FP93] to build our method. The main idea of our method based on the 2D Delaunay triangulation is to use the voxel traversal search. The advantages of our method are presented in the evaluation and discussion. The obtained results have shown that the

processing times are very fast. In fact, after simplifying the surface, a large number of points of the surface have been suppressed. So, the processing time for both methods using two ways of neighboring search are nearly the same (see table 5.1). However, the neighboring search based on the voxel traversal search is more efficient. After simplifying and triangulating a surface, the triangular faces of this surface have varied automatically from the boundary to the inside of the surface. The quality of the triangular mesh is checked by using a tool *Metro* in [CRS98, PGK02]. Therefore, the obtained results are "optimal" geological triangulated surfaces adapted to our goal.

Chapter 6

Conclusion and Future Work

Contents

6.1 Conclusion and contributions **93**

6.1.1 Boundary extraction 94

6.1.2 Boundary simplification 95

6.1.3 Rough simplification 95

6.1.4 Elaborate simplification 96

6.1.5 Surface triangulation 97

6.2 Future work **97**

6.1 Conclusion and contributions

As we mentioned in chapter one, our goal is to obtain a high quality definition of oil reservoirs in order to get the best estimation of the potential petrol they may contain. In this thesis, we have introduced our methods for reconstructing an optimal geological surface from a huge amount of 3D point clouds structured in the 3D grid. The seismic data come from an automatic acquisition; they are processed and exported into a sparse 3D volume as defined by Philippe Verney [Ver09]. We first focused on data processing to simplify the surfaces of 3D point clouds. Then, we have proposed a method for triangulating this surface while preserving the initial shape of the surface. Our works are achieved the following order: the first work is the boundary extraction and simplification (presented in chapter three); the second work is simplifying the surface inside (indicated in chapter four); and the third work is triangulating this surface (performed in chapter five). An overview of our methods is described in figure 6.1:

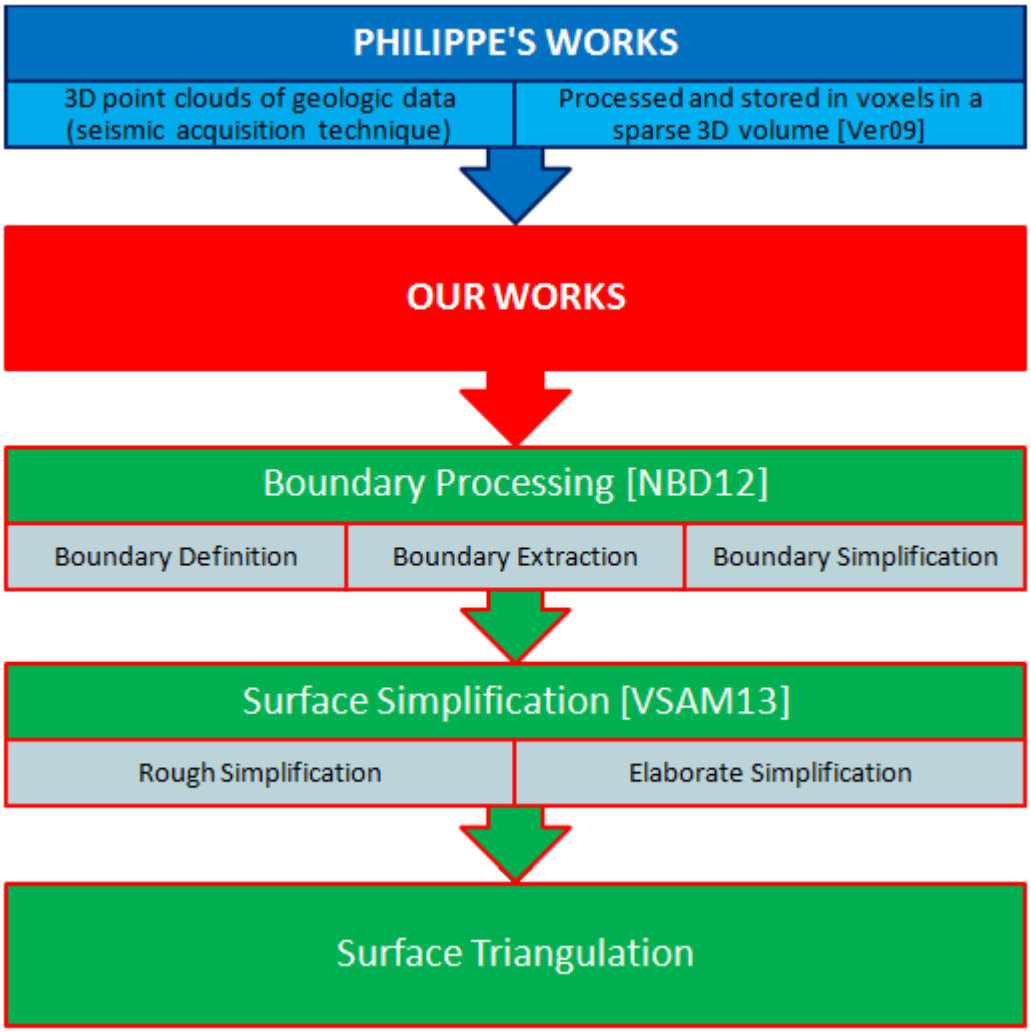

Figure 6.1: Description of our methods for constructing an optimal geological surface.

6.1.1 Boundary extraction

The most important issue of surface simplification of a 3D point cloud is to preserve the original shape of the surface. For this reason, we proposed a method for extracting the boundary of the surface before simplifying it. At first, the 3D point sets (sampled from an elevation surface and structured in a sparse 3D volume) are projected onto a natural 2D grid in x, y plane. Then, we defined an exterior boundary of 2D surface in our context through the definition of the square neighborhood $SN_k(p)$ based on the relationship between a point and its neighbors in a given distance k. The next topological boundary is defined with the classical connectivity (8-connectivity) between the points. Then, we defined a closed discrete $SN_k - curve$. At the end, we had an exterior boundary that was suitable when data are missing.
After defining an exterior boundary of a surface, we proposed a method for extracting this

boundary. This method can be applied to determine all exterior boundaries in the case of dispersive data points distributed in the different regions of the surface. Our method proved more efficient than existing methods.

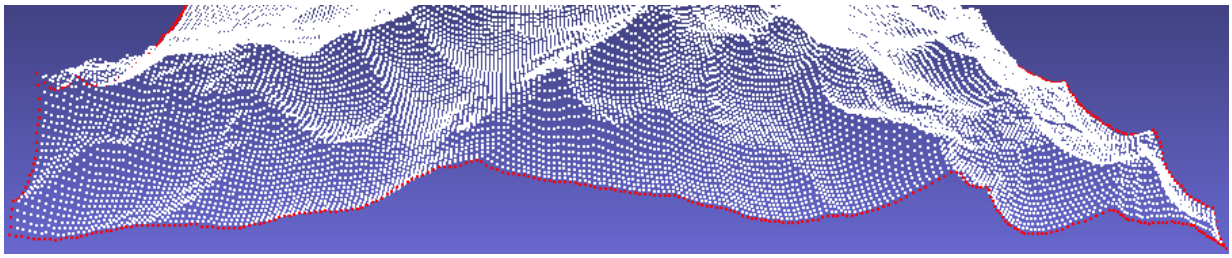

Figure 6.2: Boundary extraction of the geological surface.

6.1.2 Boundary simplification

Our goal aims at simplifying the surface completely. Therefore, after extracting the boundary of a surface, we built a method to simplify it. In fact, this boundary was a 3D exterior boundary of an elevation surface. So we first detected the 2D line segments on the boundary based on a study of the alignment of points by computing the coordinates and the slope between them. Then, we simplified the 3D line segments based on the variation of elevation.
The obtained results of our first work (boundary extraction and simplification) shown that the complexity of algorithms is more efficient than existing methods. Moreover, the initial shape of the surface is also preserved for the next simplification step since the boundaries are kept.

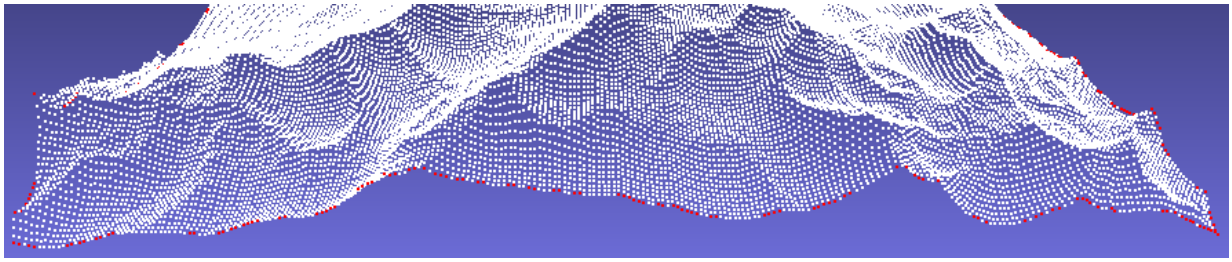

Figure 6.3: Boundary simplification of the geological surface.

6.1.3 Rough simplification

Normally, the running time and memory space of a computer program mostly depends on the data structure, algorithms and the quantity of input data. Our input data points are very large (millions of points). Therefore, we designed a rough simplification in order to reduce the number of input data points (for which a more elaborate simplification cannot

be applied directly because of time and space complexity issues). This step is simple (each cell is replaced by one representative point, except the boundary points), but very useful. It can be justified to adapt the resolution, in case the resolution of the data is too high compared to the expected results. Based on our computing experience, if the cell size less than or equal to three $s \leq 3$ (that is initial cells containing at most 9 voxels), it does not affect the shape of the surface. For this reason, rough simplification can help to reduce the processing time for the next elaborate simplification step.

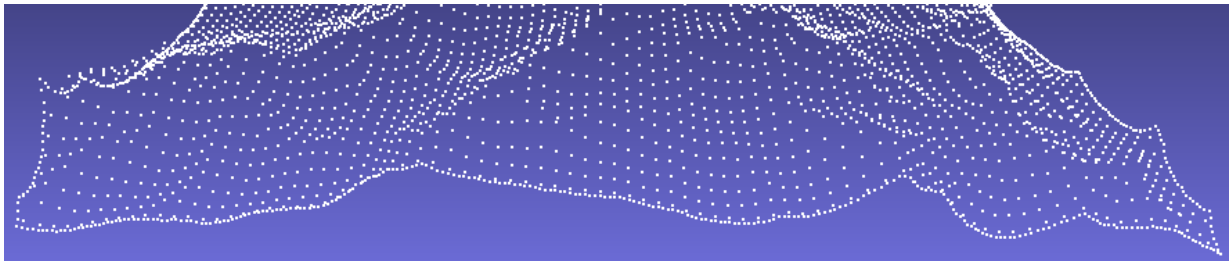

Figure 6.4: Rough simplification of the geological surface.

6.1.4 Elaborate simplification

This step can be applied to simplify completely a surface. We focused on two main points to process the surface directly in a 3D grid. Each cell is subdivided depending on the curvature within that cell; boundary density (if the cell contains boundary points); and subdivision level of its neighboring cells. In order to subdivide a cell according to the boundary density, we have created a boundary density criterion. If a cell (containing boundary points) satisfies the criterion, we based on its size to subdivide and simplify. For inner cells (containing inner points), we based on the flatness criterion. This criterion is created by estimating the curvature in a cell (based on PCA) and taking into account the size of neighboring cells.
We provided the values of ∂ (flatness criterion) in order to adapt the application of users. The running time is affected by choosing these values. If ∂ is small or close to 0, a lot of cells are subdivided and the running time is high; otherwise (∂ is higher or close to 1/3), the running time is faster but the initial shape of the surface can be affected.

Figure 6.5: Elaborate simplification of the geological surface.

6.1.5 Surface triangulation

After simplifying a surface, a large number of points have been removed while preserving the shape of the surface. In order to triangulate this surface, we have proposed a method based on the 2D Delaunay triangulation. After projecting a surface S onto the 2D grid of x, y plane, we triangulated S by using a fast voxel traversal search. Normally, for each edge e_i (endpoints: p_1^i, p_2^i), we have to check all neighboring points in one side of e_i to find an optimal one p_3^i to define a triangle $\triangle(p_1^i, p_2^i, p_3^i)$. This leads to an expensive computation. In our proposed method, we chose the first neighboring point which close to the best point. Therefore, the complexity of our algorithm is linear. We used a 2D Delaunay triangle as a criterion for triangulating a surface and controlling the quality of a triangulated surface.

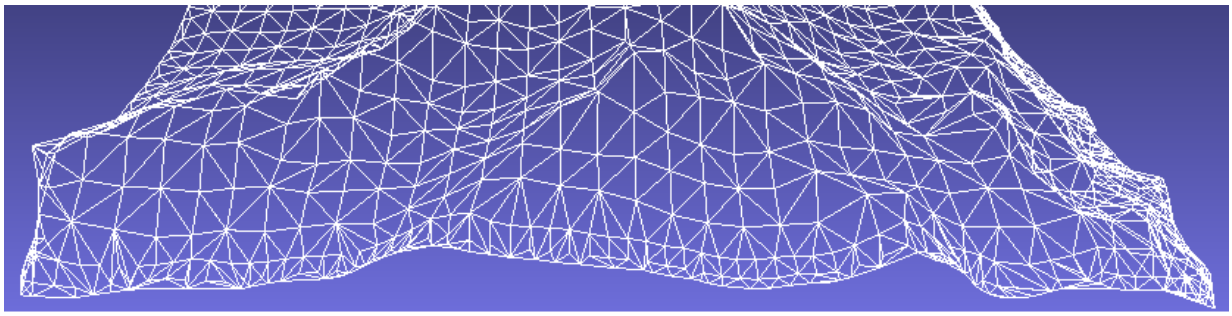

Figure 6.6: Triangulation of the geological surface.

6.2 Future work

The obtained results in this thesis can help us to extend our work in the future. Although the methods for filling the holes of a triangular mesh already existed; but for a 3D point cloud it still seems a challenge and may be a new direction of research. Boundary extraction can be applied to detect the boundary of the holes inside the surface first. Thereafter, a method for filling the holes should be studied in order to obtain a good and smooth surface of 3D point clouds.
At present, the results of our method for surface simplification have adapted the requirements as our expectations. However, each step relies on some thresholds (e.g. cell size, point density, distance between the points or ∂) defined by the users. Therefore, an automatic computing of these thresholds for a good value may be an optional work in the future. This calculation will be a bit slower because of an iterative computation for each step, but it may be adapted to every cases.
We pay attention to the complexity of our algorithm. Nevertheless we did not have the opportunity to test them on very huge data sets and we hope to have soon the opportunity to obtain such sets.

Bibliography

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. *Proceedings of the sixth ACM Symposium on Solid modeling and application*, pages 249–266, 2001. ISBN: 1-58113-366-9.

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. *In Eurographics 1987*, 87:3–10, 1987.

[BDM05] Alexandra Bac, Marc Daniel, and Jean-Louis Maltret. 3d modeling and segmentation with discrete curvatures. *Medical Informatics and Technology*, 9:13–24, 2005. ISBN: 1642-6037.

[BDRT09] A. Bac, M. Daniel, J.-F. Rainaud, and N.-V. Tran. Surface improvement for reservoir modelling. *MAMERN09: 3^{rd} International Conference on Approximation Methods and Numerical Modelling in Environment and Natural Resources Pau*, pages 163–167, June 2009.

[Bel08] David Belton. Improving and extending the information on principal component analysis for local neighborhoods in 3D point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:477–484, 2008.

[BMR+99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, Oct-Dec 1999.

[BP00] Marshall Bern and Paul Plassmann. Mesh generation. *Handbook of Computational Geometry. Elsevier Science*, pages 291–322, 2000.

[BPK+07] M. Botsch, M. Pauly, L. Kobbelt, P. Alliez, B. Lévy, S. Bischoff, and C. Rossl. Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH Course Notes*, pages 11–159, 2007. Award: Best course notes for a revised course.

[BSP+04] S. Brandel, S. Schneider, M. Perrin, N. Guiard, J.-F. Rainaud, P. Lienhardt, and Y. Bertrand. Automatic building of structured geological models. *Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 59–69, 2004. ISBN: 3-905673-55-X.

[BTD07] Alexandra Bac, Nam-Van Tran, and Marc Daniel. A hybrid simplification algorithm for triangular mesh. *GraphiCon'2007. International Conference on Computer Graphics and Vision*, pages 17–24, 2007.

[BTD08] Alexandra Bac, Nam-Van Tran, and Marc Daniel. A multistep approach to restoration of locally undersampled mesh. *GMP'08 Proceedings of the* 5^{th} *international conference on Advances in geometric modeling and processing*, 4975:272–289, 2008. ISBN: 978-3-540-79245-1.

[CG09] C. Chuon and S. Guha. Volume cost based mesh simplification. *Computer Graphics, Imaging and Visualization, CGIV'09. Sixth International Conference on IEEE*, pages 164–169, 2009.

[Cha07] William Y. Chang. Surface reconstruction from points. Technical report, For the Research Exam, February 2007. UCSD CSE CS2008-0922.

[CM91] Jean-Marc Chassery and Annick Montanvert. *Géométrie discrète en analyse d'images*. Editions Hermès, Paris, 1991. ISBN: 2-86601-271-2.

[Cou13] ISTI Italian National Research Council. Meshlab. `http://meshlab.sourceforge.net/`, 2013. [Online; accessed 21-July-2013].

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *The Eurographics Association 1998*, 17(2):167–174, 1998.

[DDW11] Tamal K. Dey, Ramsay Dyer, and Lei Wang. Localized cocone surface reconstruction. *Computers Graphics Shape Modeling International (SMI)*, 35:483–491, 2011.

[DE96] Matthew T. Dickerson and David Eppstein. Algorithms for proximity problems in higher dimensions. *Journal Computational Geometry, Theory and Applications*, 5(5):277–291, May 1996.

[DG03] T. K. Dey and S. Goswami. Tight cocone: A water-tight surface reconstructor. *in Proceedings of the 8th ACM Symposium on Solid Modeling and Applications*, pages 127–134, 2003. ISBN: 1-58113-706-0.

[DP73] D. H. Douglas and T. K. Peucker. Algorithms for the reductions of the number of points required to represent a digitized line or its caricature. *The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DWLT90] David P. Dobkin, Allan R. Wilks, Silvio V. F. Levy, and William P. Thurston. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, 9(4):389–423, 1990.

[DZ08] V. Domiter and B. Zalik. Sweep-line algorithm for constrained Delaunay triangulation. *International journal of Geographical Information Science*, 22(4):449–462, 2008.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M.Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *SIGGRAPH '95 Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 173-182, 1995. ISBN: 0-89791-701-4.

[Ede01] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge monographs on applied and computational mathematics, Cambridge University Press, USA, 2001. ISBN: 0-521-79309-2.

[EKS83] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[FB99] Pascal J. Frey and Houman Borouchaki. Surface mesh quality evaluation. *International Journal for Numerical Method in Engineering*, 44(45):101–118, 1999.

[FD07] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[FP93] T.-P. Fang and L. A. Piegl. Delaunay triangulation using a uniform grid. *IEEE Computer Graphics and Applications*, 13(3):36–47, 1993.

[FXC07] LEVET Florian, GRANIER Xavier, and SCHLICK Christophe. Triangulation of uniform particle systems: its application to the implicit surface texturing. *In 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision WSCG 2007*, pages 271–278, 2007. ISBN: 978-80-86943-98-5.

[Gar99] Michael Garland. *Quadric-Based Polygonal Surface Simplification.* Phd thesis, Carnegie Mellon University, 1999.

[GB97] P. L. George and H. Borouchaki. *Triangulation de Delaunay et Maillage.* Hermès Science Publications, Paris, 1997. ISBN: 9782746233607.

[GBGS98] A. Garrido, N. Perez De La Blanca, and M. Garcia-Silvente. Boundary simplification using a multiscale dominant-point detection algorithm. *Pattern Recognition*, 31(6):791–804, 1998.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997. ISBN: 0-89791-896-7.

[Gue97] Andre Gueziec. Surface simplification inside a tolerance volume. Technical report, IBM Research Report, May 1997. RC 20440, NY 10598.

[Gui06] Nicolas Guiard. *Construction de modèles géologiques 3D par co-raffinement de surfaces.* Phd thesis, Ecole des Mines de Paris, 2006.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchampy, John McDonaldz, and Werner Stuetzlez. Surface reconstruction from unorganized points. *Proceeding SIGGRAPH '92 Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 26:71–78, 1992.

[Hop94] Hugues Hoppe. *Surface Reconstruction from Unorganized Points*. Phd thesis, University of Washington, 1994.

[Jol02] I. T. Jolliffe. *Principal Component Analysis*. Springer, USA, 2002. ISBN: 0-387-95442-2.

[Jr.06] Richard H. Groshong Jr. *3-D Structural Geology, A Practical Guide to Quantitative Surface and Subsurface Map Interpretation*. Springer, Second Edition, USA, 2006. ISBN: 3-540-31054-1.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. *SGP '06 Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, 2006. ISBN: 3-905673-36-3.

[KL06] Seok-Il Kim and Rixie Li. Complete 3D surface reconstruction from unstructured point cloud. *Journal of Mechanical Science and Technology*, 20(12):2034–2042, 2006.

[Kle97] Reinhard Klein. Construction of the constrained Delaunay triangulation of a polygonal domain. *In CAD Systems Development. Springer Berlin Heidelberg*, pages 313–326, 1997. ISBN: 978-3-642-60718-9.

[KNSS09] Evangelos Kalogerakis, Derek Nowrouzezahrai, Patricio Simari, and Karan Singh. Extracting lines of curvature from noisy point clouds. *Journal Computer-Aided Design archive*, 41(4):282–292, January 2009.

[LJ08] Pai-Feng Lee and Bin-Shyan Jong. Point-based simplification algorithm. *WSEAS Transactions on Computer Research*, 3(1):61–66, January 2008.

[LL86] D. T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete and Computational Geometry*, 1(1):201–217, 1986.

[LS80] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[LT07] Ying-Zhe Lue and Yi-Hsing Tseng. Surface reconstruction from LiDAR point cloud data with a surface growing algorithm. *Proceedings of the* 28^{th} *Asia Conference on Remote Sensing*, pages 1944–1949, 2007. ISBN: 978-1-61567-365-0.

[LTW04] HongWei Lin, Chiew Lan Tai, and Guo Jin Wang. A mesh reconstruction algorithm driven by an intrinsic property of a point cloud. *Computer-Aided Design*, 36(1):1–9, January 2004.

[Ma11] Ji Ma. *Surface reconstruction from unorganized point cloud data via progressive local mesh matching*. Phd thesis, School of Graduate and Postdoctoral Studies, The Univerity of Western Ontario, 2011.

[Mas10] Laura Silveira Mastella. *Semantic exploitation of engineering models: application to petroleum reservoir models*. Phd thesis, l'École nationale supérieure des mines de Paris, 2010.

[Mau02] Pavvel Maur. *Delaunay Triangulation in 3D*. Phd thesis, University of West Bohemia in Pilsen, Czech Republic, 2002.

[MD04] Carsten Moenning and Neil A. Dodgson. Intrinsic point cloud simplification. *International Conference 14th GraphiCon 2004*, 2004.

[Mei11] Martijn Meijers. Simultaneous and topologically-safe line simplification for a variable-scale planar partition. *The* 14^{th} *AGILE International Conference on Geographic Information Science*, 1:337–358, 2011.

[MRB09] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On fast surface reconstruction methods for large and noisy point clouds. *ICRA'09 Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3218–3223, 2009. ISBN: 978-1-4244-2788-8.

[MVF04] B. Mederos, L. Velho, and L. H. Figueiredo. Smooth surface reconstruction from noisy clouds. *Journal of the Brazilian Computer Society*, 9(3):52–66, April 2004.

[MWZ10] R. Morales, Y. Wang, and Z. Zhang. Unstructured point cloud surface denoising and decimation using distance RBF K-rearest neighbor kernel. *Proceedings of the Advances in multimedia information processing*, 6298:214–225, 2010. ISBN: 978-3-642-15695-3.

[NBD12] Van-Sinh Nguyen, Alexandra Bac, and Marc Daniel. Boundary extraction and simplification of a surface defined by a sparse 3D volume. *Proceeding of the third international symposium on information and communication technology SoICT 2012*, pages 115–124, August 2012. ACM-ISBN: 978-1-4503-1232-5.

[Ngu12] Van-Sinh Nguyen. Shape of a voxel elevation surface, boundary extraction and simplification. *In 9 èmes Journées des doctorants du LSIS*, pages 222–234, June 2012.

[OBS05] Y. Ohtake, A. Belyaev, and H. P. Seidel. An integrating approach to meshing scattered point data. *Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 61–69, 2005. ISBN: 1-59593-015-9.

[OVBP11] E. Ovreiu, S. Valette, V. Buzuloiu, and R. Prost. Mesh simplification using an accurate measured quadratic error. *Signals, Circuits and Systems (ISSCS), 2011 10th International Symposium on IEEE*, pages 1–4, 2011. ISBN: 978-1-61284-944-7.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. *Visualization VIS IEEE 2002*, pages 163–170, 2002. ISBN: 0-7803-7498-3.

[SC06] Whenzhong Shi and Chui Kwan Cheung. Performance evaluation of line simplification algorithms of vector generalization. *The Cartographic Journal*, 43(1):27–44, 2006.

[SF09] Hao Song and Hsi-Yung Feng. A progressive point cloud simplification algorithm with preserved sharp edge data. *The International Journal of Advanced Manufacturing Technology*, 45(5):583–592, 2009.

[SKM11] Sait Suer, Sinan Kockara, and Mutlu Mete. An improved boundary detection in dermoscopy images for density based clustering. *Proceedings of the Eighth Annual MCBIOS Conference. Computational Biology and Bioinformatics for a New Decade*, 12(10):S12, 2011.

[SLK05] Jae-Young Sim, Sang-Uk Lee, and Chang-Su Kim. Construction of regular 3D point clouds using octree partitioning and resampling. *Circuits and Systems. ISCAS 2005. IEEE International Symposium*, 2:956–959, May 2005.

[Slo93] S. W. Sloan. A fast algorithm for generating constrained Delaunay triangulation. *Computers and Structures*, 47(3):441–450, 1993.

[Smi90] Michiel Smid. Maintaining the minimal distance of a point set in less than linear time. *ESPRIT II Basic Research Actions Program, under contract No.3075, Project ALCOM*, pages 33–44, June 1990.

[SS07] Aparajithan Sampath and Jie Shan. Building boundary tracing and regularization from airborne lidar point clouds. *Photogrammetric engineering and remote sensing*, 73(7):805–812, 2007.

[Tra08] Nam-Van Tran. *Traitement de surfaces triangulées pour la construction des modèles geologique structuraux*. Phd thesis, Université de la Méditerranée, 2008.

[Ver09] Philippe Verney. *Interprétation géologique de données sismiques par une méthode supervisée basée sur la vision cognitive*. Phd thesis, École Nationale Supérieure des Mines de Paris, 2009.

[VSAM13] Nguyen Van-Sinh, Bac Alexandra, and Daniel Marc. Simplification of 3D point clouds sampled from elevation surfaces. *21st International Conference on Computer Graphics, Visualization and Computer Vision WSCG 2013*, pages 60–69, June 2013. ISBN: 978-80-86943-75-6.

[WCZ+08] R.-F. Wang, W.-Z. Chen, S.-Y. Zhang, Y. Zhang, and X.-Z. Ye. Similarity-based denoising of point-sampled surfaces. *Journal of Zhejiang University SCIENCE A*, 9(6):807–815, June 2008.

[Wei08] Shen Wei. Building boundary extraction based on LiDAR point clouds data. *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:157–161, 2008.

[Wik13] Wikipedia. Delaunay Triangulations. http://en.wikipedia.org/wiki/Delaunay_triangulation, 2013. [Online; accessed 07-March-2013].

[XXFJ08] Huang Xianfeng, Cheng Xiaoguang, Zhang Fan, and Gong Jianya. Side ratio constrain based precise boundary tracing algorithm for discrete point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:349–354, 2008.

[YLL+07] M. Yoon, Y. Lee, S. Lee, I. Ivrissimtzis, and H. P. Seidel. Surface and normal ensembles for surface reconstruction. *Journal Computer Aided Design*, 39(5):408–420, 2007.

[YsW06] Zhiwen Yu and Hau san Wong. An efficient local clustering approach for simplification of 3D point-based computer graphics models. *IEEE International Conference on Multimedia and Expo*, pages 2065–2068, 2006. ISBN: 1-4244-0367-7.

[YZY+06] Y.-J. Yang, H. Zhang, J. H. Yong, W. Zeng, J.-C. Paul, and J. Sun. Contrained Delaunay triangulation using Delaunay visibility. *ISVC 2006, LNCS 4291*, 4291:682–691, 2006. ISBN: 978-3-540-48628-2.

[ZAMZ11] M. Zhang, N. Anwer, L. Mathieu, and H. B. Zhao. A discrete geometry framework for geometrical product specifications. *CIRP Design Conference*, pages 142–148, March 2011.

[ZG04] M. Zwicker and C. Gotsman. Meshing point clouds using spherical parameterization. *Eurographics Symposium on Point-Based Graphics*, pages 173–180, 2004. ISBN: 3-905673-09-6.

[ZG10] Y. J. Zhang and L. L. Ge. A robust and efficient method for direct projection on point sampled surface. *International Journal of Precision Engineering and Manufacturing*, 11(1):145–155, 2010.

Résumé

L'Infographie est un domaine important de l'informatique largement utilisé dans le monde réel. L'arrivée des cartes graphiques grand public rapides et peu coûteuses a conduit à un important besoin de programmation des différentes tâches géométriques pour les applications, y compris les modèles informatiques, la visualisation scientifique, l'analyse d'images médicales, la simulation et les mondes virtuels. Les types d'applications évoluent pour profiter des avancées en modélisation géométrique (basée sur des modèles mathématiques). Reconstruire des surfaces avec des données provenant d'une technique d'acquisition automatique entraîne toujours le problème de la masse des données acquises. Cela implique que les procédés habituels ne peuvent pas être appliqués directement. Par conséquent, un processus de réduction des données est indispensable. Un algorithme efficace pour un traitement rapide préservant le modèle original est un outil précieux pour la construction d'une surface optimale et la gestion des données complexes.

Dans cette thèse, nous présentons des méthodes pour construire une surface géologique optimale à partir d'une quantité énorme de points 3D extraits de cubes sismiques. Appliquer le processus à l'ensemble des points induit un risque important de contraction de la surface de sorte que l'extraction de la frontière initiale est une étape importante permettant une simplification à l'intérieur de la surface. La forme globale de la surface sera alors mieux respectée pour la reconstruction de la surface triangulaire finale. Nos propositions sont basées sur la régularité des données qui permet, même si des données sont manquantes, d'obtenir facilement les informations de voisinage.

Tout d'abord, nous présentons une nouvelle méthode pour extraire et simplifier la frontière d'une surface d'élévation définie par un ensemble de voxels dans un grand volume 3D où des données sont manquantes. Deuxièmement, une méthode pour simplifier la surface à l'intérieur de sa frontière est présentée. Elle comprend une étape de simplification grossière optionnelle suivie par une étape plus fine basée sur l'étude des courbures. Nous tenons également compte du fait que la densité de données doit changer graduellement afin de recevoir à la dernière étape d'une surface triangulée avec de meilleurs triangles. Troisièmement, nous avons proposé une nouvelle méthode rapide pour trianguler la surface après simplification.

Mots-clés: Nuages de Points, Extraction de la Frontière, Simplification de la Frontière, Simplification de surfaces, Triangulation.